



Universidad Autónoma de San Luis Potosí
Facultad de Ingeniería
Centro de Investigación y Estudios de Posgrado

**Modelo de condensación de conocimiento para
soporte en la localización de experiencia de la fase
de codificación durante el desarrollo de software**

T E S I S

Que para obtener el grado de:

Doctorado en Ciencias de la Computación

Presenta:

José Ramón Martínez García

Asesor:

Dr. Francisco Edgar Castillo Barrera

Co-Asesor:

Dr. Ramón René Palacio Cinco

San Luis Potosí, S. L. P.

Mayo de 2021





Autonomous University of San Luis Potosí
Faculty of Engineering
Research and Graduate Studies Center

Knowledge Condensation Model to Support Expertise Location in the Coding Phase during the Software Development Process

T H E S I S

In order to obtain the:

Doctor of Computer Science

Present:

José Ramón Martínez García

Advisor:

Dr. Francisco Edgar Castillo Barrera

Co-Advisor:

Dr. Ramón René Palacio Cinco

San Luis Potosí, S. L. P.

May 2021



Abstract

Finding relevant expertise is a critical need in software organizations since developers use it to support their knowledge needs. Developers continuously need knowledge beyond the one they already possess; this means that they need expertise —characteristics and skills that make an individual suitable to solve problems fast and effectively.

In software organizations, expertise can be found in architectural knowledge (AK), which incorporates the experience and problem reasoning of human resources through two primary sources: experts and artifacts; consequently, know how to model it could grant access to the best practices, training practices, and data across applications; this has generated considerable interest by organizations, which seek to develop formal approaches to condensate the available knowledge from different sources in a systematic manner.

The knowledge condensation concept comes from an agile development environment where tacit knowledge is preferred —the agile method states that face-to-face conversations are the most effective way to share knowledge. Moreover, it is only documented what a team considers sufficient to understand the project. A little amount of knowledge becomes explicit and usually stays in the log files of unstructured textual and electronic media (UTEM); over time, this knowledge loses meaning and context; thus, it is prone to vaporize. Therefore, knowledge condensation aims to classify, retrieve, and share valuable knowledge among stakeholders in an unsuitable form for its recovery.

This thesis presents a knowledge condensation model to support the expertise localization process during software development. The knowledge condensation model is made up of 3 main modules: Formal grammar, Semantic Knowledge, and Expertise tools. In the formal grammar module, an approach is made to formalism to describe how developers store and share their knowledge. An architectural knowledge model is proposed in the semantic knowledge module, which is implemented in an ontology for the coding phase in software development. Finally, in the module of expertise tools, two prototypes were developed that implement the ontology elements developed as part of the implementation of the semantic knowledge module.

Resumen

La localización del expertise es una necesidad crítica en las organizaciones de software. Los desarrolladores constantemente necesitan conocimiento más allá del que ellos poseen. En este sentido el expertise representa las características y habilidades que hacen a un individuo adecuado para resolver problemas o dudas de una manera rápida y efectiva.

En las organizaciones de software el expertise se puede encontrar en el conocimiento arquitectónico (AK por sus siglas en inglés), este incorpora la experiencia y el razonamiento para resolver problemas a través de sus dos fuentes principales: expertos y artefactos. Saber cómo modelarlo podría otorgar acceso a las mejores prácticas de capacitación y datos en todas las aplicaciones; esto ha generado un interés considerable por parte de las organizaciones, que buscan desarrollar enfoques formales para condensar el conocimiento disponible de diferentes fuentes de manera sistemática.

El concepto de condensación de conocimiento proviene de un entorno de desarrollo ágil donde se prefiere el conocimiento tácito; el método ágil establece que las conversaciones cara a cara son la forma más efectiva de compartir conocimiento. Además, en los ambientes ágiles solo se documenta la información que un equipo considera suficiente para comprender el proyecto. Por lo tanto, muy poco conocimiento se vuelve explícito y generalmente permanece en los archivos de registro de los medios textuales y electrónicos no estructurados (UTEM por sus siglas en inglés); con el tiempo, este conocimiento pierde significado y contexto lo cual hace que sea propenso a vaporizarse. En este sentido, la condensación de conocimientos tiene como objetivo clasificar, recuperar y compartir conocimientos valiosos entre los interesados de forma inadecuada para su recuperación.

Esta tesis presenta un modelo de condensación de conocimientos para respaldar el proceso de localización de expertise durante el desarrollo de software. El modelo de condensación de conocimiento esta conformado por 3 modulos principales: Gramatica formal, Conocimiento semantico y Herramientas de expertise. En el modulo de gramatica formal se hace un acercamiento a un formalismo para describir la manera en que lo desarrolladores almacenan y comparten su conocimiento. En el modulo de conocimiento semantico se plantea un modelo de conocimiento arquitectonico, el cual esta implementado en una ontología para la fase de codificación en el desarrollo de software. Finalmente en el modulo de herramientas de expertise se desarrollaron dos prototipos que implementan los elementos de la ontología desarrollada como parte de la implementación del modulo de conocimiento semantico.

A mis padres por su gran apoyo y comprensión durante todo este tiempo. . .

Acknowledgements

Quiero agradecer a mi asesor el Dr. Francisco Edgar Castillo Barrera por haber aceptado participar en este proyecto, por sus comentarios y sugerencias que fueron de gran ayuda para lograr los objetivos planteados. Por introducirme al interesante mundo de la web semántica y representación del conocimiento.

A mi co-asesor el Dr. Ramón René Palacio Cinco por haberme guiado durante todos estos años. Gracias por sus consejos tanto académicos como personales. Agradezco me haya introducido al mundo de la academia y la investigación.

A los miembros de comité de tesis, la Dra. Sandra Edith Nava Muñoz, por sus contribuciones y comentarios a este proyecto de tesis. Gracias al Dr. Juan Carlos Cuevas Tello, por su apoyo académico durante estos cuatro años y además por recordarme el amor por el basquetbol.

A mis compañeros de doctorado y las todas las personas que de alguna manera u otra contribuyeron un poco e hicieron más ameno este ciclo de cuatro años. Agradezco las pláticas, tazas de café, comidas y las desveladas que tuvimos mientras estuve en el doctorado.

Quiero agradecer a mi familia, esto logro que he conseguido es de ustedes. Gracias por su apoyo incondicional, por sus platicas de aliento cuando más lo necesitaba y nunca dejarme solo a pesar de la distancia y la diferencia de horario.

Al consejo de Ciencia y Tecnología (CONACYT) por darme los recursos para realizar mis estudios de doctorado.

Table of contents

List of figures	x
List of tables	xii
List of acronyms	xiii
Introduction	1
Motivation	2
Research question	3
Research Goals	3
General	3
Specific	3
Research Contributions	4
Research Methodology	4
Thesis Outline	5
1 Theoretical Framework	7
1.1 Software Development	7
1.1.1 Architectural Knowledge (AK)	7
1.1.2 Role of knowledge in Software Development	8
1.2 Agile Software Development	8
1.2.1 Knowledge reuse in Agile Software Development	8
1.2.2 Expertise location	9
1.3 Knowledge Management	9
1.3.1 Knowledge condensation the next step	10
1.4 Knowledge condensation approaches	10
1.4.1 Ontologies	10
1.4.2 Systems and other applications	15
1.5 Chapter Summary	17
2 Understanding expertise location	18
2.1 Focus Group Study	18
2.1.1 Participants	19

2.1.2	Data Analysis	19
2.2	Expertise Location Mapping	19
2.3	Comparative Analysis	20
2.3.1	Knowledge Sources	21
2.3.2	Knowledge Needs	21
2.3.3	Knowledge Problems	22
2.4	Chapter Summary	23
3	Knowledge Condensation Model	24
3.1	Formal Grammar Module	25
3.1.1	Knowledge Profile Creation	26
3.1.2	Knowledge Search	27
3.1.3	Knowledge Update	28
3.1.4	Knowledge Construction	28
3.2	Semantic Knowledge Module	28
3.2.1	An ontology for the coding phase during the Software Development Process	30
3.3	Expertise Tools Module	37
3.3.1	ExCap: A tool to capture expertise from developers	37
3.3.2	B4U an extension to capture bookmarks	40
3.3.3	Knowledge Condensation in Software Development Scenario	42
3.4	Chapter Summary	42
4	ROntDev: A methodology for developing ontologies	44
4.1	Specification Phase	44
4.1.1	Knowledge Acquisition	45
4.1.2	Requirement Specification	45
4.2	Modeling Phase	45
4.2.1	Integration	46
4.2.2	OWL Modeling	47
4.2.3	Naming Conventions	48
4.3	Formalization phase	48
4.3.1	Implementation	48
4.3.2	Properties	49
4.4	Evaluation phase	50
4.4.1	Verification	50
4.4.2	Validation	51
4.5	Maintenance	52
4.6	Chapter Summary	52

5	Evaluation of the Knowledge Condensation model	54
5.1	Method	54
5.1.1	Objective	54
5.1.2	Participants	54
5.1.3	Instrumentation	55
5.1.4	Procedure	56
5.1.5	Variables and Hypothesis	57
5.2	Results	58
5.3	Discussion	60
5.4	Chapter Summary	62
	Conclusions	63
	Appendix A Focus Group Study Interview Guidelines	67
	Appendix B Guidelines the ExCap interaction	70
	Appendix C Guidelines for the B4U extension interaction	75
	References	82

List of figures

1	Research methodology.	4
2.1	Research method for the expertise location understanding	18
2.2	Focus group affinity diagram	19
2.3	Expertise location process in software development.	20
3.1	Knowledge Condensation Model	25
3.2	Semantic knowledge model of architectural knowledge (AK).	29
3.3	Methontology Framework life cycle[85].	30
3.4	Taxonomy of Knowledge Expertise in code phase.	33
3.5	Coding phase expertise ontology: screenshot of main classes developed in Protégé.	34
3.6	Output in Protégé using Pellet OWL-DL Reasoner	35
3.7	Scenario description	36
3.8	Ontology instances example in Protégé.	36
3.9	Competency Question example with Manchester OWL Syntax in Protégé.	37
3.10	Architecture ExCap tool.	38
3.11	Developers semantic incorporation in ExCap.	39
3.12	Project semantic incorporation in ExCap.	39
3.13	Artifacts semantic incorporation in ExCap.	40
3.14	B4U extension user view.	41
3.15	B4U extension login view.	41
3.16	Expertise Condensation Scenario	42
4.1	ROntDev ontology development life cycle.	44
4.2	ORSD fragment of an ontology for the software development process.	46
4.3	Representation example of classes, individuals, and properties of an ontology for software development process.	47
4.4	Representation example in Protégé of classes, individuals, and properties of an ontology for software development process.	49
4.5	Screenshot of the properties defined in Protégé.	50
4.6	Verification process using a reasoner in Protégé	51
4.7	Reasoner output in Protégé using Pellet.	51

4.8	Competency Questions section in ORSD	52
4.9	Competency Questions example with Manchester OWL Syntax in Protégé	52
5.1	Participants Distribution	55
5.2	Box Diagram of the evaluation results	58

List of tables

1.1	Ontologies proposed for the software engineering domain	12
1.2	A comparison between method and methodologies for building ontologies .	14
2.1	Knowledge sources used by developers during the expertise location	22
2.2	Knowledge needs that lead developers to perform expertise location	22
2.3	Knowledge problems identified during the expertise location	23
3.1	Requirements to support expertise location in software development	24
3.2	Elements for the user profile building	26
3.3	Ontology Requirements Document Template Fragment	32
3.4	Ontology properties, ranges and domains	34
3.5	Competency Questions in Natural Language	35
5.1	Descriptive statistics of the TAM evaluation results	59
5.2	Mann Whitney test comparison	60

List of acronyms

AMOD Agile Methodology for Ontology Development	14
AK <i>Architectural Knowledge</i>	1
B4U Bookmarks for Us	40
CB Content Based	13
C-DOM Co-designing Ontologies Methodology	15
CQ Competency Questions	13
DILIGENT Distributed Loosely-controlled and evolvInG Engineering of oNTologies .	14
DOGMA Developing Ontology-Grounded Methods and Applications	15
ExCap Expertise Capture	37
FTP File Transfer Protocol	38
HTTP Hypertext Transfer Protocol	38
IDEF5 Integrated Definition for Ontology Description Capture	28
KM Knowledge Management	9
NeOn Networked Ontologies	14
OASys Ontology for Autonomous Systems	12

OntoSoft Ontology Software	15
ORSD Ontology Requirement Specification Document	31
OWL Ontology Web Language	11
RDF Resource Description File	33
REFSENO Representation Formalism for Software Engineering	12
SABiO Systematic Approach for Building Ontologies	12
SAMOD Simplified Agile Methodology for Ontology Development	15
SCIM Software Centric Innovation Methodology	15
SWEBOK Software Engineering Body of Knowledge	45
UML Unified Modeling Language	2
UTEM <i>unstructured text and electronic media</i>	10
XML Extensible Mark Language	33

Introduction

Knowledge in software organizations plays a vital role in the improvement and success of the software development process, which consists of activities that demand the application of knowledge representations for their understand and execution [1–3]. These knowledge representations involve different situations such as: (1) in particular projects, developers need to use unfamiliar technologies; thus, developers need to prepare themselves to use those technologies [4]; (2) during a project, developers regularly make both technical and managerial decisions [5, 6]; (3) frequently during projects, developers need to solve problems (e.g., bug fixing, requirement misunderstanding, programming errors, architecture design and others) [7].

Eventually, during a software development project, developers may find themselves in situations where they need knowledge beyond what they have; which forces them to perform *expertise* location, where *expertise* is high-level knowledge appropriate for a particular circumstance, and *expertise* location is the process of finding the right person to answer a question or a resource that helps to support the situation [8, 9]. In software organizations, *expertise* can be found in *Architectural Knowledge* (AK), which refers to the elements employed to construct an architectural design (e.g., structures, properties, and relationships) and the design decision and rationale used to attain architectural solutions [10]. AK incorporates the experience and problem reasoning of human resources into the organizational culture through two primary sources: artifacts and experts. Artifacts denote physical and digital documents (e.g., requirements, vision) and source code [11, 12], while experts are software developers specialized in problem-solving tasks at different stages of the process [9, 13].

Each time developers perform expertise location, they accumulate *expertise* about new technologies used, decisions, and problems solved. The *expertise* is a valuable knowledge resource for the organization, Mohagheghi and Conradi observed that knowledge reuse improves productivity and increase software quality [14]. However, the features of current development paradigms such as Agile development hinder expertise reuse. Agile development is a paradigm which broadly focuses on tacit knowledge [15]. In their manifesto¹, the agile development paradigm states that the most effective way to share knowledge within a development project is through face-to-face conversations; moreover, only is documented the information which the team considers sufficient to understand the software to develop. As a result, plenty of tacit knowledge stays unexploited; this knowledge represents the accumulated expertise of developers, and its prone to vaporize over time. Knowledge vaporization is conceptualized as

¹<https://agilemanifesto.org/>

the loss of AK due to its poor documentation during a software development project; it also involves the lack of expertise when the developer leaves the company or retires from a project [16].

Expertise location is an important task performed during software development projects; therefore, knowing how to model expertise in an explicit form will allow taking advantage of the best practices, training knowledge, and the data across multiple applications in the organization. As a result, it could help to avoid three recurring problems in software development [17, 18]: a) waste of time answering and finding solutions to problems already solved; b) visibility lack of technical solutions and c) knowledge loss due to developers' unavailability. Therefore, organizations must develop formal approaches to condensate knowledge from different sources in a systematic manner. Borrego et al. establish knowledge condensation as the process of capturing and classifying expertise before it loses, where the aim is to ease their retrieve [16].

The presented thesis centers on the software development process. In particular, the expertise location process and the problems caused by the knowledge vaporization. The proposal applies the concept of knowledge condensation into a model, where the aim is to reduce these problems.

The current chapter serves as an introductory description of the motivation, research questions, research goals, the main contributions, and the organization of the rest of the document.

Motivation

Plenty of research in recent years has focused on the management of *expertise*. Proposals attempt to externalize expertise which consists of a transition from tacit knowledge to explicit. These proposals externalize knowledge from one of the two primary sources of *expertise*. Artifacts proposal externalizes the expertise generated by the developers during a software development project. Artifact proposals focus mainly on the source code, which assists programmers with code snippets from sources such as StackOverflow [19–21]. Textual Records is another artifact exploited, which is used by proposals to complement and contextualize informal documents [16, 22]. Finally, Unified Modeling Language (UML) diagrams such as use cases are used by researchers to store architectural design data [23].

Besides that, researchers propose using experts as a source of expertise in software development; these proposals support tasks such as handling a bug report, system design, and problem-solving [5, 24–26].

Even though several researchers have presented proposals to condense the expertise, there is still room for improvement. Artifacts proposals exploit developers' expertise during the software development process with two characteristic limitations: (1) a narrow focus on source code; (2) informal accumulation of knowledge unnoticed by organizations. On the other hand, expert proposals employ developers as a source of expertise; experienced developers often have strategies for a particular problem; in general, these proposals do not grant access to the

artifacts produced by the developers. Therefore, the developers' availability limits access to expertise.

In summary, current proposals have some relevant challenges to address. First, proposals that condensate knowledge must grant access to both individuals with particular expertise and their artifacts. This access will ensure the availability of the required resources, even if the provider is not available, and reduce interpersonal relationship erosion. Second, a proposal should incorporate more sources than just code, like other digital sources (e.g., bookmarks, books, manuals, and tutorials) are frequently used.

Research question

Next, we present the questions derived from the problems found in the literature:

- How to reduce knowledge vaporization in the coding phase during the software development process?
- How developers perform expertise location in software development?
- Which sources does experts use in software development?
- How to link artifacts with persons in software development organizations?

Research Goals

General

To develop and validate a knowledge condensation model to capture expertise from developers during the software development process.

Specific

- To know how is performed expertise location process by the developers and what sources to they use
- To built an ontology for capturing and retrieving expertise in the coding phase during the software development process
- To design and implement a tool to capture expertise and classify expertise
- To validate the knowledge condensation model

Research Contributions

- A Knowledge Expertise Model that support expertise location in the coding phase during the software development process
- A domain ontology for knowledge in the coding phase (artifacts and experts)
- A model to map expertise location process in Software Development
- A methodology for the ontology development process focused on non-expert ontology engineers

Research Methodology

The present PhD thesis followed a research methodology which consists of three phases: **Understanding**, **Model conception**, and **Model validation**. During the **Understanding** phase, we performed activities to understand the expertise location process and the sources that developers use. We propose a design for a knowledge condensation model in the **Model conception** phase; moreover, we evaluated the semantic knowledge module proposed in this work. Finally, in the **Model validation** phase, we develop and evaluate the knowledge condensation model's modules (see Figure 1).

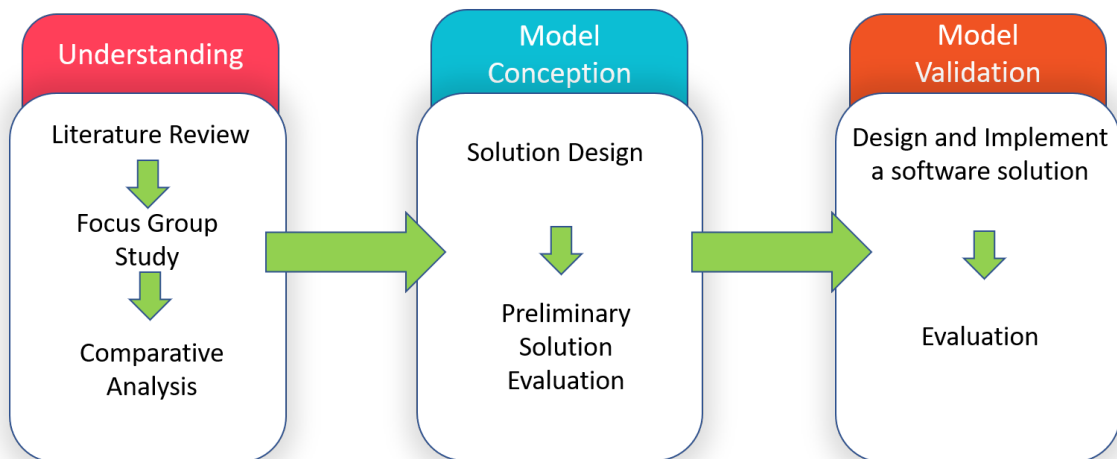


Figure 1 Research methodology.

Next, we describe the activities performed during the phases of the methodology followed:

1. **Literature Review:** This activity's purpose was to write the theoretical framework (see chapter 1) and identify the previous works addressing the same problem. Furthermore, according to the literature reviewed, we find ideas to support the expertise location based on what researchers have done.

2. **Case study:** The case study activity had the purpose of complementing the literature and understanding the expertise location direct from software developers' experience. In chapter 2 we detail the study performed and the results obtained.
3. **Comparative Analysis:** The following activity analysed similar studies to identify a wide range of developers' problems and sources during the expertise location details also in chapter 2.
4. **Solution Design:** The activity consists of designing a knowledge condensation model to support the expertise location. The knowledge condensation model designed uses the case study results and the literature review (see chapter 2).
5. **Preliminary Solution evaluation:** An ontology was developed and was evaluated during the implementation of the semantic knowledge module in the Knowledge condensation model (chapter 3).
6. **Design and Implement a software solution:** This activity consisted of designing and implementing a software solution to capture and classify expertise. The software solution integrates elements from the ontology developed for the semantic knowledge module chapter 3).
7. **Evaluation:** This activity comprised processes the evaluation of the prototype of the software solution and evaluation of the knowledge condensation model (see chapter 5).

Thesis Outline

Introduction

This chapter presents an overview of expertise location process, its relevance within the context of software development process and problems caused by the knowledge vaporization. It formally states the research goals and contributions and provides a chapter by chapter roadmap for the reader.

Chapter 1: Theoretical Framework

This chapter presents a general description of the knowledge vaporization problem and the challenges in supporting expertise location. The approaches found in the literature are described and compared to the knowledge condensation model proposed in this research.

Chapter 2: Understanding Expertise Location Process in Software Development

This chapter describes the activities performed in order to identify the sources consulted by the developers and to map the expertise location process.

Chapter 3: Knowledge Condensation Model

This chapter present a description of the modules that conform the knowledge condensation model presented in this research. Furthermore, we describe some of the implementations developed to validate part of the knowledge condensation model.

Chapter 4: ROntDev: A methodology to develop ontologies

This chapter describe the method produced to develop formal ontologies, which focus is non-experience ontology developers.

Chapter 5: Evaluation of the Knowledge Condensation Model

The chapter describes the evaluation performed to evaluate the knowledge condensation model proposed in this PhD thesis. Moreover, the results and main findings are discussed. Moreover, this chapters describe some of the implementations for the semantic knowledge module, and the expertise tools module.

Conclusions

This chapters summarize the main findings obtained from the implementation of part of knowledge expertise model. The theoretical, and practical implications are described in this chapter. Finally, the chapter addresses the strengths and limitation of the study and concludes with future work.

Appendixes

All additional information describing processes such as focus groups and evaluating the knowledge condensation model is in the appendixes. Appendix A describes the guidelines used for the focus group study. Furthermore, during the evaluation of the knowledge condensation model used two documents with the guidelines for the interaction of the mechanisms to capture and classify knowledge. Appendix B describes the use of the ExCap tool, and Appendix C describes the B4U book plugin interaction.

Chapter 1

Theoretical Framework

1.1 Software Development

Software development is a process that comprises multiple activities that are involved in the creation and maintenance of a software product: computer programming, documentation, test, and bug fixing; it requires constant interaction between stakeholders because it characterizes as a constant change process; many people work in different phases, activities, and projects. The result of a software development process is a product built by a work team's developers' that should solve a client's need raised or identified [27, 28].

Nowadays, markets' competitiveness has caused a constant change to adapt to new conditions: organizations need quality products in less time to reduce their cost. Moreover, organizations must adapt their technologies to suit new environments and build information systems that continuously evolve to meet new requirements [29]. The reuse of knowledge is considered the most accurate way to improve quality and productivity, which will reduce the time and costs of the projects [30]. In software organizations, the reuse of knowledge is known as software reuse and refers to software assets: existing components or knowledge generated by developing previous projects in an organization [31].

1.1.1 Architectural Knowledge (AK)

As mentioned before, software organizations employ software reuse to improve productivity and increase software projects' quality [32]. The software reuse considers the components embedded within the AK. Kruchten et al. define AK as a complement of two elements: (i) Architecture design and (i) Design decisions [33].

