*mathematics*

MDPI

*Article*

# Tags' Recommender to Classify Architectural Knowledge Applying Language Models

**Gilberto Borrego [1,†], Samuel González-López [2,†] and Ramón R. Palacio [3,*,†]**

[1] Departamento de Computación y Diseño, Instituto Tecnológico de Sonora, Ciudad Obregón 85000, Mexico; gilberto.borrego@itson.edu.mx
[2] Department of Information Technologies, Universidad Tecnológica de Nogales, Nogales 84097, Mexico; sgonzalez@utnogales.edu.mx
[3] Unidad Navojoa, Instituto Tecnológico de Sonora, Navojoa 85860, Mexico
* Correspondence: ramon.palacio@itson.edu.mx
† These authors contributed equally to this work.

**Abstract:** Agile global software engineering challenges architectural knowledge (AK) management since face-to-face interactions are preferred over comprehensive documentation, which causes AK loss over time. The AK condensation concept was proposed to reduce AK losing, using the AK shared through unstructured electronic media. A crucial part of this concept is a classification mechanism to ease AK recovery in the future. We developed a Slack complement as a classification mechanism based on social tagging, which recommends tags according to a chat/message topic, using natural language processing (NLP) techniques. We evaluated two tagging modes: NLP-assisted versus alphabetical auto-completion, in terms of correctness and time to select a tag. Fifty-two participants used the complement emulating an agile and global scenario and gave us their complement's perceptions about usefulness, ease of use, and work integration. Messages tagged through NLP recommendations showed fewer semantic errors, and participants spent less time selecting a tag. They perceived the component as very usable, useful, and easy to be integrated into the daily work. These results indicated that a tag recommendation system is necessary to classify the shared AK accurately and quickly. We will improve the NLP techniques to evaluate AK condensation in a long-term test as future work.

**Keywords:** agile global software engineering; architectural knowledge management; natural language processing; knowledge condensing



**Citation:** Borrego, G.; González-López, S.; Palacio, R.R. Tags Recommender to Classify Architectural Knowledge Applying Language Models. *Mathematics* **2022**, *10*, 446. https://doi.org/10.3390/math10030446

Academic Editors: Grigoreta-Sofia Cojocar and Adriana-Mihaela Guran

Received: 22 November 2021
Accepted: 24 December 2021
Published: 30 January 2022



## 1. Introduction

Architectural knowledge (AK) is a core part of any software project [1], and it is commonly documented based upon standards [2]. This documentation eases the knowledge management cycle [3] (create/capture, share/disseminate, acquire/apply), even when companies practice global software engineering (GSE). In GSE, transferring/sharing documents among the distributed facilities is commonly done to decrease the effect of the four distances in this paradigm [4,5] (physical, temporal, linguistic, and cultural).

Nowadays, GSE companies are adopting agile software development, which caused the arising of the agile GSE approach; in fact, VersionOne (https://explore.digital.ai/state-of-agile/14th-annual-state-of-agile-report, accessed on 15 March 2021) reports in 2020 that 71% of surveyed agile enterprises practice agile GSE. This trend causes a paradigms contradiction referring to knowledge managing. The agile paradigm states that functional software and face-to-face communication are preferred over processes, and comprehensive documentation [6]. In practice, most agile projects' documentation could be replaced by enhancing informal communication since the agile paradigm proposes a stronger emphasis on tacit knowledge rather than explicit knowledge [7]. This emphasis on tacit knowledge has to be done without disregarding formal documentation [8]; however, in the agile GSE

practice, this implies a reduction of AK documentation or even its disappearance [8–10], known as documentation debt [11]. This situation affects the AK management since most of the projects' AK remains tacit [12], hindering the knowledge management cycle in agile GSE environments because traditional knowledge managing on GSE is based on explicit knowledge sharing.

The prevalence of AK documentation debt and the affectation on the knowledge management cycle eventually cause AK vaporization [13]. It could trigger some of the following problems in agile GSE projects [14–16]: poorly understood technical solutions, reduced knowledge transfer between teams, low quality, defects in software evolution and maintenance, difficulty in conducting tests, stress with stakeholders, and time wasted by experts answering questions and attempting to find solutions to problems previously solved. It is worth highlighting that constant questions could cause interpersonal relationship erosion, which could affect the knowledge flow [17], and trust relationship building, which is essential for agile teams.

In the aim to address the above problems, in a previous study [18] we presented the knowledge condensation concept, which takes advantage of the agile GSE developers' preference to share AK by unstructured textual and electronic media (UTEM), e.g., instant messengers, emails, etc., [10,19]. The knowledge condensation concept includes a knowledge classification mechanism, which was initially based on social tagging to tag relevant AK during their interactions. These tags have to be registered by the development teams to relate to the teams' jargon. Furthermore, these tags must be related to a set of meta-tags, which are semantic anchors to ease the AK later retrieval in case the exact name of a tag was forgotten. This semi-fixed tagging mechanism could avoid the free and unassisted tagging problems. These problems are tag explosion, interpretation differences of a tag's meaning, incomplete context to understand a tag; tags that only make sense when used together (known as composite tags), and tags with the same meaning but written differently (known as obscure similarity) [20]. These problems could lead to tagging difficulties which could ruin the AK classification mechanism, and consequently, they would cause information retrieval problems [21]. To preserve the classification mechanism, in the same study [18], we presented a tagging helper implemented for Skype, which auto-completes tags in an alphabetic way during developers' UTEM interactions, helping them to select an existing tag and to type it correctly. Both AK classification mechanism and tagging service were evaluated, and we obtained broad acceptance by the participants. However, they suggested including a more intelligent way to help the tagging action. We considered it essential to increase the knowledge condensation concept adoption in a real agile GSE environment.