(i) Software architecture design is considered a crucial aspect in the software development life-cycle; it represents a description of a system's structure and functionality [34]. (ii) The design decision refers to decisions and rationale within a project; it incorporates implicit and explicit knowledge documented or undocumented. Furthermore, AK also can contain alternative solutions to situations presented during a software development project: technology constraints, business information, and technical information.

1.1.2 Role of knowledge in Software Development

AK in software organizations plays a vital role in the software development process's improvement and success, which consists of activities that demand the application of knowledge representations for their understand and execution. Organizations analyze the role of knowledge to improve software development from two perspectives (i) high-level and (ii) mid-level. (i) From a high-level perspective, the organization's goals are to decrease developing time and increase the projects' quality. In their work, Mohagheghi and Conradi observed that knowledge reuse could improve productivity and increase software quality; by reusing project configurations, metrics, and economic models [14]. (ii) On the other hand, developers use knowledge to support different situations during the software development process.

In their works, Bosch and Clements et al. highlighted the importance of AK recording and that organizations should consider AK an essential asset [35, 36]. Therefore, this has generated interest from researchers in the modeling and management of AK. However the current development methods make difficult the recording of the AK.

1.2 Agile Software Development

During the last decade, the agile development paradigm has become popular among software organizations. Agile is an iterative approach for software development; the main idea is that a team delivers work in small but consumable increments. The team continually evaluates the requirements, plans, and results, so teams have a natural response to change quickly [37].

Organizations with an agile environment base their development process on the agile manifesto¹. The agile manifesto comprises four core values: (1) Individuals and their interactions on processes and tools, (2) Software working on extensive documentation, (3) Collaboration with the client on contract negotiation, and (4) Respond to changes on following a plan.

1.2.1 Knowledge reuse in Agile Software Development

In agile software development, documentation is not a priority, as expressed in agile manifesto, thus, it is reflected in agile software projects with minimal documentation. This situation is caused because the face to face interaction is preferred by developers to clarify doubts or to solve problems. Hence, knowledge is prone to vaporize due to the manifest's principles [38, 16]. Borrego et al. define architectural knowledge vaporization as the loss of artifacts and architectural documents owed to poor documentation [39]. The fact of not having this knowledge generates the following problems [17, 18]: 1) poor understanding of the requirements and technical solutions, 2) knowledge transfer deficiency among developers, 3) evolution and maintenance drawbacks, 4) time wasted by developers while searching for artifacts or experts. These problems cause the increase of time and cost in software development projects [39].

¹<https://agilemanifesto.org/>

1.2.2 Expertise location

To address these problems, developers seek high-level knowledge at a given time (expertise) [9], through which developers could resolve problems or doubts that arise daily during their workday to help them fulfill their activities [40, 41].

During the software development process, regardless of the experience, developers need to perform expertise location which is the process of seeking high-level knowledge at a given time [42, 43]. In the software industry, expertise can be found in AK through their two primary sources: (1) artifacts, which comprises documentation [11] (e.g. requirements document, vision document), source code [12], and project management tools; (2) experts which represents developers with a certain level of knowledge in a specific area that could be useful to solve problems during development tasks [9, 13].

The management of the expertise in the software development process is a complicated activity, developers have different expertise needs and these needs can be found in different sources which are prone to vaporize due to the developers' unavailability or absence [44, 45]. Therefore, if an organization does not register its expertise, it is prone to repeat mistakes made in previous projects, thus the expertise needs to be managed and shared among the developers [39, 6].

1.3 Knowledge Management

Agile Software organizations generate a considerable amount of knowledge; mostly, it remains tacit since developers need to interact with other developers to get or share it—tacit knowledge represents developers' personal experience and cannot be stored or exploited. Organizations have developed diverse concepts to alleviate or guide the transformation of knowledge in software development. These concepts are supported by various methods, approaches, and tools that use symbols, graphics, and languages.

Current research in software engineering focuses on AK management to provide accurate information for better problem resolutions and better decision making [46, 43, 47]. Knowledge Management (KM) could help deliberate and coordinate people's knowledge, technology, and organizational structure, which adds value to organizations through knowledge reuse and innovation [48, 49].

Researchers in software development have been addressed the adaptation of the knowledge management practices, however, due to the characteristics of the current development models this adaptation becomes a complex process [7]. Particularly, agile software development broadly focuses in the management of tacit knowledge [15]. Agile methods are characterized by their relatively scarce documentation, and currently these methods are preferred by organizations because of the high level of communication and coordination among colleagues². The preference for the agile methods produces tacit knowledge that is not exploited.

²<https://stateofagile.com/>

1.3.1 Knowledge condensation the next step

Borrego et al. present the knowledge condensation concept, which describes the process of capturing and classifying expertise before it loses, where the aim is easing the knowledge retrieve [16].

In the externalization process, tacit knowledge becomes explicit. However, in the case of Agile development, this knowledge is not formally expressed. To access this knowledge at any time, we need to use a standardized format[53]. In their work, Nonaka and Takeuchi define two different categories of explicit knowledge: documented and formal [54]. The explicit documented refers to informal ways of articulating knowledge (e.g., text notes, drawings, audio videos). The explicit formalized refers to knowledge expressed following some standard that any person can understand, even processed by computers, if compatible with the representation format.

Compared with AK management, knowledge condensation centers on tacit and informal components that lack mechanisms to make them explicit formalized knowledge; this involves artifacts generated and shared informally among developers (e.g., unstructured text and electronic media, books, or source code). The knowledge condensation concept comes from an agile development environment where tacit knowledge is preferred [10]—agile method states that face-to-face conversations is the most effective way to share knowledge, moreover, it is only documented the information that the team consider sufficient to understand the project [50]. Little amount of knowledge becomes explicit and usually stays in the log files of *unstructured text and electronic media* (UTEM) in both agile and distributed/global development [51]; overtime this knowledge lose meaning and context, thus, it is prone to vaporize. Therefore, knowledge condensation aims to classify, retrieve, and share among stakeholders, valuable knowledge that was in an unsuitable form for its recovery. Although the intention of knowledge condensation is not to convert AK into a formal notation, it represents a step forward towards AK formalization, i.e., transforming from explicit documented knowledge into explicit formalized knowledge [52].

1.4 Knowledge condensation approaches

There are a wide variety of approaches that seek to externalize the knowledge generated in software organizations, researchers different technologies and models to support the expertise location process. Some of the main topics explored by researchers to condensate knowledge are ontologies, agents and systems (tools and other applications). Next we describe some of the proposals identified in the literature.

1.4.1 Ontologies

Researchers have begun to describe the benefits of ontologies in the software development process. Multiple proposals address activities within the requirements phase of the software

development process: project management [55–59], effort estimation [60, 61], software documentation [62–68], software measurement [69], and requirements specification [70–73]; some benefits from these ontologies are the time reduction in software documentation and the requirements specification. Other software development phases addressed are coding [74] and maintenance [75, 76], where the benefit is the time reduction solving problems and bugs by using resources generated by developers.

Although ontologies improve the software development process, how researchers developed ontologies produce some limitations. The main limitation of these proposals is the ontology development level. According to the level of development reached, ontologies can be highly informal, semi-informal, semi-formal, and rigorously-formal [77]. A high informal refers to an ontology expressed in natural language — the ontology engineers have identified the terms and concepts that describe a domain. A Semi-informal ontology uses a restricted structured form of natural language (such a knowledge representation model). Semi-formal refers to an ontology expressed in a formally defined language (e.g., Ontology Web Language (OWL)), where a knowledge representation model turns into a machine-readable model. Finally, rigorously formal refers to a formally evaluated ontology.

Another limitation of current proposals is the ontology evaluation. Gomez-Perez addresses the evaluation as two different processes: verification and validation [78]. The verification ensures that the ontology implements the correct terms and concepts to describe the domain. The validation ensures that ensure that the ontology answers questions from the domain that describes.

Finally, the lack of semantic application is another limitation of current ontologies proposed. Although some terms and concepts match among ontologies, the way proposed ontologies express their knowledge representation model hinders their primary purpose: centralize and share data from diverse sources.

Table 1.1, illustrates the limitations found in the ontologies proposed for the software engineering domain. We found that some of the limitations could arise because of the approach that researchers follow to develop an ontology. Mostly, these proposals rely on non-formal approaches. We identified two types of non-formal approaches: information system development methods and research methods. Some researchers employ information systems methods to develop an ontology; these approaches lack a description of ontology development-related activities, which results in a highly informal or semi-informal ontology. For example, Fonseca et al. [69] follow the Design Science Research Paradigm to develop a semi-informal ontology. Another type of non-formal is the research or empirical methods: empirical or research methods usually consist of tasks from the 101 ontology development guidelines [79]. In the work of Hamdan et al. [61] or the Murtazina & Avdeenko proposal [73], researchers developed the ontologies following a research method. Although researchers follow guidelines from the 101 ontology development guidelines, they do not reach a rigorously formal ontology.

Software engineers have proposed diverse empirical methods to develop ontologies; these methods emerged from the experience and insights attained from following other ontology

Table 1.1 Ontologies proposed for the software engineering domain

Authors	Formal approach	Development level	Evaluation	Evaluation type	
				Verification	Validation
Bathia et al. [70]		Semi-formal			
Martínez-García et al. [74]	✓	Rigorously-formal	✓	✓	✓
Alsanad et al. [71]	✓	Rigorously-formal	✓	✓	✓
Hovorushchenko & Pavlova [72]		Semi-formal			
Murtazina & Avdeenko [73]		Semi-formal			
Valverde et al. [57]		Semi-formal			
Rocha et al. [62]		Semi-formal	✓	✓	
Olszewska & Allison [56]	✓	Semi-formal	✓		✓
Adnan & Afzal [60]		Semi-formal			
Fonseca et al. [69]		Semi-formal			
Bathia et al. [67]		Highly-informal			
Sitthithanasakul & Choosri [68]	✓	Highly-informal			
Vizcaíno et al. [63]		Semi-informal	✓		
Bathia et al. [64]		Highly-informal			
Marques et al. [55]	✓	Semi-informal	✓		✓
Khatoon et al. [65]		Semi-formal	✓		✓
Vizcaíno et al. [66]	✓	Semi-informal			
Martinho et al. [59]		Semi-informal			
Wongthongtham et al. [58]		Semi-informal	✓		✓
Zhang et al. [75]		Semi-formal	✓		✓
Hamdan et al. [61]		Semi-informal			
Ruiz et al. [76]	✓	Semi-informal			

methods: formal methods or research methods. However, these approaches do not cover the description of formal activities to develop an ontology. For instance, Sitthithanasakul & Choosri [68] followed the Ontology for Autonomous Systems (OASys) method to develop an ontology; this resulted in a highly informal ontology.

Another formal method used in the literature is the Representation Formalism for Software Engineering (REFSENO); Vizcaíno et al. [66] and Ruiz et al. [76] used this formalism to develop their proposed ontologies. However, the REFSENO formalism only reached a semi-informal ontology in their process. Similarly, Marques et al. [55] use the Systematic Approach for Building Ontologies (SABiO). Both REFSENO and SABiO use UML to model a knowledge representation that describes a domain. UML is a widely used modeling method in the software engineering domain, but it is still not formal to model or formalize ontologies. The main difference among ontological modeling and other paradigms (UML or Entity-Relationships) is that an ontology model intends to share and have a formal, concise semantic description of a domain.

Finally, we found other works that follow a complete ontology development approach. In our previous work, Martínez-García et al. [74], we followed the Methontology framework to develop an ontology for the coding phase, resulting in a rigorously-formal ontology; however, we found some critical drawbacks during the development process. Methontology does not give a complete description of how to perform the activities from their lifecycle. Similarly, Alsanad et al. [71] followed the Enterprise Ontology combined with the guidelines from the 101 Ontology Development to develop an ontology for the requirements change management.

Other drawbacks shown in Table 1.1 are the Development level and the evaluation. As we can see, few proposals reach a rigorously-formal ontology level. We identified that the methods used by proposals usually do not include a description of how to translate a knowledge representation model into a machine-readable model (e.g., Protégé). This lack of description reduces the development level reached, resulting in a highly informal or semi-informal ontology.

The evaluation is a task often omitted by proposals, indicating a possible cause of limited ontology development level. Researchers must conduct a full ontology evaluation to reach a rigorously-formal level. The evaluation consists of two processes: validation verification. Researchers used a reasoner to check if the ontology has incongruencies or inconsistencies; for the validation task, researchers use Competency Questions (CQ) or Content Based (CB).

Methods and Methodologies to Build Ontologies

Researchers have proposed a wide range of approaches over the past years for the development of ontologies. We review some methods and methodologies using the previous section's limitations as analysis criteria: activity description, development level expected, evaluation, and naming conventions.

Activity description criteria: One of the main reasons' researchers follow non-formal approaches to build ontologies is the learning curve and complexity due to the lack of activity description. Sitthithanasakul & Choosri et al. [80] reported drawbacks when building an ontology, such as the ontology fundamentals learning curve and complexity of the current methods to develop ontologies. Moreover, Serna et al. [81] analyze ontologies to address the need for activity description on ontology development ontologies, particularly in the design of knowledge models. Therefore, we classify the approaches according to the approach detail description: insufficient details, some details, sufficient details.

Development level expected criteria: An approach reached a certain ontology level according to the tasks covered. However, one of the main limitations of the current methods or methodologies for building ontologies is the lack of details on performing tasks. We analyze the expected development level according to what an approach covers.

Evaluation criteria: evaluation is a task often omitted in the ontology development process approaches. We analyze if current approaches perform a full evaluation process: verification and validation.

Naming conventions criteria: A limitation found in the previous section is the lack of semantic application by the proposals. Schober et al. [82, 83] argue that following naming conventions is useful when researchers need to integrate an ontology to others. We analyze whether an approach suggests or mention any naming conventions guidelines.

Our purpose was to perform an analysis to look for a method or methodology that meets these criteria. Table Figure 1.2 shows some of the methods and methodologies found in the current literature for ontology development and their classification based on the criteria mentioned before.

Table 1.2 A comparison between method and methodologies for building ontologies

Method/Methodology	Steps Explained	Development Level Reached	Evaluation	Naming Conventions
Methontology	Insufficient details	Rigorously-formal	✓	
Neon	Some details	Semi-formal		
DILIGENT	Some details	Semi-informal		
AMOD	Some details	Semi-formal	✓	
SAMOD	Insufficient details	Semi-informal		
REFSENO	Some details	Semi-informal		
SCIM	Some details	Semi-formal		
C-DOM	Some details	Semi-formal		
OAsys Method	Some details	Semi-informal		✓
SABiO	Some details	Semi-informal	✓	
DOGMA	Some details	Semi-informal		
OntoSoft	Insufficient details	Semi-formal		
Enterprise Ontology	Insufficient details	Semi-formal		
UPON	Insufficient details	Semi-informal		

After analyzing the current approaches for building ontologies, we observed that none of the approaches guarantee a rigorously formal ontology level, mostly because of the omission evaluation process. Some works include an evaluation process, but they do not fully address the evaluation. For example, the Agile Methodology for Ontology Development (AMOD) includes evaluation in their development process; however, the approach only covers the verification task; authors do not explain how to perform an ontology validation. We found similar situations with SABiO and Methontology. SABiO method is a domain ontology development process [84]; it focuses on two types of domain ontologies reference and operational. The domain reference ontologies are a special kind of conceptual model. Authors give insufficient details of the evaluation process; moreover, they do not explain how to turn a knowledge model into a machine-readable by using an IDE or Ontology Editor. Similarly, it occurs when using the Methontology Framework describes an evaluation: verification and validation, but do not explain how to perform the activities [85]. In our previous work [74], we combined this framework with tutorials and other literature to reach a rigorously formal ontology; moreover, other works report similar drawbacks after using the Methontology Framework [86–88].

Other reasons for the development level limitations are complexity level, the lack of activity description, and the activities covered by an approach. The complexity of some approaches hinders the ontology development for novice ontology practitioners like when using Networked Ontologies (NeOn) Methodology and Distributed Loosely-controlled and evolInG Engineering of oNTologies (DILIGENT). NeOn is a scenario-based methodology that provides details of the ontology engineering process's essential aspects; NeOn pays special attention to the reuse of ontological and non-ontological resources [89, 90]. DILIGENT is a methodology that supports domain experts in a distributed setting [91]; DILIGENT use Rhetorical structure

theory (RST) for the terms and concepts description of a domain; however, this is an uncommon formalism for software engineer practitioners.

The lack of activity description represents a significant concern with some of the approaches reviewed. The Enterprise Ontology is a collection of terms and definitions; it was one of the first methodology proposed for ontology creation [92]. Furthermore, Software Centric Innovation Methodology (SCIM), Developing Ontology-Grounded Methods and Applications (DOGMA), and the Ontology Software (OntoSoft) Process present the same lack of activities description. SCIM is a methodology that consists of an extension of the software engineering process models [93]; SCIM defines five ontology development workflows: requirements analysis, domain analysis, conceptual design, implementation, and evaluation; although it defines an evaluation workflow, it only mentions the use of CQ to evaluate an ontology. DOGMA is an ontology engineering framework, which is based mostly on database semantics and model theory. Despite being based on formal models and theories, it does not describe an evaluation process [94]. OntoSoft is a process that explains some of the tasks needed to develop an ontology; nonetheless, authors give conceptual details on the transition from a knowledge to a machine-readable; moreover, the process does not include an evaluation [95].

The activities covered by an approach could limit their development level reached. Some approaches include some details on how to perform activities; however, mostly, the details focus on the knowledge representation model to describe a domain. The Simplified Agile Methodology for Ontology Development (SAMOD) is an agile methodology for ontology development that uses an iterative workflow focused on created documented models starting from exemplar domain descriptions [96]. Co-designing Ontologies Methodology (C-DOM) is a framework structured as a methodology that presents a strategy to draw knowledge from subject matter experts (SME) without the SME needing to know anything about creating an ontology [97]. Similar characteristics present the OASys method, which is a simplified guide of how to design an ontology.

Furthermore, some approaches like REFESENO and UPON perform knowledge modeling using UML. REFESENO is a representation formalism for the software engineering domain [98]. UPON a methodology derived from the Unified Software Development Process and adopts de Unified Modelling Paradigm (UML); UPON methodology does not provide comprehensive details of the activities for building ontologies, making it difficult to turn a UML model to a machine-readable.

A critical aspect omitted by proposals is the naming conventions to enhance ontologies' semantic application. In the OASys method, authors suggest using naming conventions; however, they do not describe or explore any.

1.4.2 Systems and other applications

There are a wide variety of approaches to externalize knowledge in software organization, the researchers purpose is to support expertise location through applications or systems focus on AK

sources. Current proposals focus only on a particular AK source; regarding artifact proposals, the main focus is source code. For example, Ponzanelli et al. [19] present SeaHawk, an eclipse plug-in that assists programmers with code snippets from Stack Overflow; also, SeaHawk uses language-independent annotations to link documents with code snippets. Similarly, Brandt et al. [21] present BluePrint, a web search interface integrated into Adobe Flex Builder development environment; BluePrint augment queries with code context through a code-centric view of search results embedded in the editor. Also, McMillan et al. [20] present Portfolio, a code search system to support developers; the system retrieves functions and describes their usage; moreover, using a combination of models, the system addresses the developers' sharing functions behaviour. Apart from that, only a few works use other types of artifacts; for instance, Borrego et al. [22] present a complementation tool to slack, which consists of a classification mechanism based on social tagging; this mechanism takes advantage of the architectural knowledge from the unstructured and textual electronic means (UTEM). In parallel, Bonilla-Morales [23] propose a tool to reuse use case diagrams; the tool allows storage information from use case diagrams.

On the other hand, some proposals use experts as an AK source, try to make aware organizations of their expertise. For example, Bhat et al. [5] propose a recommendation system that identifies developers who could be involved in the design of a software system; the approach quantifies the skills of developers to match and recommend an individual suitable with the needs of the system to design. Alternatively, Matter et al. [24] propose a vocabulary-based model, which suggests developers with the appropriate expertise to handle a bug report; the model uses developers' contributions to suggest someone. Moreover, Kagdi et al. [25] present xFinder, a tool that recommends experts based on their expertise, which is measured using the developer's commit contributions. Furthermore, Minto and Murphy et al. [26] propose Emergent Expertise Locator (EEL), an approach to ease the process of finding the right expert to ask about a problem during a software development task.

As can be seen, numerous proposals have asserted the importance of knowledge condensation in software development organizations. These proposals distinguish between artifacts and experts as the primary source of expertise for organizations. Artifacts proposals exploit the expertise generated by developers during the software development process with two characteristic limitations: a narrow focus on source code, and an informal accumulation of knowledge realized by developers but unnoticed by organizations. Expert proposals, on the other hand, employ developers as a relevant source of expertise in the software development process; to perform tasks such as handling a bug report, system design, and problem-solving. Although the empirical evidence of the described works supports the value of experts on helping, proposals give little importance to the artifacts they generate or use; furthermore, constant questions to experts could lead to an interpersonal relationship erosion.

1.5 Chapter Summary

In the current literature, we found relevant challenges to discuss. First, proposals that condense knowledge must grant access to both individuals with particular expertise level and their artifacts. This access will ensure the availability of the required resources, even if the provider is not available, and also reduce the interpersonal relationship erosion. Second, a proposal should incorporate more sources than just code, like other digital sources (e.g., bookmarks, books, manuals, and tutorials) are frequently used. One way to address these challenges could be using semantic knowledge modeling; the aim is to build a search engine for both structured and unstructured data, to search and centralize applications, databases, files, and spreadsheets. Despite the benefits identified by ontologies for the software engineering domain, the methods or methodologies for building ontologies reduce the development level of the proposals and the integration capability among ontologies.

Chapter 2

Understanding expertise location

Achieve the knowledge condensation in the software development is a challenge since it is required to identify the knowledge sources consulted by the developers and the process to capture and share these sources. To address the above mentioned challenges we perform a study to understand the expertise location process in software organizations. The study consisted of three stages: focus group study, expertise location mapping and a comparative analysis (see Figure 2.1).

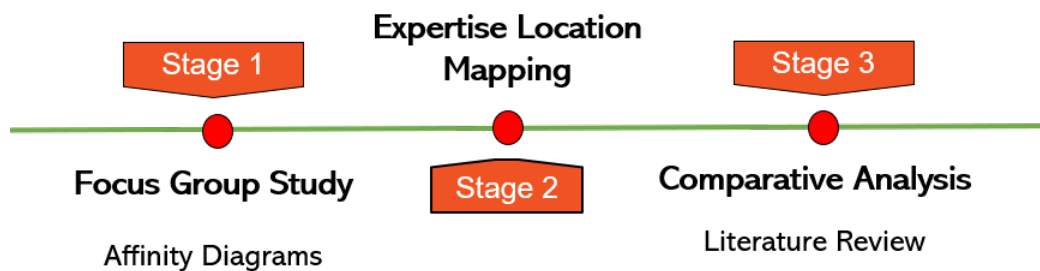


Figure 2.1 Research method for the expertise location understanding

2.1 Focus Group Study

In this stage we performed a focus group study, which had a duration of one and a half hours. The protocol focus group consisted of topics related to three categories: (1) individual search, this represents the process that developers follow to perform expertise location and once they find it who the expertise is saved (How do you seek or find knowledge?); (2) knowledge share, which was focused on the knowledge transfer process when a developer asks for help (how do you transfer knowledge?); (3) expert search, this category was focused on the process of selecting an expert to ask for help (how do you transfer knowledge?). The focus group was conducted in Spanish and then translated to English for the purposes of this PhD Thesis.

2.1.1 Participants

The focus group had 5 participants, which are developers from a software organization in México. The age of the participants ranged from 26 to 36 years old (mean age 29.45 years old), with an average work experience of 4.72 years (min= 2; max=9).

2.1.2 Data Analysis

During the data analysis, we used the Grounded Theory approach [99]. Grounded theory systematically uncovers patterns in several individuals through open, axial, and selective coding. In the open coding, the focus group was analyzed and coded sentence by sentence. Next, in the axial coding, we devised relationships between the codes and the established categories. Finally, through selective coding, we classify the relationships identified in the categories mentioned before. For the selective coding, we use affinity diagrams (see Figure 2.2), which is a tool that synthesizes a set of verbal data (e.g., ideas, opinions, expressions), grouping them according to the relationship they have with each other.

In Figure 2.2, we present the activity diagram that classifies the data of the answers that appeared most recurrently. We also selected relevant quotes that create a stronger relationship between the concepts to enrich the results.

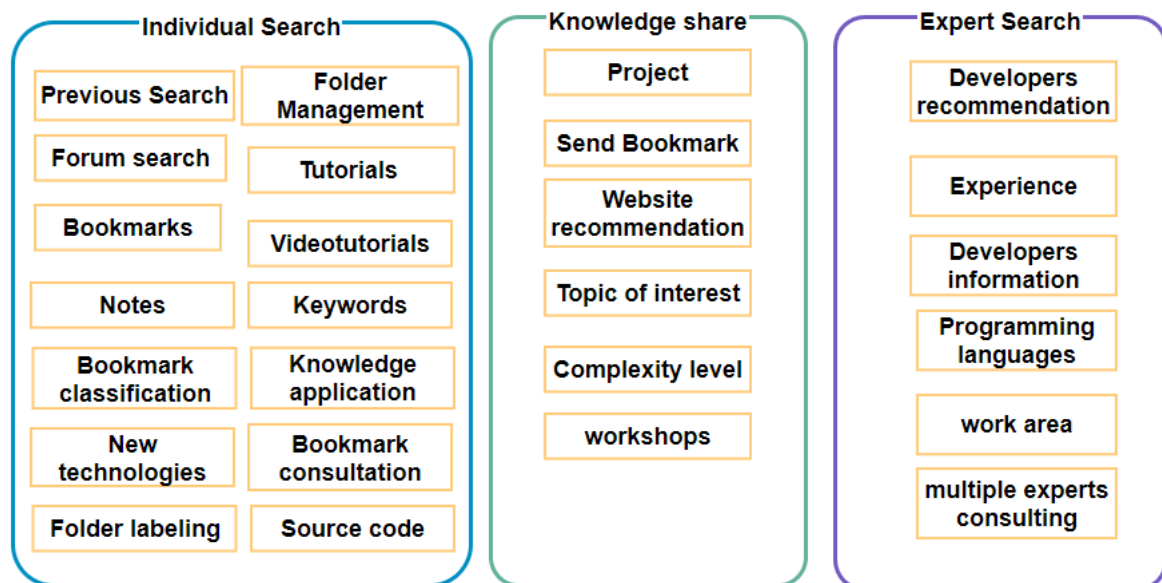


Figure 2.2 Focus group affinity diagram

2.2 Expertise Location Mapping

From the insights obtained on Stage 1, we map the expertise location process, Figure 2.3 presents the expertise location knowledge flow, we use an scenario to illustrate the expertise location in software development. A scenario starts with a developer having the need of

expertise to solve a problem or doubt (A). Then, the developer can choose between an expert search (C) or artifact search (B). The developer does a search among all available artifacts, in the case of choosing an artifact search (B). An artifact verification is done to see if it is fulfilled the objective (E). If the need of expertise is satisfied, the problem is solved (F). On the contrary, if the need of expertise is not satisfied the developer must check more artifacts to solve the problem (A). On the other hand, in the case of choosing an expert search (C), the developer does a search looking for someone with the knowledge to solve the problem or doubt, once the right person is found, their availability must be checked (D). Finally, if the expert fulfilled the objective, the problem is solved (E). Otherwise, another expert search is needed (C). Sometimes an expert can lead the requester to an artifact (B) and vice-versa (C). Sometimes the developers cannot solve the problem, so they continue the next day trying to solve it (G).

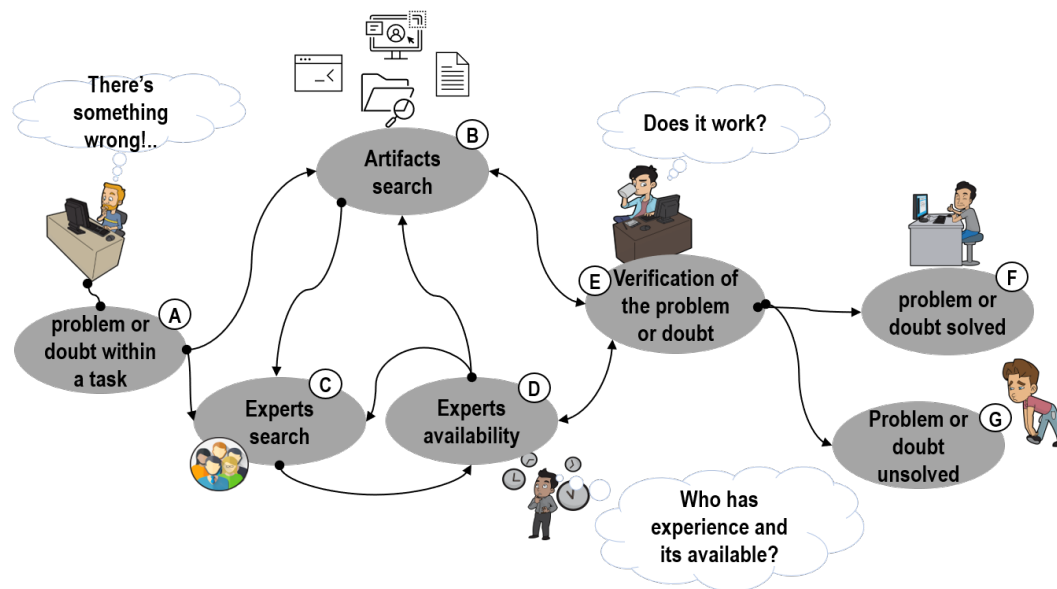


Figure 2.3 Expertise location process in software development.

2.3 Comparative Analysis

Our focus group is not the first attempt to understand the expertise location: sources, needs, problems and flow. Here we analyzed the results obtained from different studies found in the literature.

Researchers have employed different methods to obtain data about the expertise location: qualitative studies and literature review (surveys). Viana et al. performed a qualitative study based on interviews and ethnography; their purpose was to understand how lessons learned are created and maintained in software projects—lessons learned are used in agile environments to describe discussions and situations to resolve problems [6]. Furthermore, Viana et al. conducted a qualitative study of knowledge transfer in small organizations from a software developers' practitioner's view [100].

Ali performed a literature review of software reuse evidence in software engineering; the aim was to identify how developers find reliable information to make the right and informed decisions [42].

Josyula et al. aim to identify information needs and the resources used by developers in software development. In their work, they conducted 17 semi-structured interviews (6 of them were face-to-face interviews and 11 via Skype) [43].

Borrego et al. present an empirical study to understand AK articulation in UTEM in AGSD teams. The study consisted of an on-site observation of one of the four companies and an exploration of all four companies' observations [51].

Li et al. report an empirical that aimed to understand how developers seek information and which they prefer sources. The study includes two human experiments: 24 developers performing two typical software development tasks [101].

LaToza et al. conducted two surveys and eleven interviews to understand developers' tools, activities, and practices to manage knowledge [102].

Next, we describe our comparative analysis of our focus study results with the related work in the literature about knowledge: sources, needs, and problems.

2.3.1 Knowledge Sources

We identified from our focus group that developers use a knowledge source according to the situation. In the situation of understand how to solve an error/bug the forums, online tutorials and bookmarks are the most common sources of knowledge, “[Participant 4]: *well, my way of researching ... well, it's similar ... well, I look in forums, I review projects and those who could save me time i register them, and addition i store bookmarks*”, another option is the discussion with co-workers, “[Participant 5]: *usually i ask to a person (developer), because i know he has some knowledge according to what i need*”. Furthermore, when the need is to improve programming skills on a language, the most common source is the official documentation “[Participant 1]: *when start working in a project with new technology, i first search all in the official documentation, for example if it is a technology that was created by a company like google*”.

Table 2.1 shows the different sources identified by researchers during their studies. The most common sources identified are co-workers, code sites and previous projects.

2.3.2 Knowledge Needs

When we talk about knowledge needs, these include a wide range. In our results, we identified three primary needs. Developers sometimes need to understand and solve errors/bugs “[Participant 1]: *when I have an error in my code I look for examples (e.g., forums)*”. Another need occurs when a developer works on a project with new technologies. Finally, developers need the knowledge to improve their experience “[Participant 4]: *recently, we occupied learning new programming languages that we had never seen, and we didn't even know they existed*”

Table 2.1 Knowledge sources used by developers during the expertise location

Knowledge Source	Viana et al. (13)	Viana et al.(14)	Josyula et al.	Focus Group	Li et al.	Borrego et al.	Ali	LaToza et al.
Co-workers			✓	✓				✓
Management tools	✓							
Blogs			✓		✓			
Forums			✓	✓	✓			
Online tutorials				✓	✓			
Courses			✓					
Books		✓	✓					
Previous projects		✓	✓	✓			✓	
Social networks			✓					
Ofical Documentation				✓	✓			
Code sites		✓	✓		✓			
Bookmarks				✓				
UTEMS						✓		

Table 2.2 Knowledge needs that lead developers to perform expertise location

Knowledge needs	Josyula et al.	Focus Group	Borrego et al.	LaToza et al.	Viana et al. (2014)
Learn specific software tools	✓				
Improve programming skills	✓	✓			
Understand and solve errors/bugs	✓	✓	✓	✓	✓
How to code a specific module	✓				
How to test a specific task	✓				
Product design and architecture	✓	✓			
Technical information for a problem during a project			✓		

We compare the results from the focus group with the works reviewed. In Table 2.2, we present the results of a comparison of the knowledge needs identified by researchers. The most common knowledge need that developers present is the understand and solve errors/bugs. Developers spend a considerable amount of time on these types of needs; this means that a large amount of knowledge is generated with solutions that could avoid repeating errors during a software development project.

2.3.3 Knowledge Problems

During our study, we identified different problems associated with the expertise location process. Table 2.3 presents the problems that arise due to agile software development characteristics. For example, a crucial characteristic is a strong communication over extensive documentation. During the expertise location process, developers suffer the lack of documentation of the agile development environments.

Some of the problems caused by this lack of documentation are knowledge capture and retrieval, co-worker interruptions, and knowledge transfer. Knowledge capture in software is a

Table 2.3 Knowledge problems identified during the expertise location

Problems	Viana et al. (2013)	Focus Group	Ali	Borrego et al.	LaToza et al.	Viana et al. (2014)
Knowledge transfer		✓				✓
Co-workers interruptions				✓	✓	
Knowledge capture	✓		✓	✓	✓	
Knowledge retrieval		✓		✓		

challenging task to do since knowledge mostly remains tacit in agile environments. Borrego et al. state that in communication tools (e.g., skype, Trello), we can find important information to solve problems; however, it tends to get lost in conversations over time since developers do not capture it [51]. Furthermore, Viana et al. report that some lessons are not learned during the lessons learned life cycle. Sometimes the lessons are not learned during the meeting because developers only summarize the problem and the solution—a considerable amount of knowledge remains tacit since developers only discussed informally during the scrum meetings [6].

Knowledge retrieval involves different problems during the expertise location process. We identified that developers sometimes do not find knowledge previously saved “[Participant 1]: *in the case of bookmarks, sometimes you may have troubles trying to remember the keywords or the resource subject*” Knowledge from developers cannot be retrieved unless someone knows about the developers’ skills.

LaToza et al. found that understanding the rationale behind code is a challenge for developers due to the lack of documentation; this leads them to cause co-worker interruptions to ask for advice—over time, constant interruptions can produce interpersonal relationship erosion [102].

Finally, knowledge transfer among developers involves some problems. Our study identified that developers have trouble transferring knowledge; knowledge cannot be transferred unless a developer identifies the expert’s existence: a person who knows a particular topic or situation.

2.4 Chapter Summary

In this chapter, we analyze the expertise location process within software organizations. We studied the sources that developers use and their challenges to obtain, capture, and retrieve expertise. We use our findings to design an expertise location process flow. Furthermore, we compare our findings with related work in the literature. As a result, we found that bookmarks its a popular source to find knowledge to solve problems or doubts. Regarding the problems, developers argue that retrieving and sharing are critical within the expertise location process. Finally, the most common knowledge needs that developers face are improving coding skills and solving bugs.

Chapter 3

Knowledge Condensation Model

From the previous Chapter (see Chapter 2), we obtain a set of requirements needed to support the expertise location process in software development organizations. We use these requirements for a model to condensate knowledge (see Table 3.1). In this chapter, we present our knowledge condensation model. The proposed model seeks to support expertise location by managing the experts and resources developers use within a software development project.

The model comprises three modules that help to condensate knowledge within software organizations: formal grammar, semantic knowledge, and expertise tools. Figure. 3.1 illustrates the modules proposed for the knowledge condensation model. To help with the knowledge representation of the software engineering domain, we employ a Formal Grammar (1). A grammar describes the structure of a language's phrases and words and applies equally to programming languages or human languages. In software development environments is hard to capture knowledge from a person; moreover, it is complicated to present information and knowledge to end-users. Therefore, we propose a semantic knowledge module which employs ontologies to centralize data among different sources (2). Ontologies are an excellent alternative to centralize and share from a domain. According to the Semantic Web Ontology established as understood in the Semantic Web, ontologies are closer to current information systems. They are based on XML/RDF, which can be more easily captured or returned from a person to an existing information system. Current information systems usually are built with XML interfaces; therefore, it is easier to present information to end-users. Finally, in the previous section, we identified that developers use various sources to capture their knowledge;

Table 3.1 Requirements to support expertise location in software development

Requirements
R1. The model should have mechanisms where users can search for expertise and update their knowledge
R2. The model should handle the knowledge capture and representation
R3. The model will perform decision making with the expertise needs of the developers from the organization
R4. The model should condensate the knowledge from the experts in an organization and the resources used by them (artifacts)

therefore, the expertise tools module describes approaches to condensate knowledge based on the Semantic Knowledge Module (3). Next, we describe each of the modules and the implementations developed for this PhD thesis.

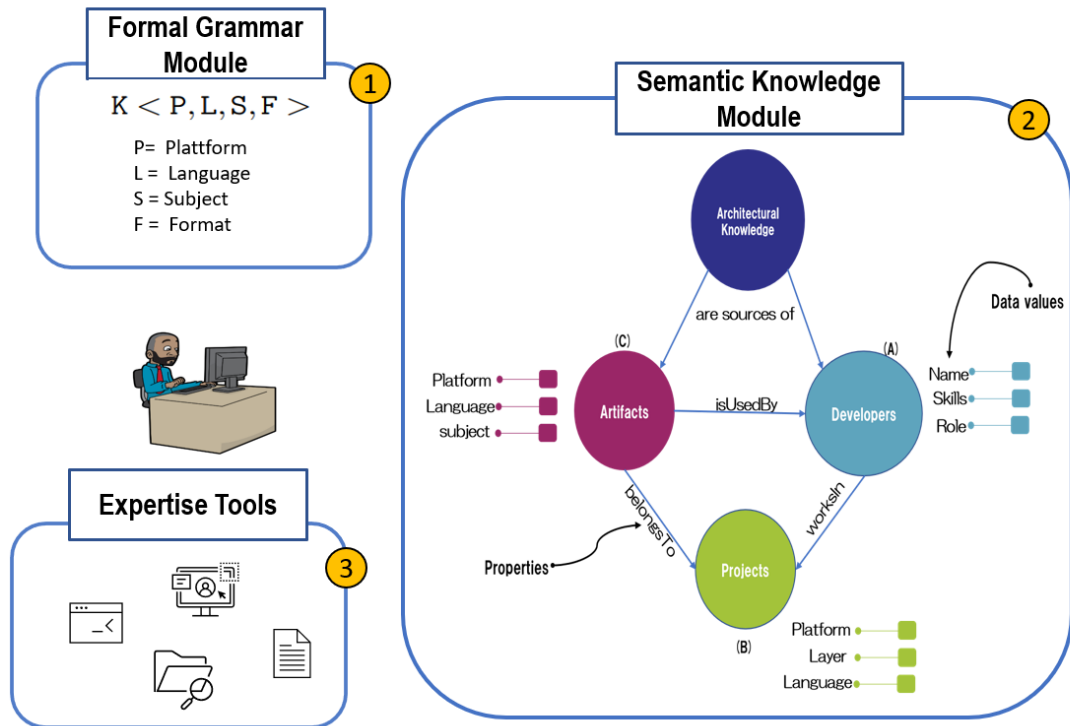


Figure 3.1 Knowledge Condensation Model

3.1 Formal Grammar Module

The purpose of this thesis is to support expertise location through the condensation of the knowledge generated by software organizations. Knowledge in software development environments is continuously changing; the formal grammar module describes knowledge flow in software organizations. We employ a formalism to describe how developers create, share, and store knowledge (**R2**). Moreover, how knowledge is in constant update through the projects and experience of developers. The formalism consists of a set of ordered data sequences (tuples) of the form:

$$K < P, L, S, F >$$

Every element of the tuples helps to describe the knowledge during the software development process. P represents the development platform associated with a software instance. Software projects are usually developed for a particular platform (e.g., Web, Mobile, Desktop, Hybrid).

Table 3.2 Elements for the user profile building

Element	Properties
Personal data	Name
	Role
	Phone
	Email
Project history	Project Name
	Platform
	Language
	Role
Contributions (tuples)	Manuals
	Tutorials
	Source code
	Documentation

L represent a programming language related to the platform used in a project. Given a project where the platform is $P = \text{Web}$, L can be any language for web development (e.g., python, PHP, HTML, CSS).

S represents the subject, the aim is to give a detailed description of the knowledge using keywords (e.g., GUI, Scripts, Events). Finally, F represents the format in which knowledge (e.g., manuals, source code, bookmarks).

3.1.1 Knowledge Profile Creation

To support the expertise location through knowledge condensation, first we build a user profile to keep a history of registered users and generated by them. Moreover, it is necessary to update the users' information and build new searches based on the system's profile.

To build a user's profile, we need to obtain their personal information and the resources they use to solve problems or doubts during a software development project (contributions). Furthermore, the user profile building involves a registered user's ability to help a colleague and the value that shared resources could have within the system to solve the problem or situation which makes a request. Therefore, build a user profile will enable access to potential experts and their resources to solve particular problems or doubts.

Table 5.3 shows the main elements needed to build a user profile: *personal data*, *project history*, *contributions* (tuples). The *contributions* (tuples), also known as artifacts, are resources that developers generated or find to solve a particular problem or doubt during a software project. *Personal data* represents basic information to allow users to contact experts and associate them with their contributions. Such contributions rely on the *project history* of the developer. Developers obtain experience and resources to solve problems from every project they had work.

In this thesis, we focused on modeling the knowledge of software developers during a software project. Each registered user stores knowledge contributions as tuples, as shown next:

$$K_i \begin{bmatrix} k_1 \langle P, L, S, F \rangle \\ k_2 \langle P, L, S, F \rangle \\ k_3 \langle P, L, S, F \rangle \\ \vdots \\ k_n \langle P, L, S, F \rangle \end{bmatrix}$$

K_i represent the knowledge set from an organization, where i represent a particular developer within the organization, and n their tuples. A developer can store n tuples, which represents the set of contributions.

3.1.2 Knowledge Search

Developers perform an expertise search to solve a doubt or problem during a project. This search can be modeled as a tuple, as shown next:

$$Sr \langle P, L, S \rangle$$

All knowledge is divided into substrings (tokens) of information that form the knowledge of the search requested by a developer. In Sr (search request), the F element is omitted since the knowledge format is irrelevant.

We describe the search for knowledge as follows. Given $y = f(Sr, K_{in})$, where Sr represent the knowledge needed by a developer to solve a problem or doubt, and K_{in} an artifact from a particular developer from the organization. $f(Sr, K_{in})$ performs a comparison between the request and the current knowledge within the organization. Therefore, y is the intersection of the tuples that fulfill the developer's search request, which is denoted as $Sr \cap K_{in}$ when $K_{in} \in S$. In this sense, $S = \{E1, E2\}$ represent the set of scenarios when a resource could be useful given a particular $Sr \langle P, L, S \rangle$.

$E1$ is the scenario that occurs when the request Sr exactly matches K_{in} . For example, $Sr \langle \text{Web, Python, GUI} \rangle$ and $K_{in} \langle \text{Web, Python, GUI} \rangle$.

$E2$ is the scenario that occurs when the request Sr matches only with the platform and language of K_{in} . Developers may misunderstand the root cause of their problems, using the wrong keywords when searching for useful resources. Therefore, it will be useful to suggest recourses related to the same language. For example, $Sr \langle \text{Web, Python, GUI} \rangle$ and $K_{in} \langle \text{Web, Python, Events} \rangle$.

3.1.3 Knowledge Update

The developer profile and their contributions need to be updated continuously. The knowledge update can be through the project history; developers gain experience and generate different artifacts from every project. These artifacts (e.g., manuals, bookmarks, source code) generated during a software can be viewed as contributions since they could be useful in future projects. Every time a developer performs expertise, they find or create a resource. Thus, their contributions increased during a software development project. The above is denoted as follows:

$$K_i \begin{bmatrix} k_1 \langle P, L, S, F \rangle \\ k_2 \langle P, L, S, F \rangle \\ k_3 \langle P, L, S, F \rangle \\ \vdots \\ k_n \langle P, L, S, F \rangle \end{bmatrix} \therefore K'_i \begin{bmatrix} k_1 \langle P, L, S, F \rangle \\ k_2 \langle P, L, S, F \rangle \\ k_3 \langle P, L, S, F \rangle \\ \vdots \\ k_{n+1} \langle P, L, S, F \rangle \end{bmatrix}$$

3.1.4 Knowledge Construction

Using our proposed grammar, we can built new knowledge by analyzing the developers' behavior to solve doubts or problems. For example, developers sometimes use resources from a language different from the one with the problem. In this case, developers analyze other language resources to implement the solution for the language.

3.2 Semantic Knowledge Module

From the Chapter 1.4, we found that proposals accumulate expertise without the organization being aware of their existence. Each proposal uses its inputs, and outputs cannot be centralized. In this sense, semantic knowledge modeling could be a way to link experts and the artifacts developers produce and consume during software development. Here we describe a semantic knowledge model of the software development domain and the ontology developed based on the semantic model. We aim to highlight how semantic modeling could improve knowledge condensation. The semantic model was developed using knowledge modeling techniques; these techniques consist of performing tasks such as knowledge acquisition, requirements specification, and knowledge conceptualization. The knowledge acquisition task consists of acquiring the knowledge from the domain experts (e.g., interviews, focus groups, or surveys) or from the literature. The requirements specification task consists of following guidelines to capture users' knowledge and produce a main concept vocabulary. Finally, the conceptualization task consists of organizing and model the acquired knowledge using external representations (e.g., UML, Integrated Definition for Ontology Description Capture (IDEF5)).

Software organizations produce and consume knowledge, which is formally known as AK. Figure. 3.2 show the main elements of the semantic model: projects, artifacts, and developers.

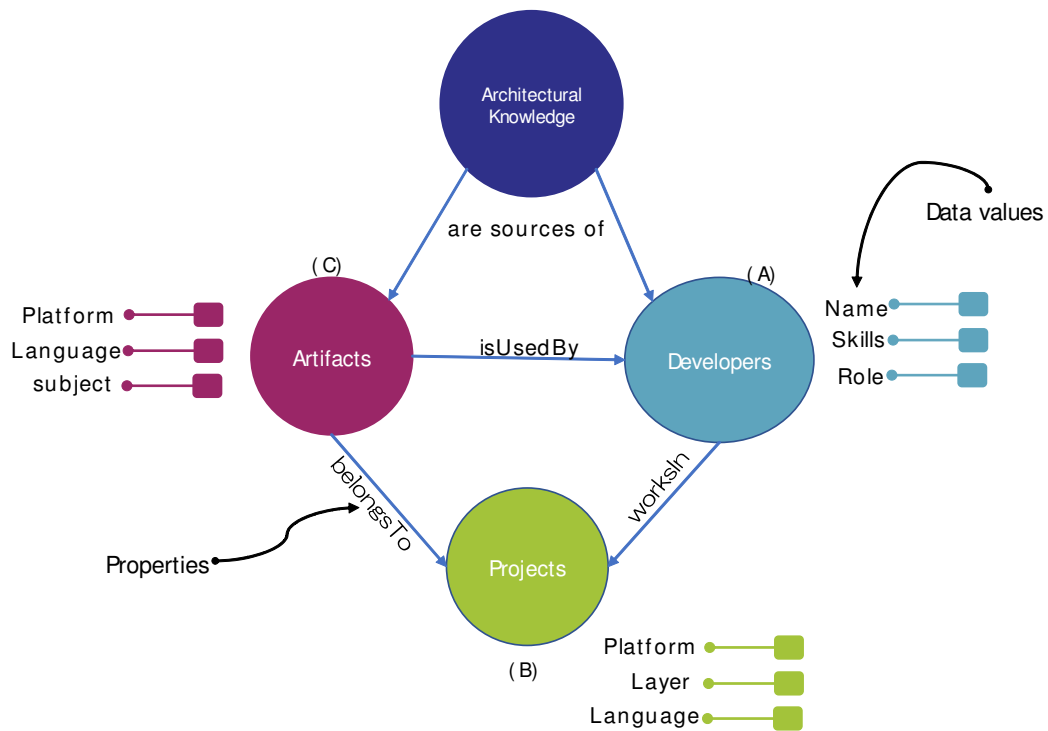


Figure 3.2 Semantic knowledge model of architectural knowledge (AK).

Developer represents a team member's description in a software development project; this description could include properties such as a name, email, or cellphone. Projects represent a description of a developer's current or past works; the aim is to create a developer's skills profile based on the project record; the properties used to describe a project are a language, platform, role, and project name. Artifacts represent a description of developers' different resources to solve problems or doubts while developing software (e.g., bookmarks, code snippets, documentation, and tutorials); artifact descriptions include properties such as a platform and a subject.

The presented model allows a link among their elements through the following characteristics. The first characteristic of the model is data values. The data values characteristic help describe an element; for example, it describes personal info in the developer element.

The second characteristic is the properties. Properties are binary relations on instances of elements from the model; their purpose is to link two instances. For example, the property "isUsedBy" links an instance of Artifacts with Developers; thus, knowing the provider of an artifact grants access to the expert and their experience. The property "belongsTo" link an instance of Artifacts with a Project; the aim here is to locate in which projects developers create or generate an artifact. Finally, the last property is "worksIn"; the property link developers and projects to build a background of developers' skills based on the project history.

This module's primary goal is to generate a knowledge representation of the software development process. The objective is to identify, represent, and share this knowledge (expert or experts' artifacts) (**R4**). We aim to avoid wasting time-solving the already solved problems.

3.2.1 An ontology for the coding phase during the Software Development Process

In this thesis, we develop an ontology to perform an implementation of the semantic knowledge module. An ontology brings several benefits, such as a shared concept of the knowledge in the code phase and a way to link the artifacts (resources used by developers in the project) and the experts (artifacts provider). Furthermore, ontologies present a visual way to share a common understanding of an information structure between several people or computer systems (**R2**), make reasoning about data, and also they allow to reuse of knowledge through ambiguities clarification (**R3**).

This section presents an implementation of the semantic Knowledge Module. Here we present how a semantic model (see Figure. 3.2) integrates into an ontology to support the knowledge condensation process. Moreover, we describe how the elements of the ontology help to link artifacts and developers. Results show that the proposed ontology does not present incongruence or inconsistency and answers the competency questions correctly. The ontology brings several benefits, such as a shared concept of the knowledge in the code phase and a way to link the artifacts (resources used by developers in the project) and the experts (artifacts provider). The ontology was developed by following the Methontology Framework [85], an accepted methodology to define the development life cycle in Ontological Engineering (from requirements specification to maintenance). The Methontology Framework life cycle (see Figure. 3.3) includes five phases: (1) Specification, (2) Conceptualization, (3) Formalization, and (4) Evaluation, (5) Maintenance. The next subsections describe the phases and activities needed for the development of our ontology.

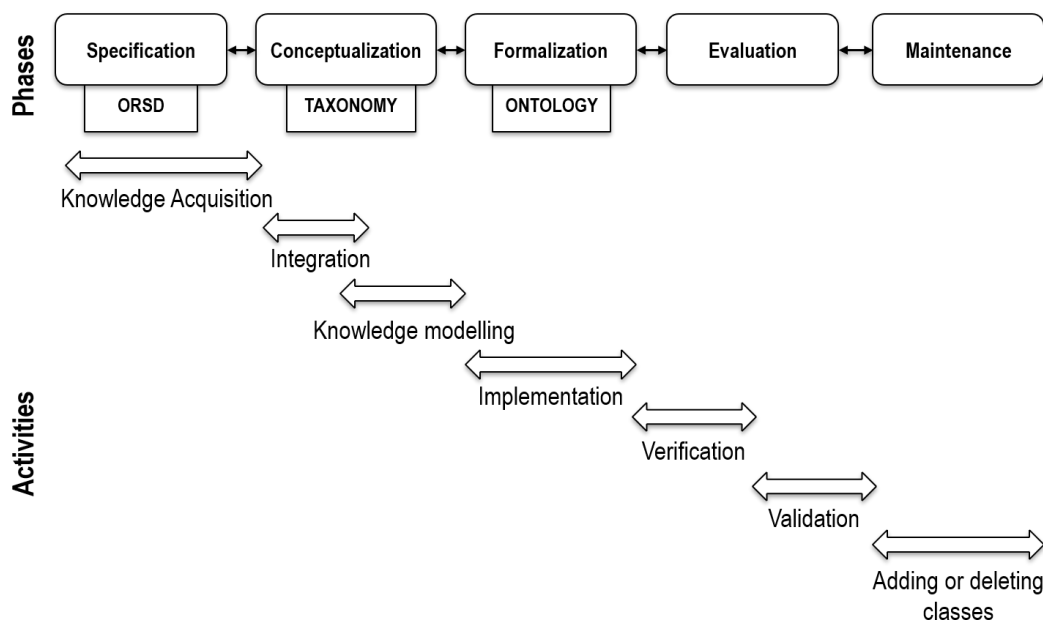


Figure 3.3 Methontology Framework life cycle[85].

Specification Phase

Specification phase establishes a document covering the ontology's purpose (1), scope (2), implementation language (3), intended End-Users (4) and Intended Uses (5). This document is done by doing the following task:

Knowledge Acquisition activity: In software development, the key of project success lies in the software specification [60]. Suarez-Figueroa et al. [61] present guidelines based on the use of the Competency Questions (CQ) and the existing methodologies to build ontologies. These guidelines help to capture knowledge from users and to produce the Ontology Requirement Specification Document (ORSD). The ORSD document helps to identify the knowledge that the ontology contains, and it is useful to define the requirements the ontology must cover.

For the knowledge acquisition activity, we come up with a focus group approach as mentioned in Section 2.1. The objective of the focus group was to know the process of searching for expertise within the software development teams either to store it or to share it, as well as the process of finding an expert for consulting. The participants were asked about the process of individual search, knowledge sharing and expert search.

The data were extracted from focus group and was modeled then using affinity diagrams (see Figure. 2.2), which is a tool that synthesizes a set of verbal data (e.g., ideas, opinions, expressions) grouping them according to the relationship they have with each other. This process begins with the transcription of the interviews to find the key data of the participants' responses. From that, the data of the answers that appeared most recurrently were classified.

Later we continue with the analysis of the data to identify the relationships between the processes of search for expertise.

Finally, from the affinity diagram and the defined categories, conclusions were obtained. With the collected information as part of the specification phase we create a document following the ORSD guidelines. In Table 3.3 shows a fragment of the developed ORSD with the requirements defined for the ontology for the coding phase.

Conceptualization Phase

Once all the needed knowledge has been acquired, it must be organized. Conceptualization phase is focused on organizing and structuring the acquired knowledge using external representations (e.g., UML, IDEF5) which are independent of the ontology implementation languages. The organizing and structuring tasks are:

Integration activity: To avoid redundant information, it must be considered the reuse of ontologies (definitions already built). The Ontologies were consulted in the following databases:

- Swoogle
- DAML Ontology Library
- ONKI Ontology Library Service

Table 3.3 Ontology Requirements Document Template Fragment

Otology Requirements Specification Document Template	
1 Purpose	The integration of the artifacts, projects, and experts in the code phase of the software development process
2 Scope	The ontology has a focus just on the code phase of the software development process domain. The level of granularity is directly related to the competency questions and terms defined.
3 Implementation Language	The ontology must be implemented in OWL language using Protégé ontology tool.
4 Intented End-Users	User 1. Programmer search for resources to solve a problem (e.g., requirements, bugs or doubts with a process). User 2. Programmer searching for an expert to ask for help, User 3. Programmer searching for information about a project and his participants User 4. Programmer updating or registering his expertise.

- Linked Open Vocabularies (LOV)

These databases were searched on internet and some of them are the most common cited in research articles available. In addition, we searched for ontologies of the same domain in academic databases (e.g. IEEE Xplore, ScienceDirect), like the one trying to build in this work in academic databases.

As a result of the integration task, it was not found any ontology with same domain as the one trying to build in this work, neither in ontology databases nor in the literature review.

Knowledge modelling activity: This task consists of storing statements about facts by building meaningful information structures through multiple representations (e.g., mind maps).

Based on the terms and concepts identified from the focus group, we created a taxonomy. In Figure. 3.4 we present a taxonomy of the knowledge produced in the code phase of the software development (programming knowledge). The main elements of the taxonomy are i) profile, ii) projects and iii) artifacts.

The Profile entity represents a description of a programmer in an organization with information such as name, role, skills, projects has worked or is working currently.

The Project entity represent information about developer's current project. In this way, you can know the developers' skills based on the project history and the artifacts used in those projects. So, developers create artifacts by working on projects, and those are used by others to solve problems. In summary, programmers have a profile and work assigned in a project,

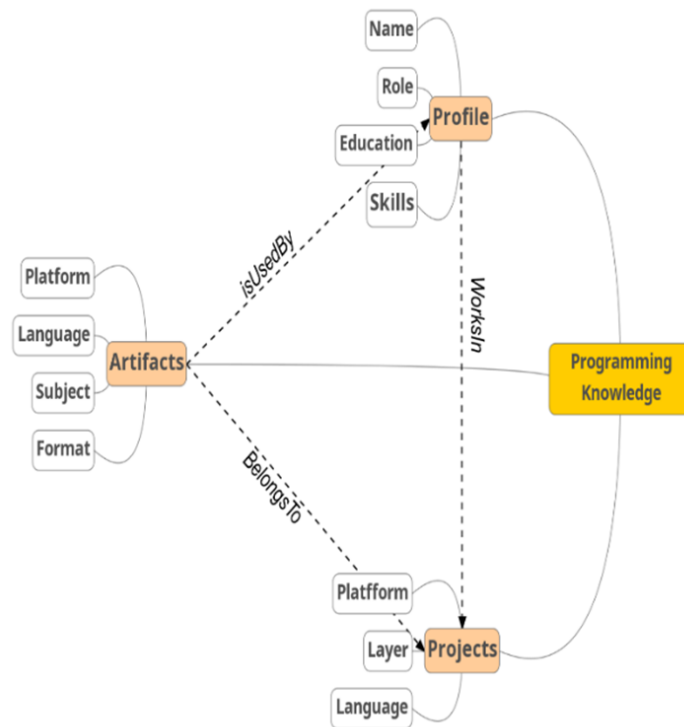


Figure 3.4 Taxonomy of Knowledge Expertise in code phase.

which is developed in a certain platform (e.g. web, database, desktop and mobile) which has layers (e.g. backend and frontend) and a programming language (e.g. JavaScript).

Formalization Phase

Formalization phase converts a conceptual model (taxonomy) to a formal model which is known as Implementation activity. For the Implementation activity we use Protégé. This tool uses Ontology Web Language (OWL) to define an Ontology.

Using the taxonomy (see Figure. 3.4), we defined the classes' names (in OWL, classes are interpreted as a set of individuals or objects), properties, and instances. In Figure. 3.5 we present a screenshot of the Protégé tool. The principal class that represents a set of all individuals is "Thing", thus all classes are subclasses of that one. The main classes of our ontology: *Team*, *Artifacts*, *Project*, *Layers*. *Team* class represents a developer team in an organization. *Artifact* class represents the resources used by developers to solve a doubt or a problem. *Project* class represents a description of the work and activities done by developers.

In conclusion, developers (members of a *Team* class) work in a *Project* in an organization, and when a developer has a doubt or problem uses *Artifacts*.

Properties in OWL represent a relationship between two individuals. There are two types of properties: object and data type properties. The object properties link an individual to another individual. The datatype properties link an individual to a data value expressed in Extensible Mark Language (XML) or Resource Description File (RDF).

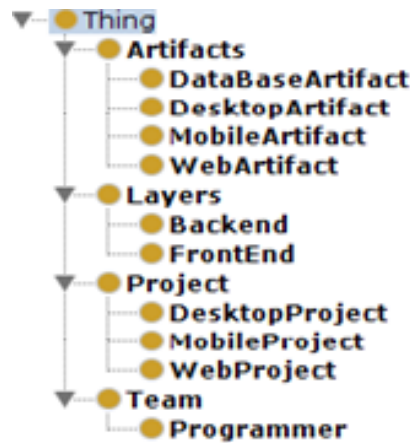


Figure 3.5 Coding phase expertise ontology: screenshot of main classes developed in Protégé.

Table 3.4 Ontology properties, ranges and domains

Property	Inverse	Type	Domain	Range
isUsedBy	hasUsed	Functional	Artifacts	Programmer
isMemberOf	N/A	Functional	Programmer	Projects
isMadeBy	hasMadeBy	Functional	Artifacts	Programmer
hasWorked	N/A	Functional	Programmer	Projects
hasUsedIn	N/A	Fucntional	Artifacts	Projects
isBasedOn	N/A	Functional	Projects	Layers

The properties defined to our ontology are shown in Table 3.4. The property “isUsedBy” help to link a programmer (a subclass of a team class) to the artifacts that have been used to solve problems or doubts in a project. The “isMemberOf” and “hasWorked” property helps to identify in which Programmer has worked or which project is currently working on. The properties “IsMadeBy” and “hasUsedIn” help to identify who creates an artifact and in which project was created or used.

Evaluation Phase

In the traditional Methontology Framework the evaluation is considered as an activity which is carried out during all the phases. For our ontology development, this activity is considered as another phase in the proposed methodology, which consist of carrying out a technical judgment of the ontology, according to the ORSD, by doing the following tasks:

Verification activity: This activity is a technical process which is done to guarantee the correctness of the ontology, according to the specification requirements. The verification activity was be done using the Pellet plugin reasoner on Protégé (see Figure. 3.6). An ontology reasoner is a software program able to infer logical consequences from a set of asserted facts or axioms. During the verification using Pellet no incongruence or inconsistency were found in the ontology, when the ontology was analyzed

```

INFO 14:02:43 ----- Running Reasoner -----
INFO 14:02:43 Pre-computing inferences:
INFO 14:02:43   - class hierarchy
INFO 14:02:43   - object property hierarchy
INFO 14:02:43   - data property hierarchy
INFO 14:02:43   - class assertions
INFO 14:02:43   - object property assertions
INFO 14:02:43   - same individuals
INFO 14:02:43 Ontologies processed in 236 ms by Pellet
INFO 14:02:43

```

Figure 3.6 Output in Protégé using Pellet OWL-DL Reasoner

Table 3.5 Competency Questions in Natural Language

Competency Questions
CQG1(Expert seeking)
CQ1. In which project "developer name" has been worked?
CQ2. In which language "developer name" programs?
CQ3. Developer with skills on "language"?
CQ4. Which resources has been used by "developer name"
CQG2(Artifact seeking)
CQ1. Resources for web developing?
CQ2. Resources used in "name" project?
CQ3. Resources used by "developer name" in "name" project?

Validation activity: It is the process done to ensure that the ontology fulfills the purpose for which it was built. The validation was be done by using Competency Questions (CQ), which consist in a set of questions defined in the ORSD in a natural language, the ontology must answer these questions correctly. The CQs were based on examples of doubts of problems that developers try to solve. Furthermore, questions were transformed into a computer language, we used the Manchester OWL syntax to translate the questions in natural language into a computer language applied in Protégé.

Table 3.5 shows the questions designed to query in the validation activity. The questions are divided in two groups: a) Expert seeking b) Artifact seeking. Due to the two types or searches done to solve a doubt or problem (see Figure. 2.3). You can either look for a resources (artifacts) or look for an expert the recommends you an artifact.

In the aim to perform the evaluation, the ontology must be populated by creating instances. This process usually involves linking data to the elements of the ontology. We describe an scenario to populate the ontology with the elements for the evaluation. Figure. 3.7 present a description of a scenario application used in the evaluation phase.

“Ana is a developer within an organization, she is a new member of the Project_one, one day she had a problem with one of the modules she was programming, so she had to spend time searching the web to solve the problem. Sometime later he met Omar another member of the Project_one, he told her that he had had the same problem, which would have been useful he had the sources of how he had solved the problem.”

Figure 3.7 Scenario description

Figure. 3.8 shows an example of instances created during the ontology population. These instances represent a scenario of programmer working in an organization. Omar represents an instance from the Programmer subclass, Project_One an instance from the Project class, and all the resources are instances from the Artifacts class. Omar is currently working on Project_One and has used many resources (artifacts) to solve doubts or problems in the project.

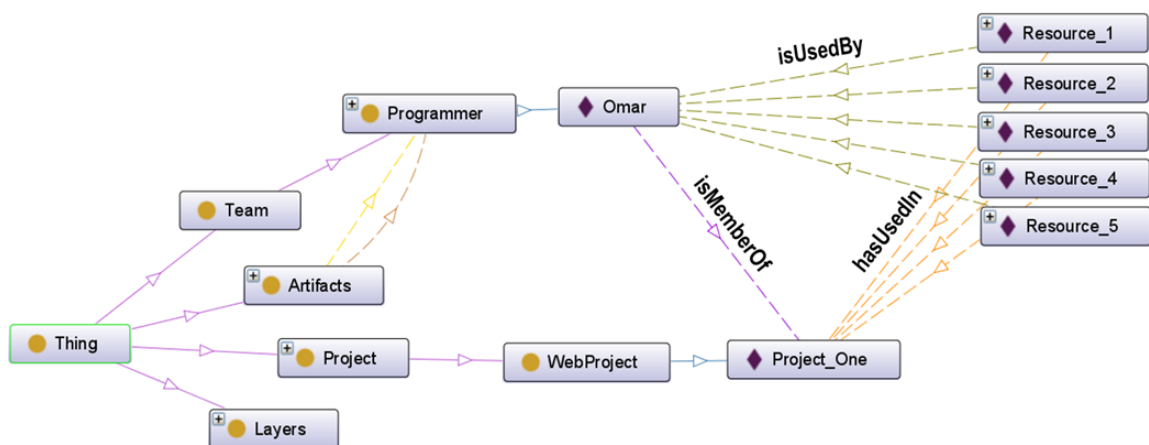


Figure 3.8 Ontology instances example in Protégé.

Figure. 3.9 presents an example of a question done in Protégé during the validation activity. In this case, the object properties link the resources that Omar used in the Project_One. In this way, Ana could reuse the resources used by Omar, since Omar's resources will be associated with him and the project in which he used them.

Maintenance phase

Finally, at the end of all the phases, there is the maintenance phase, which consists of tasks covering from erasing obsolete instances or adding new ones over the time. This phase was not considered because the scope of we were focused only on the ontology development.

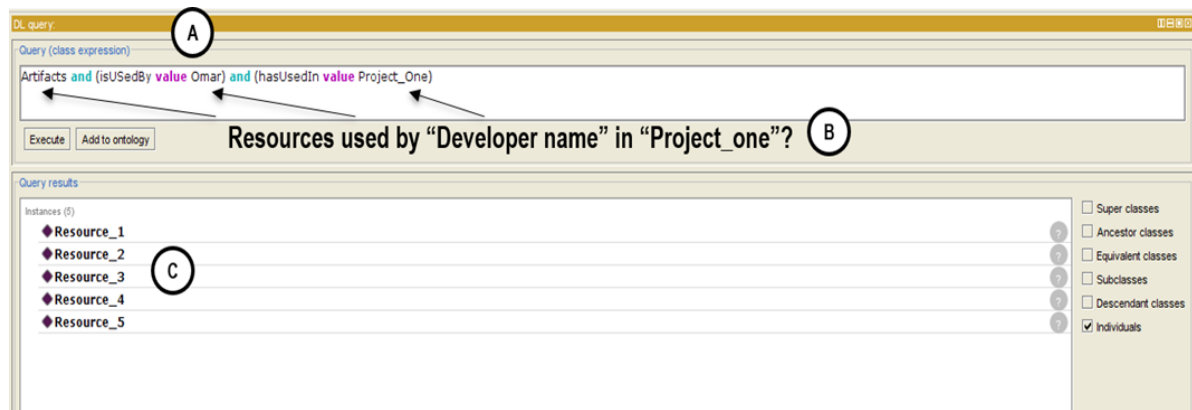


Figure 3.9 Competency Question example with Manchester OWL Syntax in Protégé.

3.3 Expertise Tools Module

The expertise tools module's primary goal is to provide various tools to condensate knowledge (R1). In Section 2.1, we identified the most popular sources among developers (e.g., websites, official documentation, forums, and video tutorials). In addition to this, it was identified that commonly regardless of the source used, developers use two main ways to capture this knowledge: bookmarks or filing on the computer. In the case of official documentation or programming forums, developers usually save pages using bookmarks. On the other hand, in many cases, developers externalize knowledge using documents such as word, notepad, pdfs, or downloading videos. Thus, this thesis proposed two tools to capture expertise in the coding phase during the software development process.

Next, we describe ExCap and B4U the tools proposed in this thesis to implement some of the semantic knowledge module elements. The goal is to provide developers with a user interface to capture artifacts from two sources (bookmakrs and digital documents).

3.3.1 ExCap: A tool to capture expertise from developers

Here we present Expertise Capture (ExCap), a tool to condensate expertise in software developer. ExCap uses all the digital artifacts that developers create and consume during a software development project. Figure. 3.10 shows a description of the current architecture of ExCap too, which consist of two layers: (i) Client Layer and, (ii) Server Layer.

i) In the Client Layer ExCap works as a background demon process, using digital documents as a source of expertise for the developers. In this sense, the tool contemplates any type of digital document to condensate such as code classes, manuals, books, or video tutorials. ExpCap tool was implemented to capture and search expertise in software organizations, this

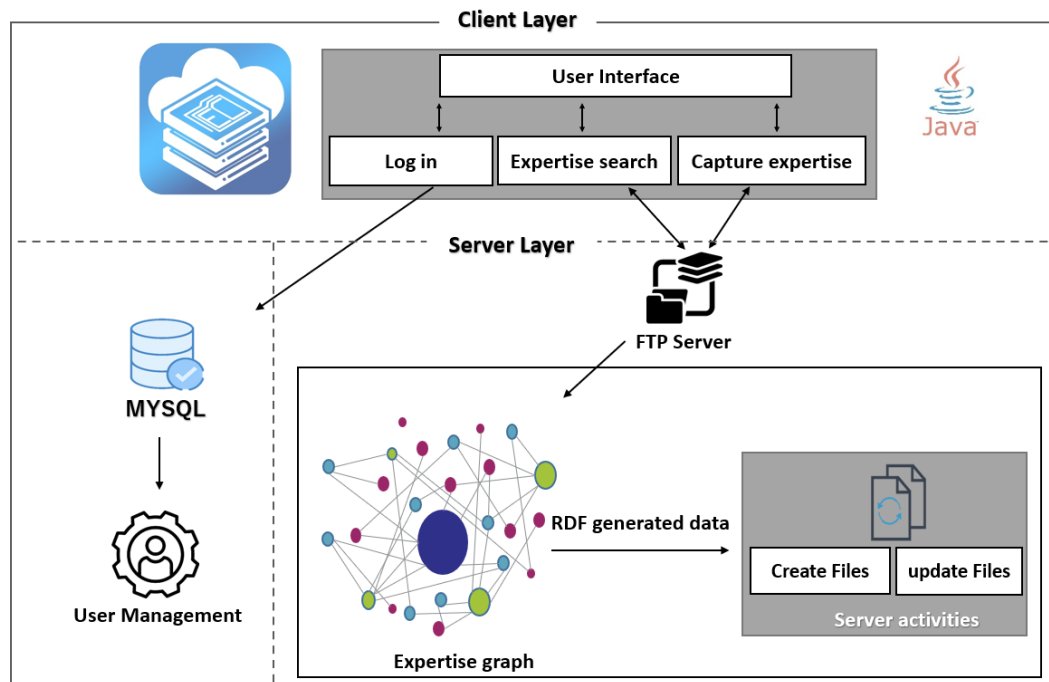


Figure 3.10 Architecture ExCap tool.

tool use a Java interface that makes it easier for the user to carry out these two activities; it uses a drag and drop function to facilitate the artifact sharing in the tool.

(ii) On the Server Layer, we have two different sections which help the application to perform its functions. First, we have a section that manages the user data, the ExCap connects to a database to perform basic functions such as login and project registration, it uses a communication protocol with the database through Hypertext Transfer Protocol (HTTP) and MySQL. Second, the server has an File Transfer Protocol (FTP) communication, which is configured directly to work directly with application to perform its functions; these functions consist of tasks such as filtering, uploading files and creating new projects. All the expertise share by developers by means of using ExCap is represented as an expertise graph, which consist of interlinked instances of the elements described in the semantic modelling (see Figure. 3.2). From the expertise graph, an RDF file is generate. RDF is data model, which is based on making statements about resources in expressions of the form subject-predicate-object commonly known as triples. The subject denotes the resource, and the predicate denotes aspects of the resources; moreover, it expresses the relationship between the subject and the object. Next, we describe the tool functionality, which consist in tabs that incorporate the elements from the semantic modeling presented.

Developers

First, the system requests a login before starting the daemon; otherwise, you must create a user, filling the necessary data (see Figure. 3.11).

The user profile helps us to keep a record of the expertise of a developer working in the organization. This record includes information about the artifacts produced and consumed by developers; also, where these artifacts were applied. This tab helps to create developer's instances from the model presented before, we used email and name as data values to give a description about the developer.

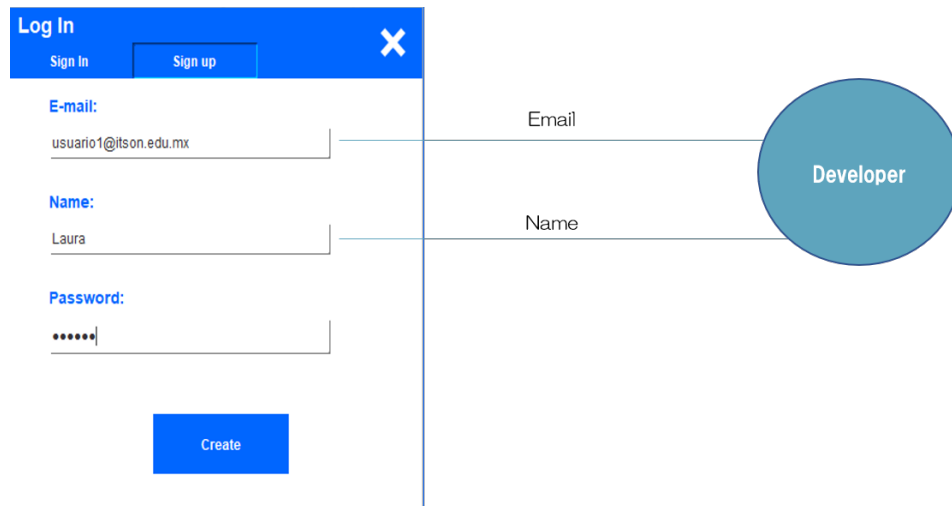


Figure 3.11 Developers semantic incorporation in ExCap.

Projects

In the projects tab the developer can create a new project description; the creation of projects is an essential aspect to link developer's experience with their projects. Developers can create or update current or past projects they have participated in. Projects are classified based on their platform (e.g., web, desktop or mobile).

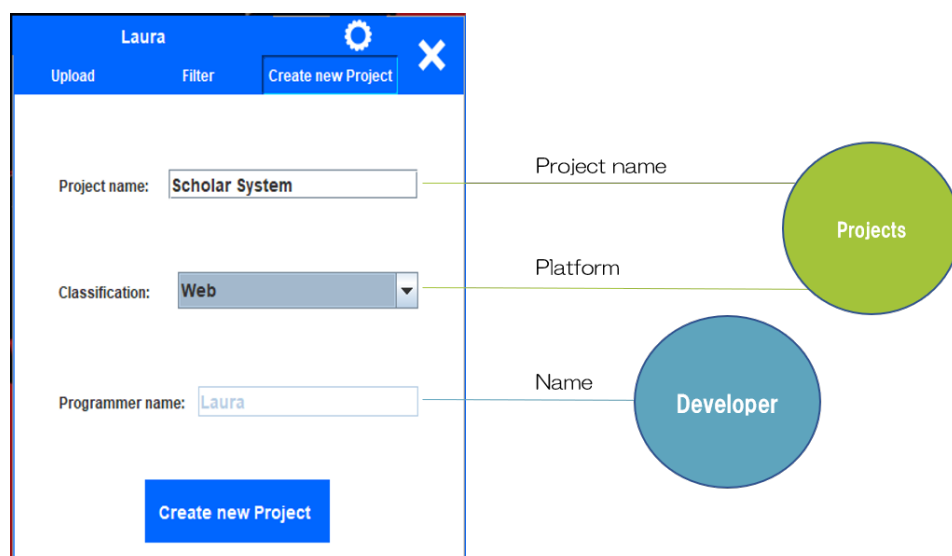


Figure 3.12 Project semantic incorporation in ExCap.

Artifacts

In the upload tab, user can capture knowledge, the user selects the file (drag) and drop into the tool. In this case, the tool considers digital artifacts: video tutorials, word or pdf documents which could be manuals, and any file that could help a developer to resolve a doubt or problem.

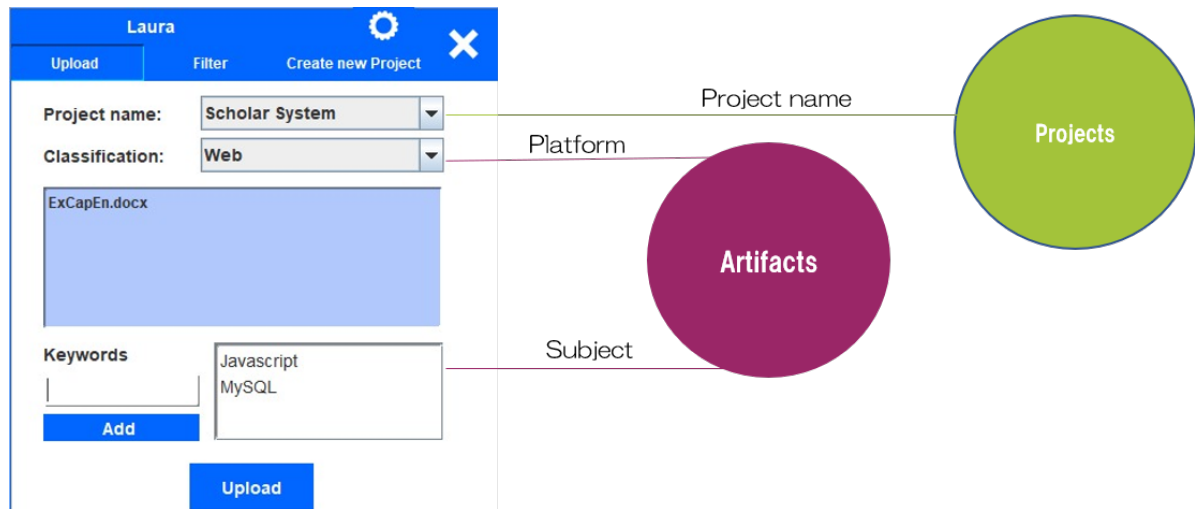


Figure 3.13 Artifacts semantic incorporation in ExCap.

As well as projects, artifacts are classified based on their platform; furthermore, we use keywords to associate these artifacts to a problem or topic.

3.3.2 B4U an extension to capture bookmarks

In this section we present Bookmarks for Us (B4U) an extension for Google Chrome capable of capturing a website and in addition to adding a category to it, as well as tags for easy classification. After several iterations it was possible to reach the final result that fulfills the function of sending the marker with all its data to a REST API.

B4U connects and saves bookmarks through an REST API which this stores the users and data saved by them. The values of the user and mail field are saved in the localStorage, therefore, when a user login the the user data is extracted from the localSotrage.

When the extension starts, it displays a login screen, as well as a user creation function; both screens request two main data: username and email.

Figure. 3.15 shows the main view of B4U and the elements used to capture bookmarks description which are URL, Type and Labels. URL:shows the user the URL that will be saved in the bookmark, this is automatically saved. Type: the bookmark should be part of one of three pre-made categories: Web, Mobile, or Desktop. Labels: the labels create a meta-description of the knowledge that the developers want to store.

The image shows two side-by-side forms for user authentication. The left form is titled 'Ingreso.' and contains the text 'Ingresa tu usuario y email.' Below this are two input fields: 'Usuario' with the placeholder 'Ingresa tu usuario' and 'Email' with the placeholder 'Ingresa tu email'. At the bottom are two buttons: 'Login' and 'Crear Usuario'. The right form is titled 'Registro' and contains the text 'Ingresa tu usuario y email.' Below this are two input fields: 'Usuario' with the placeholder 'Tu usuario' and 'Email' with the placeholder 'Tu email'. At the bottom are two buttons: 'Agregar' and 'Regresar'.

Figure 3.14 B4U extension user view.

Users can store all the bookmarks they need. An important aspect is that the more labels (keywords) the resource becomes easier to classify and describe for future consultations.

The image shows a form for adding a bookmark. At the top, it says 'Bienvenido/a Susana Oria' with a 'Log out' button. Below is a 'URL:' field containing 'https://developer.chrome.com/extensions/devguide'. Then, a 'Selecciona un tipo:' dropdown menu with 'Web' selected. Next is an 'Agrega una etiqueta:' field with the placeholder 'Agrega una etiqueta presionando enter.'. Below that, 'Etiquetas:' shows three tags: 'Chrome', 'Extensión', and 'Documentación', each with a red 'x' icon. At the bottom, a 'Confirmar marcador:' section contains an 'Enviar.' button.

Figure 3.15 B4U extension login view.

All the bookmarks store by a user represent an instance of an artifact within the knowledge of an organization.

3.3.3 Knowledge Condensation in Software Development Scenario

In this section we describe a scenario of the knowledge condensation process within a software development organization. The developer must initiate the tool and sign up to build a developers' profile (1); thus, every registered user generates an instance of a developer. Developers register their current or past projects (2); those projects will be associated with the current developer to describe her/his skills. Developers can share resources that might be useful to other colleagues (3). We consider digital artifacts such as manuals source code and video tutorials. Both artifacts and projects are classified based on a development platform: desktop, web or mobile. Furthermore, artifacts include keywords to relate resources with a particular topic or situation on which it can be used. Moreover, artifacts are associated to a particular project where a developer used them. The instances created within the ExCap and B4U tool are linked among each other; thus, expertise condensed turn into a graph with all the knowledge from the organization (4).

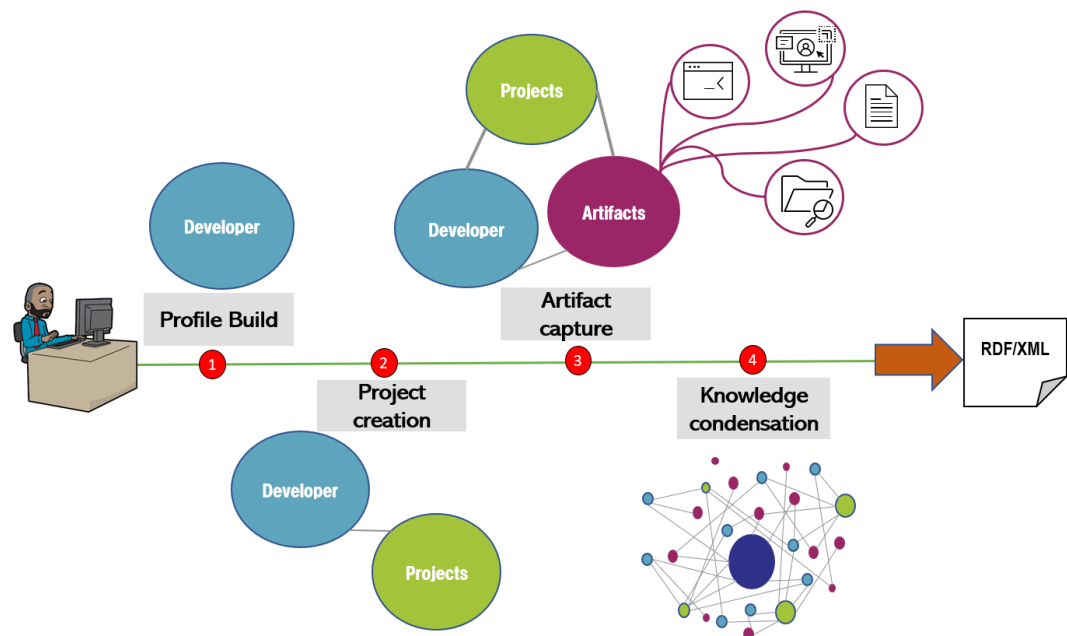


Figure 3.16 Expertise Condensation Scenario

3.4 Chapter Summary

This chapter presents a knowledge condensation model to support the expertise location process in software development organizations. The knowledge condensation model consists of three modules: Formal grammar, Semantic Knowledge, and Expertise tools. In the formal grammar module, an approach is made to formalism to describe how developers store and share their knowledge. An architectural knowledge model is proposed in the semantic knowledge module, which is implemented in an ontology for the coding phase in software development. In the

module of expertise tools, two prototypes were developed that implement the ontology elements developed as part of the semantic knowledge module's implementation.

Chapter 4

ROntDev: A methodology for developing ontologies

In chapter 1 we identified drawbacks on the current methodologies or method employed to develop ontologies. Therefore, in this chapter we present ROntDev, a methodology proposed for software engineer researchers to build ontologies. The proposed methodology aims to give software engineers a simplified ontology development process, which includes ontology fundamentals concepts with a step by step explanation. ROntDev consists of four phases: Specification, Modeling, Formalization, and Evaluation. Next, we describe the phases and the activities in our proposed methodology (see Figure 4.1).

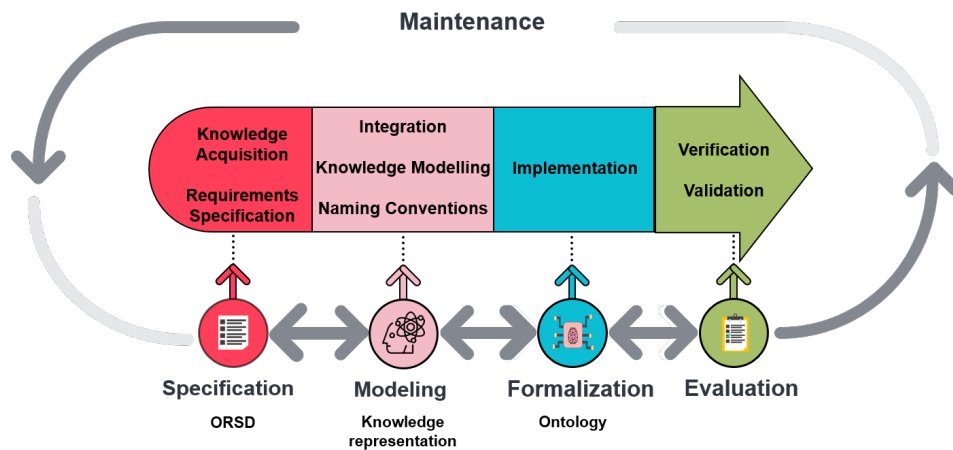


Figure 4.1 ROntDev ontology development life cycle.

4.1 Specification Phase

The specification phase establishes the requirements, which consist of the following ontology aspects: purpose, scope, implementation language, intended end users, and intended uses. Developers can achieve the aspects listed above by performing the following tasks.

4.1.1 Knowledge Acquisition

In software engineering, the key to project success lies in the software specification [103]. Therefore, we suggest following the Ontology Requirements Specification Document (ORSD) proposed by Suarez-Figueroa et al. [104] to formally document an ontology. Ontology engineers usually fill the ORDS using knowledge from diverse sources. We can extract the domain experts' knowledge using interviews, focus groups, or survey forms. Furthermore, we can use books, surveys, and literature review articles to extract knowledge of a domain. For example, the Software Engineering Body of Knowledge (SWEBOK) contains many software development terms and concepts [105].

In the case of extracting knowledge from the domain experts', there are some key aspects to focus when extracting knowledge from the domain experts: knowledge sources, knowledge topics, and knowledge flow. Identify the sources involved or used within the domain; this include information or data used or could be useful for performing the different activities composing the domain.

The knowledge topics refers to the identification of the main knowledge topics or areas related to the activities performed in the domain; for instance, knowledge required to perform the activities, or created from them. This is an important aspect to identify important knowledge topics not stored anywhere, or that are tacit knowledge from the domain experts.

The knowledge flow refers to the identification of the life-cycle of knowledge within the domain: how the knowledge is flowing through activities and processes. The ontology engineer must analyze the relationship between the knowledge sources and topics, to identify how these interact.

4.1.2 Requirement Specification

As mention before, we consider the guidelines proposed by Suarez-Figueroa et al. [106] during this phase. The ORSD helps to specify ontology requirements; moreover, it defines Competency Questions which we use forward to validate the ontology. We can perform the requirements specification and knowledge acquisition as parallels tasks. In Figure. 4.2 is shown a fragment of the ORSD template filled with the information for an ontology for the coding phase of software development.

At the end of the specification phase, we obtain a formal document that helps us be clear about the use of the ontology, the users, and the type of knowledge modeled. Moreover, the Competency Questions established in the ORSD will help us to evaluate the ontology.

4.2 Modeling Phase

Once we acquire the knowledge and fill the ORDS with the ontology specifications, we can start modeling a domain's description using the identified terms and concepts. The modeling phase focuses on organizing and model the acquired knowledge using a representation based

Ontology Requirements Specification Document Template	
1	Purpose
	Integration of the artifacts, projects, and experts in the code phase of the software development process
2	Scope
	The ontology has a focus just on the code phase of the software development process domain. The level of granularity is directly related to the competency questions and terms identified.
3	Implementation Language
	The ontology has to be implemented in OWL language using protégé ontology tool
4	Intended End-Users
	User 1. Programmer searching for resources to solve a problem (e.g. requirements, bugs or doubts with a process) User 2. Programmer searching for an expert to ask help User 3. Programmer searching for information about a project and his participants User 4. Programmer updating or registering his expertise (projects or resources)
5	Intended Uses
	Use 1. Register programmer profile Use 2. Register resource Use 3. Update expertise Use 4. Update profile information (projects and profile)

Figure 4.2 ORSD fragment of an ontology for the software development process.

on the OWL components; moreover, we include naming conventions guidelines to assure a semantic among ontologies developed with ROntDev methodology. The output of this phase is a knowledge representation model that implements the concepts of OWL. Therefore, this will ease the transition from a knowledge representation model to a machine-readable.

4.2.1 Integration

To avoid redundant information, we must be considered the reuse of ontologies (definitions already built). Researchers can consult ontologies in the following databases:

- Swoogle
- DAML Ontology Library
- ONKI Ontology Library Service
- Linked Open Vocabularies (LOV)

4.2.2 OWL Modeling

Before starting modeling, we need to learn the basic concepts of ontology descriptions; these concepts will simplify the transition from the conceptual model to a computational model using the Protégé tool.

An ontology describes concepts within a domain and the relationships that exist among them. OWL is one of the most common ontology languages. OWL¹ consists of a description of individuals, properties, and classes.

OWL uses classes to build descriptions of a domain; an owl class includes specifications needed by an individual to be part of a class. According to the example in Figure. 4.3, the *team* represents the persons involved in a software development project, *artifacts* represent the resources used during a software development project. The *project* class represents the projects where the organization is currently working.

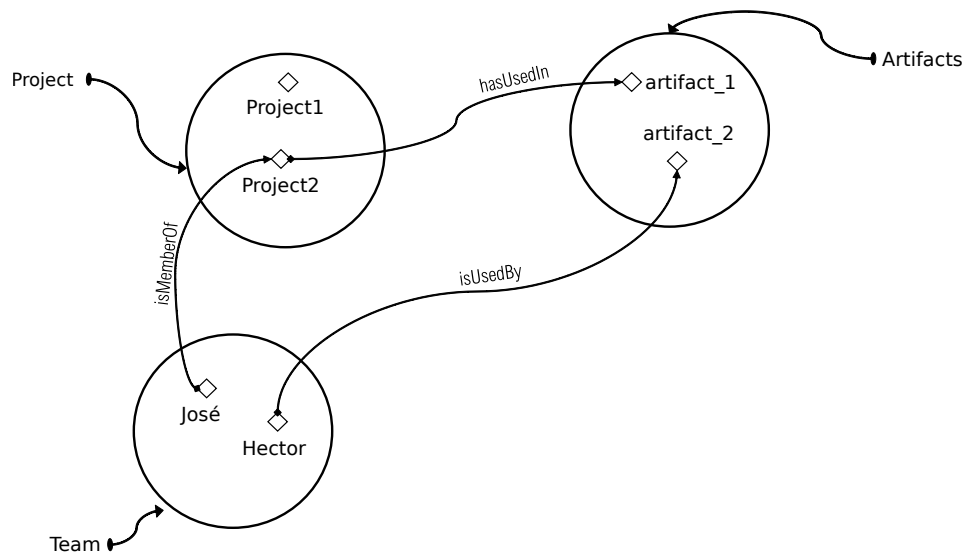


Figure 4.3 Representation example of classes, individuals, and properties of an ontology for software development process.

Individuals in an ontology represent instances from the represented domain (*classes*) following the example the “Jose” and “Hector” are instances from the *class team*, “Project1”, “Project2” are instances from the class projects, and “artifact1”, “artifact2” are instances from the *class artifacts*.

Finally, in OWL, properties are binary relations on individuals; they have different purposes, such as linking two *individuals*. For the example of the property “isUsedBy” links the “Hector” *individual* to the “artifact1” *individual*, which represents that “Hector” uses or creates the “resource1”.

¹<https://www.w3.org/OWL/>

4.2.3 Naming Conventions

Another essential aspect to know before modeling is the naming conventions. Naming conventions help avoid lexical inaccuracies and increase the robustness and exportability of an ontology, especially when vocabularies should be interlinked with other ontologies. Grangel-González et al. proposed a set of guidelines to use proper naming conventions based on the CamelCase notation [107]. CamelCase notation is a common practice in programming to write phrases without spaces or punctuation where the separation of words is indicated with a single capitalized letter, and the first word starting with either case. The following points describe the guidelines to create a naming conventions which are exemplified in Figure. 4.3:

- **Concepts as Single Nouns:** name all concepts as single nouns (e.g., DataBaseProject, Project).
- **Properties as Verb Sense:** Name all properties as verbs sense. The name of a property should be a plain noun phrase, in order to distinct from name classes (e.g., isUsedBy, hasUsedIn).
- **Short names:** provide short and concise names for the elements.
- **Conjunctions and ambiguous words:** avoid names with “and”, “or”, “Other”, “part” and those related to datatypes.

Figure. 4.3 was modeled using the guidelines mentioned above, the main characteristic is that properties start with a lower case letter and the Classes start with a capital case letter.

In the modeling phase, using the terms and concepts identified during the specification phase, we obtain a human-understandable model that uses elements from the Ontology Web Language (OWL); therefore, it will reduce the learning curve when implementing in Protégé².

4.3 Formalization phase

The formalization phase converts a conceptual model (taxonomy) to a formal model (computable). In ROntDev, we illustrate the transition from a knowledge model to a machine-readable using the Protégé tool.

4.3.1 Implementation

The implementation activity of using the knowledge representation model (see Figure 4.3) to define the classes' names, properties, and instances.

²<https://protege.stanford.edu/>

Classes

The principal class that represents all individuals is “Thing”. Thus all classes are subclasses of that one. Figure 4.4 shows the main classes of our ontology: Team, Artifacts, Project. The team class represents a developer team in an organization. The artifact class represents the resources used by developers. Finally, the Project class represents a description of the work and activities done by developers.

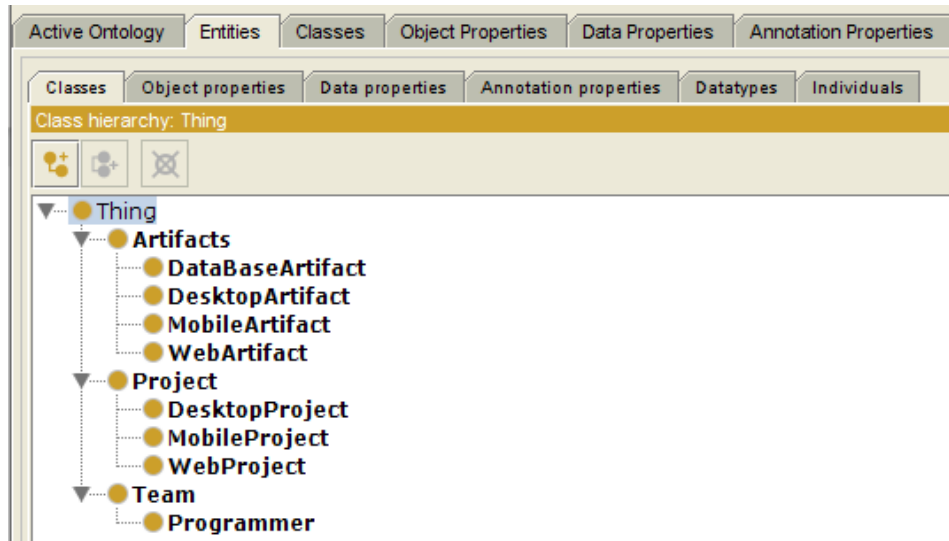


Figure 4.4 Representation example in Protégé of classes, individuals, and properties of an ontology for software development process.

4.3.2 Properties

Properties in OWL represent a relationship between two individuals. There are two types of properties: object and datatype properties. The object properties link an individual to another individual. The datatype properties link an individual to a data value expressed in Extensible Markup Language (XML) or Resource Description Framework (RDF).

Figure 4.5-B, the object properties are defined and classified within the category called has-Experience. Figure 4.5-A shows, we classify the data properties into two: artifactDescription and projectDescription.

Once we introduce the classes and properties into Protégé, at the end of the formalization phase we obtained a semi-formal ontology, which in the evaluation phase turn into a rigorously-formal.

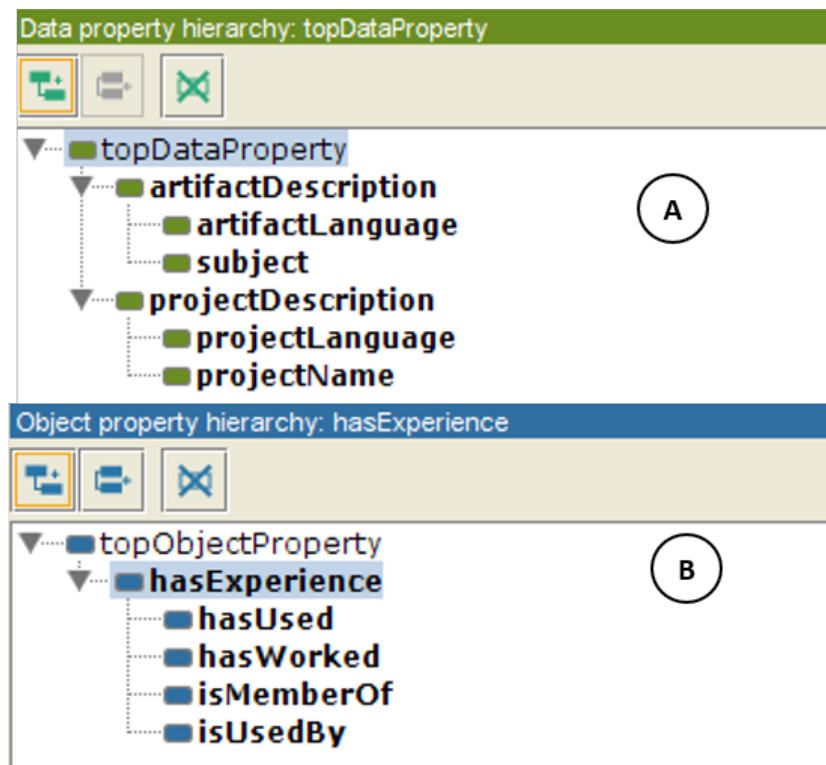


Figure 4.5 Screenshot of the properties defined in Protégé.

4.4 Evaluation phase

The evaluation phase is related to two important aspects of the ontology: quality, correctness. ROntDev consider the following tasks to evaluate the elements mentioned before.

4.4.1 Verification

This task is a technical process which is done to guarantee the correctness of the ontology, according to the specification requirements. The verification activity can be done using reasoner on Protégé. An ontology reasoner is a piece of software able to infer logical consequences from a set of asserted facts or axioms. Reasoners should play a vital role in developing and using an ontology written in OWL. Protégé has a wide range of reasoners such as Pellet, Hermit, Fact++, etc. Figure 4.6 shows a guide to the verification process in Protégé tool using Pellet reasoner ³.

In Figure 4.6 once we click "Start Reasoner" in the Protégé editor first, it checks whether there exists a model, then if the model satisfies the structure proposed. Once you initiate the reasoner, if the ontology does not have any incongruency or consistency, Protégé console shows the following output (see Figure 4.7).

³<https://www.w3.org/2001/sw/wiki/Pellet>

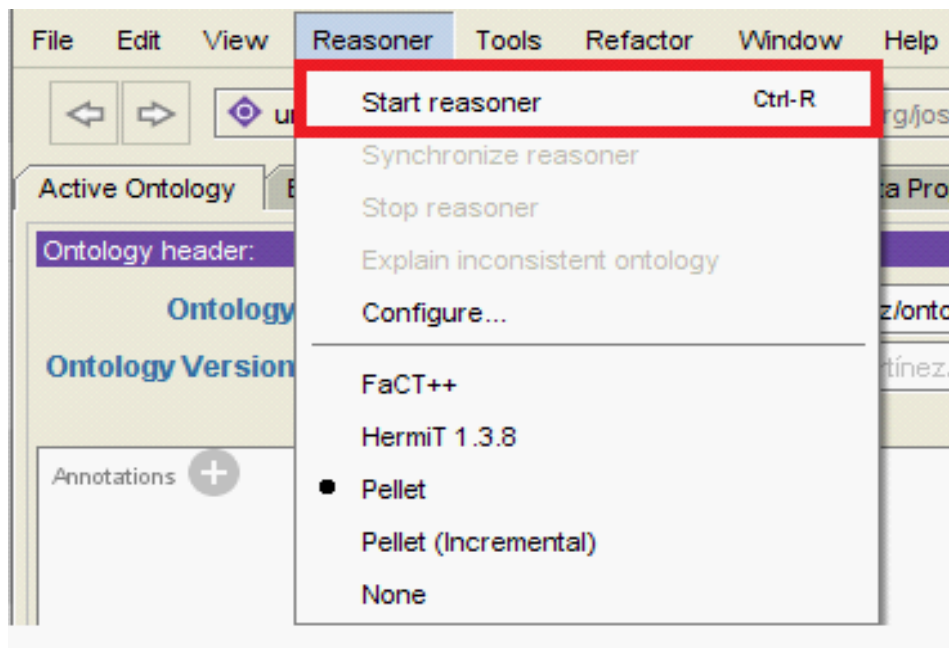


Figure 4.6 Verification process using a reasoner in Protégé

```

INFO 14:02:43 ----- Running Reasoner -----
INFO 14:02:43 Pre-computing inferences:
INFO 14:02:43   - class hierarchy
INFO 14:02:43   - object property hierarchy
INFO 14:02:43   - data property hierarchy
INFO 14:02:43   - class assertions
INFO 14:02:43   - object property assertions
INFO 14:02:43   - same individuals
INFO 14:02:43 Ontologies processed in 236 ms by Pellet
INFO 14:02:43

```

Figure 4.7 Reasoner output in Protégé using Pellet.

4.4.2 Validation

The validation ensures that the ontology fulfills the purpose. The process is done by using Competency Questions (CQ) [108]. CQ consists of a set of questions previously defined in the ORSD in a natural language (see Figure 4.8). The questions must be transformed into a computer language using Manchester OWL syntax to translate the questions in natural language into a computer language applied in Protégé. The ontology must be populated before the translation of the competency questions, which consist of the process of creating instances and their relation in an ontology.

a. Non-Functional Requirments
NFR 1. The ontology must support english language
NFR 2. The ontology must be based on multiple software organizations
b. Functional Requirements: Groups of Competency Questions
CQG1 (Developer Seeking)
CQ1. In wich projects "name" has been worked?
CQ2. Which resources has used "developers name"?
CQG2 (Resource Seeking)
CQ1. Resources for web developing?
CQ2. Resources used in "name" project?

Figure 4.8 Competency Questions section in ORSD

Figure 4.9 show the process of convert a natural language questions (CQ) into a query in Protégé. For example, we select a questions from the Developer seeking group from the Figure 4.8 (A), then the questions is translated into a query in Protégé (B). Finally, in the query results we select individuals to only see instances of the ontology populated (C).

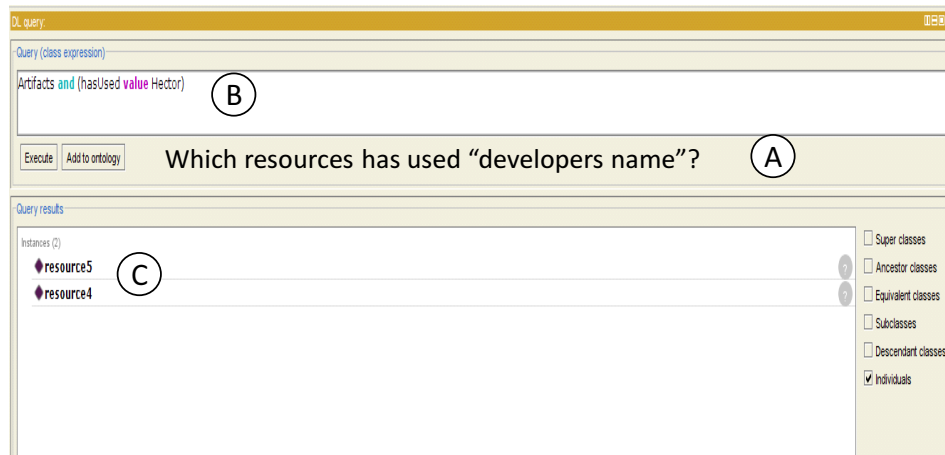


Figure 4.9 Competency Questions example with Manchester OWL Syntax in Protégé

4.5 Maintenance

Our proposal considers the maintenance as part of the iterative process, where the ontology goes through the whole process to modify or update elements of the ontology. We can extend or improve the ontology by adding or deleting classes.

4.6 Chapter Summary

Nowadays, ontologies represent a powerful tool for supporting the representation, processing, storage, and retrieval of the knowledge generated during the software development process. Although various ontologies have been proposed, little attention has been paid to the ontology

development process. Current methods and methodologies for building ontologies are either intended for experienced engineers or have background ontology concepts lack; moreover, some avoid the ontology evaluation. There are some essential elements to consider when building an ontology: (i) ontology concepts, (ii) ontology formalization, and (iii) evaluation. In this chapter, we present a methodology called ROntDev to build ontologies in the software engineering domain. Our methodology simplifies the transition from a knowledge representation model into a machine-readable using Protégé as an ontology editor; moreover, we illustrated how to perform an ontology evaluation. The ROntDev is illustrated by a case study of an ontology for the software development process.

Chapter 5

Evaluation of the Knowledge Condensation model

In Chapter 3, we presented the implementation of two mechanisms for the capture of knowledge. The mechanisms implement the elements of the ontology for the coding phase during the software development. The mechanisms' main goal was to evaluate the ontology elements' integration to capture different developers' artifacts. In this chapter, we present the method used to perform the evaluation. Furthermore, we include the results and the discussion of our findings.

5.1 Method

Next, we describe the method followed to evaluate the mechanisms ExCap and B4U extension. The method is structured as follows: objective, participants, materials, procedure, variables, and hypothesis.

5.1.1 Objective

To evaluate the potential users' perception of the mechanisms in terms of perceived usefulness and ease of use regarding the knowledge capture and classification.

5.1.2 Participants

To confirm our hypotheses, we form three empirical study groups. Two groups evaluated the proposed mechanisms (Excap and the B4U extension), while a third group evaluated the traditional tools used in software development. Thus, participants from the third group evaluated tools according to their preferred traditional tool to capture knowledge. In this sense, traditional tools are those that developers usually use to capture knowledge, such as bookmarks, notepads notes, folders, and wiki repositories (see Section 2.1).

The group from each study was composed of two contexts: academic and industrial. The academic participants were students from the software engineering career at the Technological Institute of Sonora (ITSON by his Spanish acronym). Student participants were selected from the last semester of the career since, at this point, they already been in contact with agile methodologies and with software development.

Regarding the industrial context participants, they were software developers with experience in agile and distributed development. All companies in the industry context are either fully dedicated to software development or have a department within their organization dedicated to developing software for the same organization.

In Figure. 5.1, we can observe the distribution among students and companies from the industrial context. There were 110 participants distributed among the three empirical study groups. 20 participants for the ExCap group (4 academic and 16 developers), 44 for the B4U bookmarks extension (23 academic and 21 developers), and 46 for the traditional tools group (25 academic and 21 developers). The average age was 21.41 for the participants from the academic context and 27.37 for the industrial context.

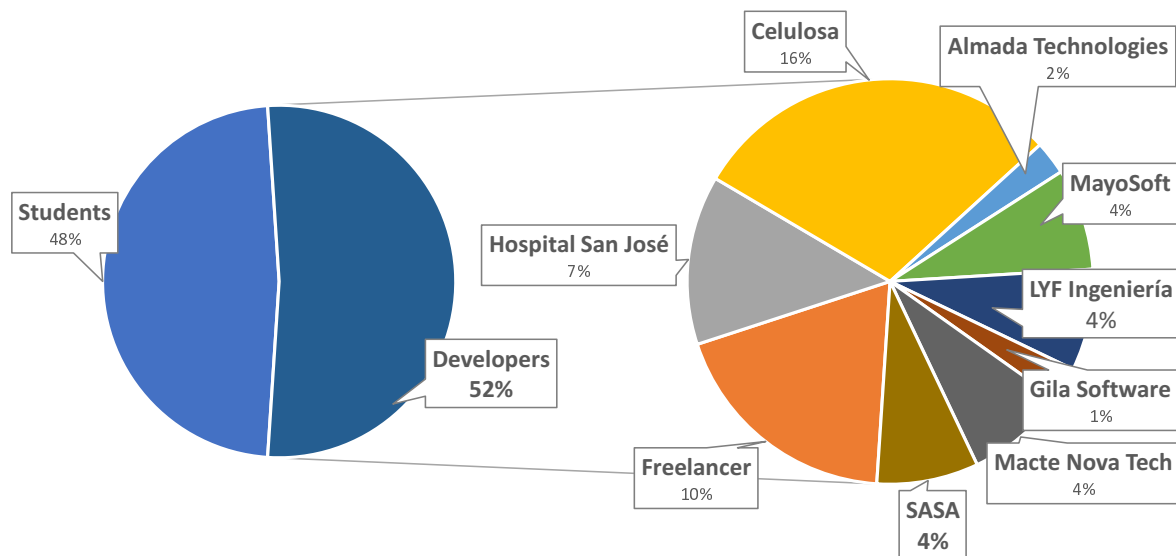


Figure 5.1 Participants Distribution

5.1.3 Instrumentation

To perform the evaluation, we used the following materials

- Excap interaction Guidelines: Participants were provided with a document that explains the ExCap tool's functionality and the instructions to do some knowledge capture and classification tests.

- **B4U extension interaction Guidelines:** Participants were provided with a document that explains the functionality of the B4U extension plugin. Moreover, the document includes the instruction to test the capture and classification of knowledge.
- **Slideshow presentation:** To evaluate the traditional tools, we use a PowerPoint presentation to describe the expertise location process. The aim was to discuss and evaluate the traditional tools used by the participants to capture knowledge.
- **Google forms Questionnaire B4U extension:** A TAM (Technology Acceptance Model) questionnaire (Likert-7) was prepared to focus on the B4U extension plugin regarding the capture and classification [109].
- **Google forms Questionnaire ExCap application:** A TAM (Technology Acceptance Model) questionnaire (Likert-7) was prepared to focus on the ExCap application regarding the capture and classification.
- **Google forms Questionnaire traditional tools:** A TAM (Technology Acceptance Model) questionnaire (Likert-7) was prepared to focus on developers' traditional tools to capture knowledge.

5.1.4 Procedure

As discussed before, we formed three empirical study groups which lasted approximately 1 and a half hour. Each for one of the proposed mechanisms (ExCap application and B4U extension) and for the traditional tools. Next, we describe the procedure followed by each group.

ExCap application group

- *Preparation:* A compressed folder with the following items was mailed to all participants: ExCap application, Guidelines for the interaction, and support material.
- *Execution:* First, we give an Introduction (within the Guidelines document) to participants explaining the experiment's objective. Then, following the ExCap application guidelines document, the participants began to interact with the extension as indicated in the guide. Finally, participants answer a questionnaire to evaluate ExCap application.

B4U extension plugin group

- *Preparation:* First, we verified that the participants were connected to the session and could listen. Subsequently, we shared a compressed file with the materials needed for the experiment. Finally, we verify that participants could visualize the presentation to contextualize the problem addressed.

- *Execution*: First, we give an introduction to participants explaining the experiment's objective. Then, following the B4U extension guidelines document, the participants began to interact with the extension as indicated in the guide. Finally, participants answer a questionnaire to evaluate B4U extension.

Traditional tools group

- *Preparation*: First, we verified that the participants were connected to the session and could listen. Subsequently, we verify that participants could visualize the presentation to contextualize the problem addressed. Finally, we give participants a PowerPoint presentation to give context to participants with the expertise location process.
- *Execution*: First, we give an introduction to participants explaining the experiment's objective. We then discussed the traditional tools that participants use to capture knowledge during a software development project. Finally, participants answer a questionnaire to evaluate the traditional tools used by each participant.

5.1.5 Variables and Hypothesis

This section defines the variables: dependent and independent—we include how to measure them. Moreover, we define the hypothesis for the evaluation.

The independent variables were the mechanisms to capture and classify knowledge: ExCap, B4U extension, and the traditional tools. The dependent variables were the following:

- *Usefulness*: the degree to which a person believes that using a particular system would enhance their job performance.
- *Ease of use*: the degree to which a person believes that using a particular system would be free from effort.

Both dependent variables were measured using a questionnaire with 12 questions (six for usefulness and six for ease of use). Finally, based on the following dependent variables, we established the following hypothesis.

Usefulness

- H1_a: There is a difference between the mechanisms proposed over traditional tools regarding the usefulness to capture and classify knowledge.
- H1₀: There is no difference between the mechanisms proposed over traditional tools regarding the usefulness to capture and classify knowledge.

Ease of use

- H2_a: There is a difference between the mechanisms proposed over traditional tools regarding the ease of use to capture and classify knowledge.
- H2_o: There is no difference between the mechanisms proposed over traditional tools regarding the ease of use to capture and classify knowledge.

5.2 Results

In this section, we describe the results obtained during the three empirical group studies. As mentioned before, each tool's evaluation was conducted using a (Likert 7-scale) questionnaire based on the TAM. Values and interpretation for possible answers are as follows: extremely likely (7), quite likely (6), slightly likely (5), neither (4), slightly unlikely (3), quite unlikely (2), and extremely unlikely (1). First we analyze our data using a box plot (see Figure 5.2), we aimed to obtain a visual representation of the data distribution among the different mechanisms to capture and classify knowledge. Furthermore, Table 5.1 reports overall descriptive statistics of the TAM results for each tool.

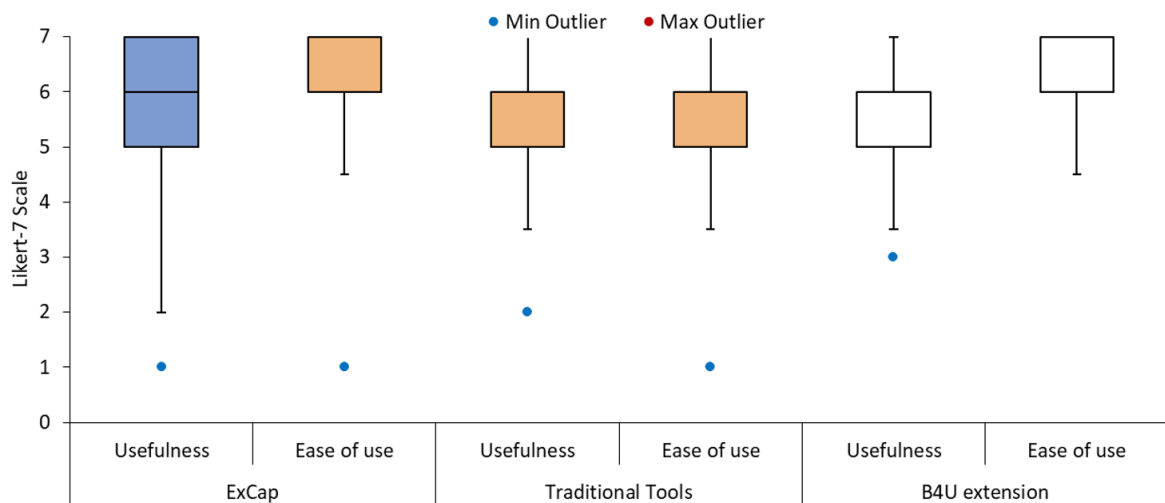


Figure 5.2 Box Diagram of the evaluation results

From the ExCap application evaluation, the TAM results show a median and mode of 6 (quite likely) for the usefulness construct. For the ease of use construct, we obtain a median of 6 and a mode of 7 (extremely likely). Thus, in general, participants perceived ExCap application as (quite likely) useful and (quite likely) usable.

From the B4U extension plugin evaluation, the TAM results show a median and mode of 6 (quite likely) for the usefulness construct. For the ease of use construct, we obtain a median of 6 and a mode of 7 (extremely likely). Thus, in general, participants perceived the B4U extension plugin application as (quite likely) useful and (extremely likely) usable.

In contrast to the mechanism proposed, traditional tools TAM result show a median and mode of 5 (neither) for both constructs (usefulness and ease of use). Our results showed a significant difference of improvement regarding usefulness and ease of use for knowledge capture and classification.

Table 5.1 Descriptive statistics of the TAM evaluation results

Construct	TAM evaluation results from ExCap						
	Median	Mode	Min	Max	Percentile 25	Percentile 50	Percentile 75
	Perceived usefulness	6	6	1	7	5	6
Perceived ease of use	6	7	2	7	6	6	7
Construct	TAM evaluation results from Traditional tools						
	Median	Mode	Min	Max	Percentile 25	Percentile 50	Percentile 75
	Perceived usefulness	5	5	2	7	5	5
Perceived ease of use	5	5	1	7	5	5	6
Construct	TAM evaluation results from B4U extension						
	Median	Mode	Min	Max	Percentile 25	Percentile 50	Percentile 75
	Perceived usefulness	6	6	3	7	6	6
Perceived ease of use	7	7	2	7	6	7	7

To verify the validity of our findings, we performed Mann-Whitney U tests. We selected The Mann-Whitney U test because the data do not follow a normal distribution. We tested each mechanism compared to the traditional for both constructs: usefulness and ease of use; Table 5.2 shows the overall results from the Mann-Whitney U test.

From the mechanisms compared to traditional tools on the perceived usefulness construct, we obtain the following results (U-value = 14842, Z-score = -1.94879, p-value = 0.022559, alpha = 0.05) for the ExCap application and (U-value = 31665, Z-score = -3.01737, p-value = 0.00126, alpha = 0.05) for B4U extension plugin. The Mann-Whitney U tests on the perceived usefulness construct indicate a significant difference between results from both mechanisms compared to traditional tools; participants perceived the mechanisms as useful. Thus, we reject the H_{1_0} usefulness.

From the mechanisms compared to traditional tools on the perceived ease of use construct, we obtain the following results (U-value = 10911.5, Z-score = -5.63576, p-value = 0.00001, alpha = 0.05) for the ExCap application and (U-value = 24161, Z-score = -7.09082, p-value = .00001, alpha = 0.05) for B4U extension plugin. The Mann-Whitney U tests on the perceived ease of use construct indicate a significant difference between results from both mechanisms compared to traditional tools; participants perceived the mechanisms as usable. Thus, we reject the H_{2_0} ease of use.

Table 5.2 Mann Whitney test comparison

	ExCap	Traditional tools	B4U	Traditional tools
	Usefulness			
Z-score	-1.94879		-3.01737	
U-value	14842		31665	
p-value	.022559		.00126	
	Ease of use			
Z-score	-5.63576		-7.09082	
U-value	10911.5		24161	
p-value	.00001		.00001	

5.3 Discussion

This section discusses the results presented in the previous section. We divided the discussion into three aspects: Perception of the mechanism and proposed improvements, Traditional tools, and Expertise location and knowledge needs. An important aspect to highlight is that the result and discussion during this thesis are regarding knowledge capture and classification using our proposed mechanisms.

Perception of the mechanisms and proposed improvements

As mentioned in the previous section, we obtained significant results from our proposed knowledge capture mechanisms. However, despite the significant results obtained, there are some crucial aspects to discuss. The low values are an essential aspect to discuss from our results, the low values from participants regarding the capture and classification using our mechanisms. The main reason that can be associated with the low values is the participants' experience in the software development industry. As we mentioned, evaluation was composed of groups from two contexts (academic and industrial). In the academic context, students are not yet aware of the demands of the professional software development environment where they constantly need to produce faster and better. Thus, students may be not perceived the importance of reusing knowledge from previous projects and with the expertise location process.

We mentioned that our results are regarding the capture and classification of knowledge. Thus, it is necessary to discuss the relation of our results with the knowledge condensation model. We evaluated the mechanism's perception in terms of usefulness and ease of use regarding capture and classification. The capture and classification use the elements from the ontology developed for the coding phase during the software development. Since the capture and classification are based on our model, the following implies the participants accept capturing and classifying knowledge using a semantic model. In the future, the search implementation of the mechanisms will ease the expertise retrieve.

Finally, despite that, our main objective was to evaluate a capture and classification using mechanisms based on a semantic knowledge model. We received several suggestions to improve the mechanisms (ExCap and B4U extension). In the case of the ExCap application, participants suggested the following improvements:

1. Select multiple files at the same time and not individually.
2. Upload files from a web browser.
3. Filter implementation for the search.
4. Improve the user interface by making it more intuitive

Furthermore, in the case of the B4U extension plugin, participants suggested the following improvements:

1. Add a history or list of created tags.
2. Autocomplete with tags already created previously.
3. cleaner and more minimalist design.
4. Use folders or categories to cluster.

Traditional tools

During the evaluation of traditional tools, we had the opportunity to discuss the tools they use to capture knowledge with the participants.

The most critical aspects to discuss are the type of tools used and the low values obtained in the evaluation. Participants reported various ways of capturing knowledge (externalization process). As mentioned above, there are two types of explicit knowledge: documented and formalized. Documented knowledge tends to predominate due to agile practices. Developers use or adapt different tools to capture knowledge. Some tools such as notepad notes, repositories, or conventional bookmarks help capture knowledge. Using these tools, developers externalize knowledge for their particular use—problem-solving during a software development task. Some tools such as notepad notes, repositories, or conventional bookmarks help capture knowledge. However, there are still limitations to these or difficulties in terms of use. When we talk about markers, developers report that they forget the name they saved the marker or its objective on many occasions.

Although some participants use formalized knowledge, they report that their tools are challenging to learn to use (e.g., wiki repositories). Thus, this could be related to the lower values obtained from the evaluation of the traditional tools.

Expertise location and knowledge needs

An essential aspect of this thesis work is the expertise location process. During the evaluation of the mechanisms, we identified some insights about the knowledge needs of developers.

At the beginning of the evaluation, some of the assumptions were related to the knowledge needs and the expertise reuse. During our evaluation, we find out that developers create artifacts during software development projects. However, the way developers externalize knowledge hinder their retrieve in future projects or share among colleagues. The following leads to time wasted already solving problems. Furthermore, developers externalize knowledge for their use; thus, if a developer is unavailable, their knowledge became inaccessible and prone to vaporize.

As developers gain more experience during projects. They begin to systematize their knowledge reuse process to produce better and faster. Therefore, a semantic knowledge capture will help classify and condensate the knowledge generated by developers, as we observe from the proposed mechanisms' results. Furthermore, the knowledge generated could train novice developers, reduce time on solving problems, and reduce knowledge vaporization when experts are unavailable.

5.4 Chapter Summary

This chapter discusses the process carried out to evaluate the mechanisms for capturing and classifying knowledge. These mechanisms implement elements of the ontology for the coding phase developed. The assessments were conducted using 3 different groups made up of developers and students. Each group followed a protocol to interact with a mechanism for the capture and classification of knowledge. Finally, the results obtained show an acceptance in the proposals of our knowledge capture and classification mechanisms.

Conclusions

In this thesis, we described the construction process of a knowledge condensation model. The main goal was to support the expertise location process to reduce the knowledge vaporization caused by agile methods' preference for tacit knowledge. We followed the knowledge condensation definition for the construction of the present model. As a result of the construction process, different aspects of the expertise location process were analyzed, leading to the following leading contributions:

- The design of an expertise location process map.
- The semantic knowledge model for the software development process.
- The ontology for the capture of knowledge in the coding phase during the software development process.
- The formal grammar for the knowledge description.
- The methodology for ontology development focused on non-expert ontology engineers.
- The mechanisms to capture knowledge that implements the elements from the ontology.

In this chapter, we discuss the main findings obtained as a result of this thesis and provide directions for future research opportunities.

Main Findings

Throughout this thesis, we have introduced a knowledge condensation model to support expertise location during the software development process. The model consists of three modules: *formal grammar*, *semantic knowledge*, and *expertise tools*.

This section discusses the main findings obtained from the overall construction process of the knowledge condensation model and present the general conclusions.

Formal Grammar Module

The behavior of the developers is highly unpredictable. They generate new knowledge from the combination of this with previous resources. Developers can use or use a resource differently.

From this, they can generate new knowledge, be it an implementation of a code fragment from one language to another, a new use of an explanation from a theoretical book for another area or language.

Current systems or capture mechanisms do not acknowledge this type of knowledge reuse because it is complex to model. Moreover, with each new knowledge acquired, they increase their technical problem-solving skills.

In the Formal Grammar Module, we propose a language based on tuples; we focused on developers' contributions, representing the artifacts generated by a developer during a software development project. The tuples help to describe the capture, search, and knowledge update of developers.

Our goal is to provide an insight into a way to model the way developers continue to find resources. Moreover, to establish a formalism to describe the process of externalization and combination of knowledge. Therefore, we expect that researchers propose or implement some of the behaviors described in the Formal Grammar Module.

Semantic Knowledge Module

This module presented a semantic knowledge model for *structured* and *unstructured data*; the goal is to search and centralize applications, databases, and files. Proposals accumulate expertise accumulate without the organization being aware of their existence, and since each proposal uses its inputs and outputs cannot be centralized. In this sense, semantic knowledge modeling could be a way to link experts and the artifacts developers produce and consume during software development.

Furthermore, we developed an ontology for the coding phase during the software development process. We present a description of an ontology's development process (formalized). Furthermore, we described the evaluation, which means that the model is ready to implement in a system. The ontology's main contributions are the support to expertise location through an ontology that can link the information about programmers or any team member with the resources used in a project. Therefore, the developer will identify the provider or the artifact or developers' source to solve problems or doubts in a specific domain. It will mitigate the time wasted trying to find solutions to solve problems or doubts; consequently, the knowledge reuse will reduce the architectural knowledge vaporization.

Finally, during the ontology development, we found several issues with the Methontology framework—lack of description in the translation process from a taxonomy to an ontology IDE and in the evaluation process. Thus, we reviewed the methods and methodologies to build ontologies. We found that current approaches (methods or methodologies) hold some drawbacks for software engineer researchers. Most approaches present a lack of details on performing tasks. Some approaches cover focus only on knowledge modeling; moreover, the evaluation is an activity often omitted. Due to these drawbacks, current approaches do not guarantee a rigorously-formal ontology level. Finally, an important aspect omitted

by proposals is the naming convention guidelines. Naming conventions could enhance the semantic application to support ontologies integration. Therefore, we present RontDev, a methodology to build ontology focused on non-expert ontology engineers.

ROntDev considers the limitations identified in the current approaches. We have found that ontologies can contribute to any phase of activity in the software development process. Thus, we expect that the presented methodology helps software developers with the learning curve and complexity when building ontologies. Furthermore, since we include naming conventions to enhance the semantic applications, as future work, we plan to include a detailed description of the integration of existing ontologies during the conceptualization phase.

Expertise Tools

During software development, developers generate knowledge; this knowledge is externalized (documented or formalized) for their reuse. In agile environments, teams only externalize what they consider sufficient to understand the project. However, despite the preference for tacit knowledge over explicit knowledge in the agile paradigm, developers externalize knowledge for their particular use during their tasks. Thus, organizations are constantly producing knowledge that could be valuable in future projects to improve the quality. Therefore, this thesis work aims to capture and classify knowledge to ease their retrieval.

In the expertise tools module, we present the first efforts to implement the knowledge condensation concept, where the classification mechanism is based on an ontology formally obtained and validated. Consequently, using an ontology enables automated reasoning about architectural knowledge (artifacts and experts), reasoning with concepts and relationships similar to how humans perceive interlinked concepts, and a model that evolves with data growth without affecting processes. The mechanisms proposed in this module are ExCap, a tool to capture digital artifacts (e.g., manuals, code files, and videos), and B4U extension, a plug-in to capture bookmarks from the web browser. Our results show that the participants receive useful and easy use of the proposed tools to capture and classify knowledge.

Therefore, this implies that using capture mechanisms based on the condensation of knowledge will capture and classify the different types of knowledge (documented and formalized), even when these have not been formally captured due to preferences in the agile paradigm. Moreover, the mechanisms based on our semantic knowledge model will centralize valuable data for the organization.

Future Work

Diverse aspects of the knowledge condensation model have been analyzed in this thesis, revealing future research opportunities to extend this work. Next, we describe some of these opportunities regarding three different aspects: (i) knowledge search and retrieval, (ii) ontologies development, (iii) knowledge update and construction.

(i) Since we obtain positive results regarding the capture and knowledge classification as future work, we plan to extend the proposed mechanism's functionality to perform an evaluation that includes capture, classification, and retrieval of knowledge.

(ii) Another opportunity area is the knowledge update and construction. Developers generated a significant amount of knowledge. In the formal grammar module, we describe that developers update their knowledge every time they solve a doubt or problem since they find or generate resources. In this sense, we plan to use machine learning techniques to analyze the update and construction of knowledge.

(iii) Finally, regarding the ontologies development. As future work, we plan to include a more detailed description of the ontology's implementation from a technological perspective.

Appendix A

Focus Group Study Interview Guidelines

Expertise location process in software development organizations

Focus group objective: Understand the expertise location process within software developers' teams: capture, retrieve, and share knowledge. Moreover, to understand the expert location process.

Research Question: How do developers perform the expertise location during a software development project?

Assumptions List:

- Developers do not share their work.
- Developers do not label the knowledge captured.
- Developers don't ask for help
- There is no tool to manage knowledge in software organizations

Focus group protocol

Duration: 1 hour

INDIVIDUAL KNOWLEDGE SEARCH

How do you seek or find knowledge?

How you capture knowledge?

Do you use a specific tool to capture?

- If so
 - ¿What tools do you use?

KNOWLEDGE SHARE

How do you transfer knowledge?

Do you transfer knowledge?

What means do you use to share knowledge?

EXPERT SEEKING

What happen when your knowledge is not enough?

Do you usually seek for an expert?

Which elements do you consider for identifying an expert?

Appendix B

Guidelines the ExCap interaction

ExCap Application interaction Guidelines

Welcome participant,

Today's activity aims to test and evaluate ExCap (expertise capture) application, our proposed mechanism to capture and classify knowledge. ExCap help to capture and classify knowledge from digital artifacts such as source files, pdf, word documents, and any text file.

Before we start, we will talk about the problem that our application addresses. Currently, knowledge plays an essential role within software development companies. This knowledge helps developers solve problems or doubts during the development process. Companies try to reuse knowledge; however, due to tacit knowledge preference over explicit knowledge in agile development environments, this reuse of knowledge is challenging. In some cases, over time, this knowledge is prone to vaporize. Our proposed mechanism seeks to reduce this knowledge vaporization by capturing and classification knowledge from digital artifacts.

Next, we describe an activity scenario for the interaction with the ExCap application:

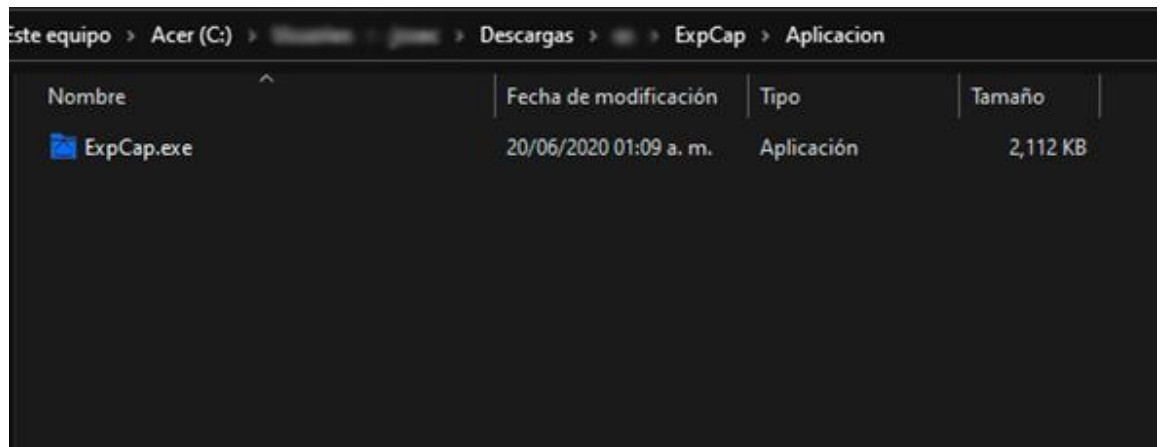
Imaginemos que acabas de participar en dos distintos proyectos de software, los cuales son los siguientes:

- 1- Development of an application that captures the sales of three stores that make individual and total sales reports.
- 2- A mobile application which helps a company to sell its products, generate electronic notes and support distributors with orders and collections. Additionally, it includes a system to generate sales reports.

While working on these projects you faced various problems which made you investigate and generate greater knowledge. This knowledge was stored in resources generated from a technical solution. Next, using ExCap you will capture and classify these resources generated from these situations during the projects.

Now, it will be explained the in detail how to install the application as well as the interaction (**Note**: we only implemented the capture and classification of knowledge. Therefore, resources captured are not visualized).

- Open ExCap Application. You can find it in the "Application" folder within the compressed file that we provide.



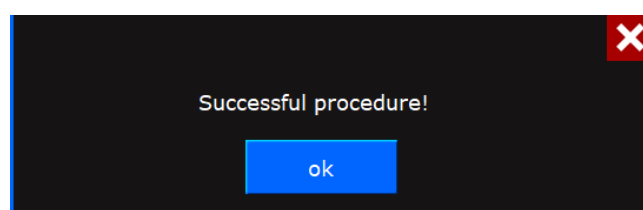
- Since you are a new user, you will have to create a user. Go to the "Sign up" tab and fill in the requesting fields, the email and password will be your access data. Once the information is filled, press the "create" button.

The screenshot shows a 'Log In' dialog box with a blue header and a close button (X). The 'Sign up' tab is selected. The form contains the following fields:

- E-mail:** ejemploExpCap@hotmail.com
- Name:** Ejemplo
- Password:** (masked with dots)

A blue 'Create' button is located at the bottom of the form. The Windows taskbar at the bottom shows the time as 03:35 p. m. on 18/06/2020.

- Once we create the user, ExCap will show a successful operation message and we will return to the beginning.



- Now you need to login with your new user in the Sign in tab.

Log In

Sign In Sign up

e-mail:

Password:

Forgot your password?

Log In

03:32 p. m.
18/06/2020

- Once you sign in ExCap will show you the following interface.

Ejemplo

Upload Filter Create new Project

Project name: ---

Classification: ---

Keywords

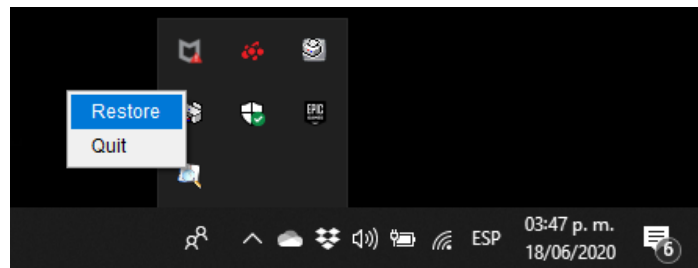
Add

Upload

03:37 p. m.
18/06/2020

- Now we ask you to test captures within ExCap using the resources provided in the commit file.

- First you need to create two projects, where you were working. Give the following names to them “Sistema de tienda” and “Sistema de cobranza”, the first Project is a desktop software and the second is hybrid (web and desktop).
- Add the information to each project. In the "support material" folder you will find the resources generated during the projects described in the scenario description, now select the indicated project, and drag the material to the blue box and fill in the corresponding fields.
- Use keywords related to the project, file, or the topic of knowledge.
- Repeat this step for all the resources provided for the activity.
- Don't worry if the application closes suddenly, ExCap Works as a widget always running in background.



- Close ExCap by clicking on the “Quit” button to close the process:



Once you have finished the test, we ask you to please answer the following form:

<https://forms.gle/STzDRyv4z3YWwMR89>

Appendix C

Guidelines for the B4U extension interaction

B4U extension interaction Guidelines

Welcome participants,

Today's activity aims to test the B4U extension plugin (Bookmarks for us), our proposed mechanism to capture the knowledge. B4U extension help to capture and classify knowledge using bookmarks from the web browser.

Next, we describe a scenario to describe the problem addressed:

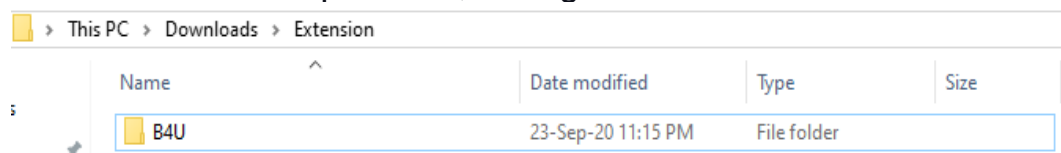
Let's imagine that your project leader commissioned your team and you to make a mobile application, of which you and another colleague were asked to make:

1. A GPS functionality to locate hospitals near the user's neighborhood.
2. How to visualize the closest possible route to the closest hospitals.

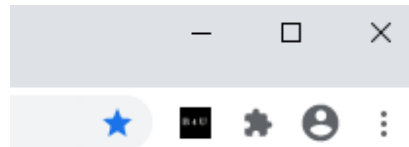
You should search the internet for the solution to the problem, and you find a useful site to get closer to the solution, use the extension to create a bookmark. **Participants must create at least seven markers with three labels each.**

Now it will be explained in detail how to install the extension as well as how to use it (**Note**: we only implement the capture and classification of the plugin. Therefore, bookmarks saved are not visualized):

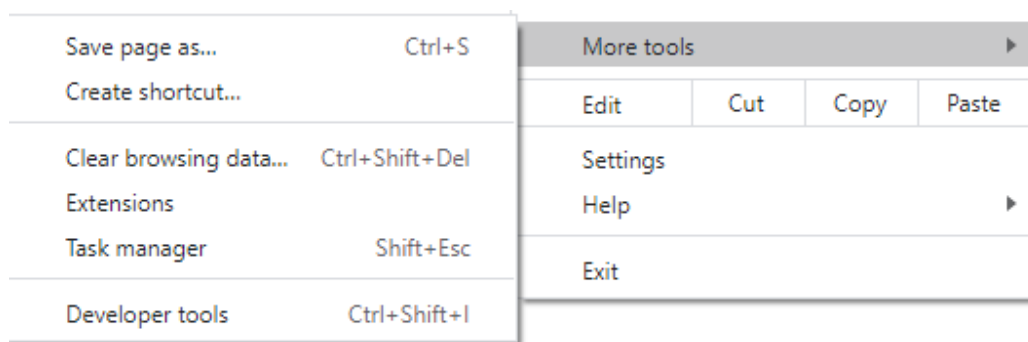
1. Download and unzip the file, storage it in an accessible route



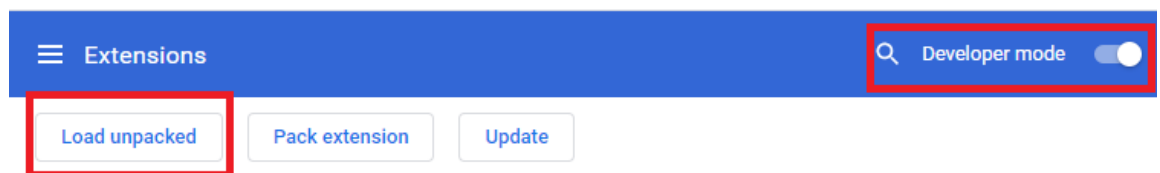
2. Access your Google Chrome browser (or Chromium if you prefer) and once inside, click on the three vertical points in the upper right corner.



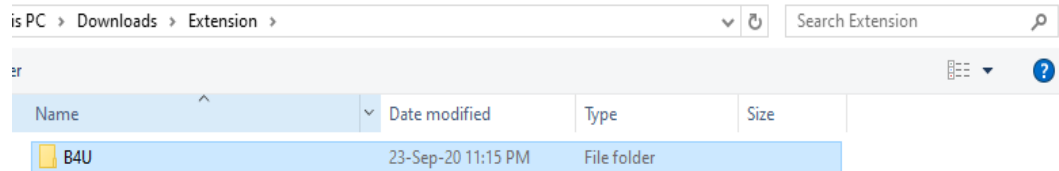
3. Then, click on More tools > Extensions.



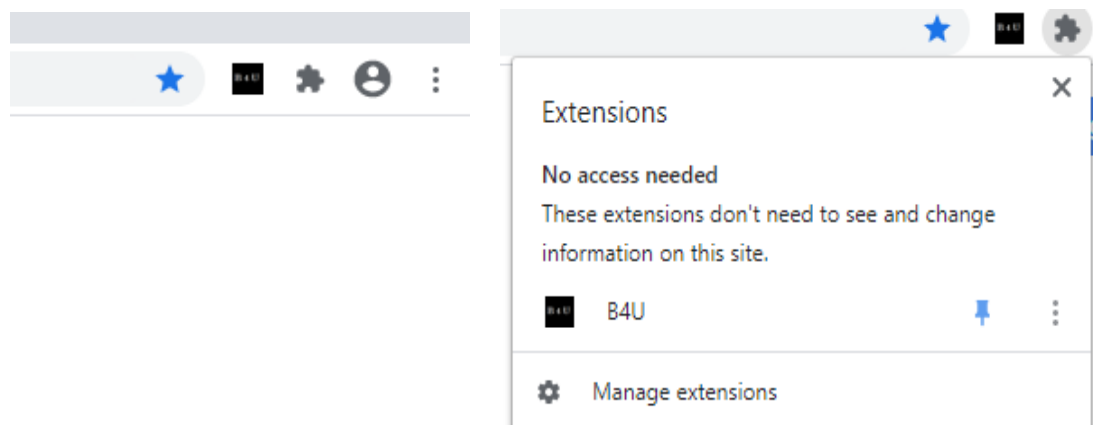
4. Once you get to the extension section, enable the developer mode and click on load unpacked.



- Now you just must find the folder where the extension is saved and select the folder.

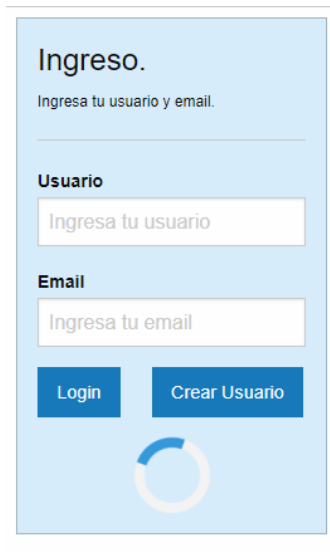


- The extension should appear in the upper right part of the browser (if for some reason it does not appear, press the puzzle icon and mark the extension).



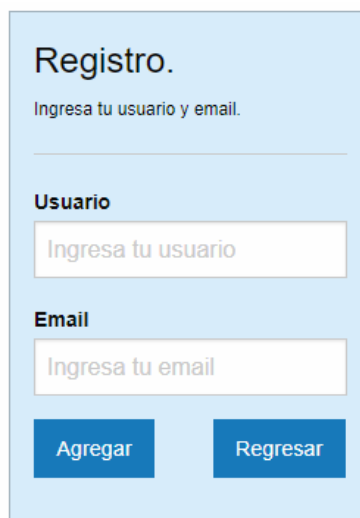
Next, we describe how to interact with the extension plugin. Ahora se explicará cómo utilizar la extensión.

1. The first thing we will see will be the Login part, whereby already having a user you can enter, but first let's create a new account, selecting "Crear usuario".



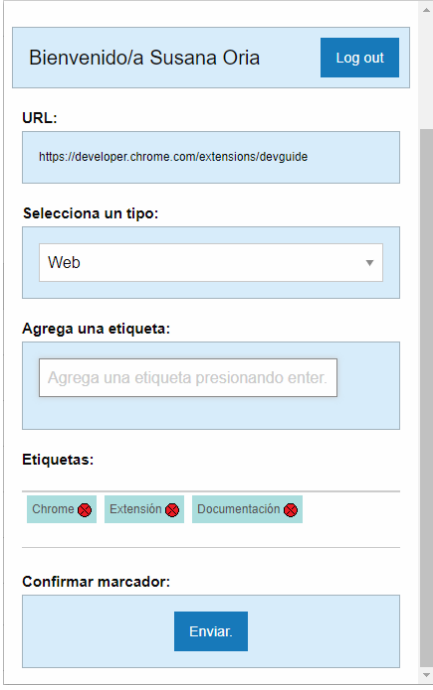
The image shows a login form with a light blue background. At the top, it says "Ingreso." followed by the instruction "Ingresa tu usuario y email." Below this, there are two input fields: "Usuario" with the placeholder "Ingresa tu usuario" and "Email" with the placeholder "Ingresa tu email". At the bottom, there are two buttons: "Login" and "Crear Usuario". A circular loading indicator is visible at the bottom center of the form.

2. Here you will simply choose username and an email, this to be able to identify yourself among the users when sending your bookmarks.



The image shows a registration form with a light blue background. At the top, it says "Registro." followed by the instruction "Ingresa tu usuario y email." Below this, there are two input fields: "Usuario" with the placeholder "Ingresa tu usuario" and "Email" with the placeholder "Ingresa tu email". At the bottom, there are two buttons: "Agregar" and "Regresar".

3. Once the data is entered, just press add and you will be returned to the login section.
4. On the next screen is where we can save the bookmarks, we will have the URL of the current website where you are, an option of type of bookmark (whether you are creating a bookmark of a Web, Mobile or Desktop application) and A section for labels, these with functionality like a hashtag, to be able to classify the markers more easily. Once you have selected a type (in the case of this test it would be "Mobile") and the labels you like (3 at least, please) you can press the submit button to save your bookmark.



The screenshot shows a web form for creating a bookmark. At the top, it displays a welcome message: "Bienvenido/a Susana Oria" next to a blue "Log out" button. Below this, the "URL:" field contains "https://developer.chrome.com/extensions/devguide". The "Selecciona un tipo:" dropdown menu is set to "Web". The "Agrega una etiqueta:" section has a text input field with the placeholder "Agrega una etiqueta presionando enter.". Under "Etiquetas:", there are three tags: "Chrome", "Extensión", and "Documentación", each with a red 'x' icon. At the bottom, the "Confirmar marcador:" section features a blue "Enviar." button.

5. When you finish you can press the Logout button if another user wants to use the extension.

Ready, that should be it for the extension, now if it is not too much trouble you will be asked to answer the following survey as honestly as possible:

https://docs.google.com/forms/d/e/1FAIpQLSeAN2r7jXSQSr08wKp9CALIUzhG6AbqeRKzu8qbA8g5b-wTBw/viewform?usp=sf_link

References

- [1] X. Larrucea, R. V. O'Connor, R. Colomo-Palacios, and C. Y. Laporte, "Software process improvement in very small organizations," *IEEE Software*, vol. 33, pp. 85–89, mar 2016.
- [2] M. Levy and O. Hazzan, "Knowledge management in practice: The case of agile software development," in *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering, CHASE 2009*, pp. 60–65, 2009.
- [3] K. Schneider, *Experience and knowledge management in software engineering*. Springer Berlin Heidelberg, 2009.
- [4] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, and A. Marcus, "How Can I Use This Method?," in *International Conference on Software Engineering*, pp. 880–890, 2015.
- [5] M. Bhat, K. Shumaiev, K. Koch, U. Hohenstein, A. Biesdorf, and F. Matthes, "An Expert Recommendation System for Design Decision Making: Who Should be Involved in Making a Design Decision?," in *Proceedings - 2018 IEEE 15th International Conference on Software Architecture, ICSA 2018*, pp. 85–94, Institute of Electrical and Electronics Engineers Inc., jul 2018.
- [6] D. Viana, J. Rabelo, T. Conte, A. Vieira, E. Barroso, and M. Dib, "A qualitative study about the life cycle of lessons learned," in *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2013 - Proceedings*, pp. 73–76, 2013.
- [7] S. Dorairaj, J. Noble, and P. Malik, "Knowledge management in distributed agile software development," in *2012 Agile Conference*, IEEE, aug 2012.
- [8] S. Sonnentag, C. Niessen, and &. Ludith Volmer, "Expertise in Software Design," in *Cambridge handbook of expertise and expert performance*, ch. Expertise, pp. 373–387, Cambridge, 2006.
- [9] K. A. Ericsson, M. J. Prietula, and E. T. Cokely, "The Making of an Expert," *Harvard business review*, pp. 115–121, 2007.
- [10] V. Clerc, P. Lago, and H. Van Vliet, "Architectural knowledge management practices in agile global software development," in *Proceedings - 2011 6th IEEE International Conference on Global Software Engineering Workshops, ICGSE Workshops 2011*, pp. 1–8, 2011.
- [11] A. Jedlitschka, M. Ciolkowski, C. Denger, B. Freimut, and A. Schlichting, "Relevant Information Sources for Successful Technology Transfer: A Survey Using Inspections as an Example," in *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pp. 31–40, IEEE, sep 2007.
- [12] K. Y. Sharif and J. Buckley, "Observation of Open Source programmers' information seeking," in *2009 IEEE 17th International Conference on Program Comprehension*, pp. 307–308, IEEE, may 2009.

- [13] C. R. Rupakheti and D. Hou, “Satisfying Programmers’ Information Needs in API-Based Programming,” in *2011 IEEE 19th International Conference on Program Comprehension*, pp. 250–253, IEEE, jun 2011.
- [14] P. Mohagheghi and R. Conradi, “Quality, productivity and economic benefits of software reuse: A review of industrial studies,” *Empirical Software Engineering*, vol. 12, pp. 471–516, oct 2007.
- [15] S. Nerur and V. G. Balijepally, “Theoretical reflections on agile development methodologies,” *Communications of the ACM*, vol. 50, no. 3, pp. 79–83, 2007.
- [16] G. Borrego, A. L. Morán, R. R. Palacio, A. Vizcaíno, and F. O. García, “Towards a reduction in architectural knowledge vaporization during agile global software development,” *Information and Software Technology*, apr 2019.
- [17] H. Holz, G. Melnik, and M. Schaaf, “Knowledge management for distributed agile processes: models, techniques, and infrastructure,” in *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.*, pp. 291–294, IEEE Comput. Soc, 2003.
- [18] N. Uikey, U. Suman, and A. K. Ramani, “A Documented Approach in Agile Software Development,” Tech. Rep. 2, 2011.
- [19] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza, “Mining stackoverflow to turn the IDE into a self-confident programming Prompter,” in *11th Working Conference on Mining Software Repositories, MSR 2014 - Proceedings*, (New York, New York, USA), pp. 102–111, Association for Computing Machinery, Inc, may 2014.
- [20] C. McMillan, D. Poshyvanyk, M. Grechanik, Q. Xie, and C. Fu, “Portfolio: Searching for relevant functions and their usages in millions of lines of code,” *ACM Transactions on Software Engineering and Methodology*, vol. 22, pp. 1–30, oct 2013.
- [21] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer, “Example-centric programming,” in *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*, (New York, New York, USA), p. 513, ACM Press, 2010.
- [22] G. Borrego, G. Salazar-Lugo, M. Parra, and R. Palacio, “Slack’s knowledge classification mechanism for architectural knowledge condensation,” in *International Conference on Computational Science and Computational Intelligence*, (Las Vegas, Nevada, USA), pp. 1121–1126, 2019.
- [23] B. Bonilla-Morales, S. Crespo, and C. Clunie, “Reuse of Use Cases Diagrams: An Approach based on Ontologies and Semantic Web Technologies,” jul 2012.
- [24] D. Matter, A. Kuhn, and O. Nierstrasz, “Assigning bug reports using a vocabulary-based expertise model of developers,” in *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories, MSR 2009*, pp. 131–140, 2009.
- [25] H. Kagdi, M. Hammad, and J. I. Maletic, “Who can help me with this source code change?,” in *IEEE International Conference on Software Maintenance, ICSM*, pp. 157–166, 2008.
- [26] S. Minto and G. C. Murphy, “Recommending emergent teams,” in *Proceedings - ICSE 2007 Workshops: Fourth International Workshop on Mining Software Repositories, MSR 2007*, 2007.

- [27] R. S. Pressman, *Software engineering: a practitioner's approach*. Palgrave macmillan, 2005.
- [28] K. Schwaber, "Agile project management with scrum, microsoft," *Press, Redmond, WA, USA*, 2004.
- [29] S. Ammar-Khodja and A. Bernard, "An overview on knowledge management," in *Methods and Tools for Effective Knowledge Life-Cycle-Management*, pp. 3–21, Springer Berlin Heidelberg.
- [30] W. Frakes and K. Kang, "Software reuse research: status and future," *IEEE Transactions on Software Engineering*, vol. 31, pp. 529–536, jul 2005.
- [31] M. Dabhade, S. Suryawanshi, and R. Manjula, "A systematic review of software reuse using domain engineering paradigms," in *2016 Online International Conference on Green Engineering and Technologies (IC-GET)*, IEEE, nov 2016.
- [32] W. Spoelstra, M. Iacob, and M. van Sinderen, "Software reuse in agile development organizations," in *Proceedings of the 2011 ACM Symposium on Applied Computing - SAC '11*, ACM Press, 2011.
- [33] P. Kruchten, P. Lago, and H. van Vliet, "Building up and reasoning about architectural knowledge," in *Quality of Software Architectures*, pp. 43–58, Springer Berlin Heidelberg, 2006.
- [34] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Addison-Wesley Professional, 2003.
- [35] J. Bosch, "Software architecture: The next step," in *Software Architecture*, pp. 194–199, Springer Berlin Heidelberg, 2004.
- [36] P. Clements, D. Garlan, R. Little, R. Nord, and J. Stafford, "Documenting software architectures: views and beyond," in *25th International Conference on Software Engineering, 2003. Proceedings.*, IEEE, 2003.
- [37] A. Cockburn, *Agile software development: the cooperative game*. Pearson Education, 2006.
- [38] R. Noordeloos, C. Manteli, and H. V. Vliet, "From RUP to scrum in global software development: A case study," in *2012 IEEE Seventh International Conference on Global Software Engineering*, IEEE, aug 2012.
- [39] J. S. Edwards, "Managing software engineers and their knowledge," in *Managing Software Engineering Knowledge*, pp. 5–27, Springer Berlin Heidelberg, 2003.
- [40] D. W. McDonald and M. S. Ackerman, "Just talk to me," in *Proceedings of the 1998 ACM conference on Computer supported cooperative work - CSCW '98*, ACM Press, 1998.
- [41] R. Milewicz and E. Raybourn, "Talk to me: A case study on coordinating expertise in large-scale scientific software projects," in *2018 IEEE 14th International Conference on e-Science (e-Science)*, IEEE, oct 2018.
- [42] N. bin Ali, "Is effectiveness sufficient to choose an intervention?," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '16*, ACM Press, 2016.

- [43] J. Josyula, S. Panamgipalli, M. Usman, R. Britto, and N. B. Ali, "Software practitioners' information needs and sources: A survey study," in *2018 9th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, pp. 1–6, IEEE, 2018.
- [44] A. Mockus and J. Herbsleb, "Expertise browser: a quantitative approach to identifying expertise," in *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, ACM.
- [45] K. Ehrlich and N. S. Shami, "Searching for expertise," in *Proceedings of the SIGCHI Conference on Human factors in Computing Systems*, pp. 1093–1096, 2008.
- [46] Y. Andriyani, R. Hoda, and R. Amor, "Understanding knowledge management in agile software development practice," in *International Conference on Knowledge Science, Engineering and Management*, pp. 195–207, Springer, 2017.
- [47] C. P. Maciel, É. F. de Souza, R. de Almeida Falbo, K. R. Felizardo, and N. L. Vijaykumar, "Knowledge management diagnostics in software development organizations: a systematic literature review," in *Proceedings of the 17th Brazilian Symposium on Software Quality*, pp. 141–150, 2018.
- [48] I. Becerra-Fernandez and R. Sabherwal, *Knowledge management: Systems and processes*. Routledge, 2014.
- [49] K. Dalkir, *Knowledge Management in Theory and Practice*. Routledge, sep 2013.
- [50] M. A. Razzak and R. Ahmed, "Knowledge sharing in distributed agile projects: Techniques, strategies and challenges," in *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, IEEE, sep 2014.
- [51] G. Borrego, A. L. Moran, R. Palacio, and O. M. Rodriguez, "Understanding architectural knowledge sharing in AGSD teams: An empirical study," in *2016 IEEE 11th International Conference on Global Software Engineering (ICGSE)*, IEEE, aug 2016.
- [52] I. Nonaka, "The knowledge-creating company," in *The Economic Impact of Knowledge*, pp. 175–187, Elsevier, 1998.
- [53] T. Clear, "Documentation and agile methods," *ACM SIGCSE Bulletin*, vol. 35, pp. 12–13, jun 2003.
- [54] S. L. Hoe, "Tacit knowledge, nonaka and takeuchi seci model and informal knowledge processes," *International Journal of Organization Theory & Behavior*, 2006.
- [55] A. B. Marques, J. R. Carvalho, R. Rodrigues, T. Conte, R. Prikładnicki, and S. Marczak, "An ontology for task allocation to teams in distributed software development," in *2013 IEEE 8th International Conference on Global Software Engineering*, IEEE, aug 2013.
- [56] J. I. Olszewska and I. K. Allison, "ODYSSEY: Software development life cycle ontology," in *Proceedings of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, SCITEPRESS - Science and Technology Publications, 2018.
- [57] M. A. Paredes-Valverde, M. del Pilar Salas-Zárate, R. Colomo-Palacios, J. M. Gómez-Berbís, and R. Valencia-García, "An ontology-based approach with which to assign human resources to software projects," *Science of Computer Programming*, vol. 156, pp. 90–103, may 2018.

- [58] P. Wongthongtham, E. Chang, T. Dillon, and I. Sommerville, "Development of a software engineering ontology for multisite software development," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, pp. 1205–1217, aug 2009.
- [59] R. Martinho, J. Varajao, and D. Domingos, "Using the semantic web to define a language for modelling controlled flexibility in software processes," *IET software*, vol. 4, no. 6, pp. 396–406, 2010.
- [60] M. Adnan and M. Afzal, "Ontology based multiagent effort estimation system for scrum agile method," *IEEE Access*, vol. 5, pp. 25993–26005, 2017.
- [61] K. Hamdan, H. Khatib, J. Moses, and P. Smith, "A software cost ontology system for assisting estimation of software project effort for use with case-based reasoning," in *2006 Innovations in Information Technology*, IEEE, nov 2006.
- [62] R. G. C. Rocha, A. Araujo, D. Cordeiro, R. R. Azevedo, and D. da Silva, "DKDOnto: An ontology to support software development with distributed teams," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, IEEE, aug 2018.
- [63] A. Vizcaino, F. Garcia, M. Piattini, and S. Beecham, "A validated ontology for global software development," *Computer Standards & Interfaces*, vol. 46, pp. 66–78, may 2016.
- [64] M. Bhatia, A. Kumar, and R. Beniwal, "Ontology based framework for automatic software's documentation," in *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 421–424, IEEE, 2015.
- [65] A. Khatoon, Y. H. Motla, M. Azeem, H. Naz, and S. Nazir, "Requirement change management for global software development using ontology," in *2013 IEEE 9th International Conference on Emerging Technologies (ICET)*, IEEE, dec 2013.
- [66] A. Vizcaino, F. Garcia, I. Caballero, J. Villar, and M. Piattini, "Towards an ontology for global software development," *IET Software*, vol. 6, no. 3, p. 214, 2012.
- [67] M. Bhatia, A. Kumar, and R. Beniwal, "Ontology based framework for detecting ambiguities in software requirements specification," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 3572–3575, IEEE, 2016.
- [68] S. Sitthithanasakul and N. Choosri, "Using ontology to enhance requirement engineering in agile software process," in *2016 10th International Conference on Software, Knowledge, Information Management & Applications (SKIMA)*, IEEE, 2016.
- [69] V. S. Fonseca, M. P. Barcellos, and R. de Almeida Falbo, "An ontology-based approach for integrating tools supporting the software measurement process," *Science of Computer Programming*, vol. 135, pp. 20–44, feb 2017.
- [70] M. P. S. Bhatia, A. Kumar, R. Beniwal, and T. Malik, "Ontology driven software development for automatic detection and updation of software requirement specifications," *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 23, pp. 197–208, jan 2020.
- [71] A. A. AlSanad, A. Chikh, and A. A., "A domain ontology for software requirements change management in global software development environment," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 3, 2019.

- [72] T. Hovorushchenko and O. Pavlova, "Evaluating the software requirements specifications using ontology-based intelligent agent," in *2018 IEEE 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT)*, IEEE, sep 2018.
- [73] M. S. Murtazina and T. V. Avdeenko, "Ontology-based approach to the requirements engineering in agile environment," in *2018 XIV International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE)*, IEEE, oct 2018.
- [74] J. R. Martínez-García, F.-E. Castillo-Barrera, R. R. Palacio, G. Borrego, and J. C. Cuevas-Tello, "Ontology for knowledge condensation to support expertise location in the code phase during software development process," *IET Software*, vol. 14, pp. 234–241, jun 2020.
- [75] Y. Zhang, R. Witte, J. Rilling, and V. Haarslev, "Ontological approach for the semantic recovery of traceability links between software artefacts," *IET Software*, vol. 2, no. 3, p. 185, 2008.
- [76] F. Ruiz, A. Vizcaíno, M. Piattini, and F. García, "An ontology for the management of software maintenance projects," *International Journal of Software Engineering and Knowledge Engineering*, vol. 14, no. 03, pp. 323–349, 2004.
- [77] A. Gomez-Perez, M. Fernandez-Lopez, and O. Corcho, "Theoretical Foundations of Ontologies," in *Ontological Engineering*, pp. 1–45, Springer-Verlag, apr 2006.
- [78] A. Gomez-Perez, "Ontology Evaluation," in *Handbook on Ontologies*, pp. 251–273, Springer Berlin Heidelberg, 2004.
- [79] N. F. Noy, D. L. McGuinness, *et al.*, "Ontology development 101: A guide to creating your first ontology," 2001.
- [80] S. Sitthithanasakul and N. Choosri, "Application of software requirement engineering for ontology construction," in *2017 International Conference on Digital Arts, Media and Technology (ICDAMT)*, IEEE, 2017.
- [81] E. S. M. and A. S. A., "Ontology for knowledge management in software maintenance," *International Journal of Information Management*, vol. 34, pp. 704–710, oct 2014.
- [82] D. Schober, B. Smith, S. E. Lewis, W. Kusnierczyk, J. Lomax, C. Mungall, C. F. Taylor, P. Rocca-Serra, and S.-A. Sansone, "Survey-based naming conventions for use in OBO foundry ontology development," *BMC Bioinformatics*, vol. 10, no. 1, p. 125, 2009.
- [83] D. Schober, I. Tudose, V. Svatek, and M. Boeker, "OntoCheck: verifying ontology naming conventions and metadata completeness in protégé 4," *Journal of Biomedical Semantics*, vol. 3, no. Suppl 2, p. S4, 2012.
- [84] R. de Almeida Falbo, C. S. de Menezes, and A. R. C. da Rocha, "A systematic approach for building ontologies," in *Lecture Notes in Computer Science*, pp. 349–360, Springer Berlin Heidelberg, 1998.
- [85] M. Fernandez, A. Gomez-Perez, and N. Juristo, "Methontology: from ontological art towards ontological engineering," in *Proceedings of the AAAI97 spring symposium series on ontological engineering*, pp. 33–40, Stanford, USA, 1997.
- [86] J. Park, K. Sung, and S. Moon, "Developing graduation screen ontology based on the METHONTOLOGY approach," in *2008 Fourth International Conference on Networked Computing and Advanced Information Management*, IEEE, sep 2008.

- [87] O. Corcho, M. Fernández-López, A. Gómez-Pérez, and A. López-Cima, “Building legal ontologies with METHONTOLOGY and WebODE,” in *Law and the Semantic Web*, pp. 142–157, Springer Berlin Heidelberg, 2005.
- [88] A. F. Sawsaa and J. Lu, “Building information science ontology (OIS) with methontology and protégé,” *Journal of Internet Technology and Secured Transaction*, vol. 1, pp. 100–109, dec 2012.
- [89] M. C. Suárez-Figueroa, A. Gómez-Pérez, and M. Fernández-López, “The NeOn methodology framework: A scenario-based methodology for ontology development,” *Applied Ontology*, vol. 10, pp. 107–145, sep 2015.
- [90] M. C. Suárez-Figueroa, A. Gómez-Pérez, and M. Fernández-López, “The NeOn methodology for ontology engineering,” in *Ontology Engineering in a Networked World*, pp. 9–34, Springer Berlin Heidelberg, dec 2011.
- [91] H. S. Pinto, S. Staab, C. Tempich, and Y. Sure, “Distributed engineering of ontologies (DILIGENT),” in *Semantic Web and Peer-to-Peer*, pp. 303–322, Springer-Verlag.
- [92] M. Uschold and M. King, *Towards a methodology for building ontologies*. Citeseer, 1995.
- [93] S. John, N. Shah, and C. Stewart, “Towards a software centric approach for ontology development: Novel methodology and its application,” in *2018 IEEE 15th International Conference on e-Business Engineering (ICEBE)*, pp. 139–146, IEEE, 2018.
- [94] P. Spyns, Y. Tang, and R. Meersman, “An ontology engineering methodology for dogma,” *Applied Ontology*, vol. 3, no. 1-2, pp. 13–39, 2008.
- [95] J. B. Machado, S. Isotani, A. Barbosa, J. Bandeira, W. Alcantara, I. Bittencourt, and E. F. Barbosa, “Ontosoft process: Towards an agile process for ontology-based software,” in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, pp. 5813–5822, IEEE, 2016.
- [96] S. Peroni, “SAMOD: an agile methodology for the development of ontologies,” in *Proceedings of the 13th OWL: Experiences and Directions Workshop and 5th OWL reasoner evaluation workshop (OWLED-ORE 2016)*, pp. 1–14, 2016.
- [97] M. Kingsun, T. Myers, and D. Hardy, “C-dom: a structured co-design framework methodology for ontology design and development,” in *Proceedings of the Australasian Computer Science Week Multiconference*, pp. 1–10, 2018.
- [98] C. Tautz and C. G. von Wangenheim, *REFSENO: A representation formalism for software engineering ontologies*. Fraunhofer-IESE, 1998.
- [99] A. Strauss and J. M. Corbin, *Grounded theory in practice*. Sage, 1997.
- [100] D. Viana, T. Conte, and C. R. de Souza, “Knowledge transfer between senior and novice software engineers: A qualitative analysis,” in *SEKE*, pp. 235–240, 2014.
- [101] Z. Li, P. Liang, and P. Avgeriou, “Application of knowledge-based approaches in software architecture: A systematic mapping study,” *Information and Software Technology*, vol. 55, pp. 777–794, may 2013.
- [102] T. D. LaToza, G. Venolia, and R. DeLine, “Maintaining mental models,” in *Proceeding of the 28th international conference on Software engineering - ICSE '06*, ACM Press, 2006.

- [103] I. Sommerville, *Software Engineering, 10th Edition* | Pearson. Pearson, 2016.
- [104] M. C. Suárez-Figueroa, A. Gómez-Pérez, and B. Villazón-Terrazas, “How to Write and Use the Ontology Requirements Specification Document,” in *Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on On the Move to Meaningful Internet Systems: Part II*, pp. 966–982, Springer-Verlag, 2009.
- [105] P. Bourque, R. E. Fairley, et al., *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press, 2014.
- [106] M. C. Suárez-Figueroa, A. Gómez-Pérez, and B. Villazón-Terrazas, “How to write and use the ontology requirements specification document,” in *On the Move to Meaningful Internet Systems: OTM 2009*, pp. 966–982, Springer Berlin Heidelberg, 2009.
- [107] I. Grangel-González, L. Halilaj, G. Coskun, and S. Auer, “Towards vocabulary development by convention,” in *Proceedings of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, SCITEPRESS - Science and Technology Publications, 2015.
- [108] C. Bezerra, F. Freitas, and F. Santana, “Evaluating ontologies with competency questions,” in *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, IEEE, nov 2013.
- [109] N. Marangunić and A. Granić, “Technology acceptance model: a literature review from 1986 to 2013,” *Universal access in the information society*, vol. 14, no. 1, pp. 81–95, 2015.