Therefore, the tagging service must minimize the negative cost that interruption could cause [22,23], since developers must redirect their attention from the main activity to properly tag the current interaction and then return naturally to its primary activity. Thus, a tagging service is required to ease the knowledge flow and not become a constant source of disruption [24,25].

Based on the stated above, we developed an AK classification mechanism based on semi-fixed tagging, which was implemented as a Slack (https://slack.com/, accessed on 20 October 2021) service, including a tags recommendation module based on natural language processing (NLP) and statistical language models techniques.

This paper aims to determine with which tagging method, either assisted by auto-completion or by NLP, users select tags with a better coherence (named in this paper as semantic correctness) regarding the context of the messages. A coherent tagging is essential to the AK condensation concept since it is the base to find the required AK in the future. Another objective in this paper is to determine with which tagging method a developer could select in a faster way a given tag. The time to tag a message is also an essential factor to consider since interruptions could cause negative consequences in the developer work (as we stated before), as well as the time to accomplish a task is relevant to enhance the usability of any software product [26]. Thus, a quick tag selection could increase the adoption possibility of our proposed classification mechanism, as well as the knowledge

condensation concept per se. Thus, the research questions of this paper are the following: (RQ1) With which tagging method (assisted by auto-completion or assisted by NLP) does the tag selection by users fit better to the message context?; (RQ2) With which tagging method (assisted by auto-completion or assisted by NLP) the tag selection by users is faster? and (RQ3) Which is the perception of usefulness, ease of use, and integration into the work of the tagging service implemented in Slack?

The main contributions of this paper are the following: (1) the implementation of a tagging service based on NLP to help to condense AK; (2) evidence about an NLP tagging service cause that users tag faster and more coherently than using tag auto-completion; (3) encouraging evaluation results of the implemented tagging service in terms of usefulness, ease of use and integration in software developer activities; (4) relevant results about the pertinence of the recommended tags by the NLP mechanism; and (5) a corpus of 291 tagged messages contained in seven categories.

The remainder of this paper is organized as follows: Section 2 presents the work related to AK extraction/classification and tagging recommendation in software engineering; in Section 3, we present the Slack service used to implement the AK classification mechanism; Section 4 presents a description of the NLP models used to obtain the tags recommendation; Section 5 presents the method used to evaluate the service; in Section 6 and Section 7, we, respectively, present the results of the evaluation and its threats of validity. Finally, in Section 8 we discuss the obtained results and present our conclusions in Section 9.

## 2. Related Work

In this section, we present the related work about AK, where we clarify the abstractions levels used in this concept and how AK management is a challenge in agile GSE nowadays. Furthermore, we present the approaches to address the lack of explicit AK in agile GSE, based on using repositories, tagging, Q&A sites, to leverage the work done by software developers to provide a low-intrusive means of capturing AK during day-to-day work.

### *2.1. Architectural Knowledge and Agile Global Software Engineering*

According to Kruchten et al. [27] AK is composed of architecture design and the design decisions and rationale used to attain architectural solutions. This definition could be clear for everyone; however, there are many definitions of the concept of software architecture; thus, the term architecture design is not as straightforward as possible. The definitions of software architecture from Bass et al. [28], from the ISO/IEC/IEEE 42010:2011 standard [29], from Kruchten [30] and many others, could build up the idea that software architecture is a very abstract and general definition of a structure on which software will be developed. As architecture implies abstraction, it suppresses purely local information; thus, private details of components are not architectural [31].

Deeping in the software architecture concepts, there also exists the architectural views definition: a representation of a coherent set of architectural elements as written by and read by system stakeholders [28]. Thus, depending on the stakeholders, the level of design detail could vary. The typical architectural views include the functional view (called the logical view), the concurrency view (called the process or thread view), the code view, the development view (called the implementation view), and the physical view [31]. The means above that detailed views on code and development could be considered architectural topics. Further, if the concern's scope is limited to a subsystem within a system, what is considered architectural is different from what the software architect considers architectural [31]. Therefore, the context influences what is architectural, leading to the concepts of macro-architecture and micro-architecture. Macro-architecture covers the architecture levels to which architecturally relevant elements are assigned (large-scale architecture); thus, it covers aspects such as requirements, decisions, and structures at a high level of abstraction: decisions concerning necessary system interfaces. Micro-architecture covers aspects with a lower level of abstraction, i.e., detailed design or "small-

scale" architecture, which is closely associated with the source code with no fundamental influence on an architecture [32].

Then, the concept of AK (design decisions and rationale used to attain an architectural design) takes another dimension. Software architecture could cover either high abstract and general topics (macro-architecture) or detailed design at low abstraction levels (micro-architecture) regarding the architectural concern's scope.

On the other hand, AK management is challenging in agile GSE since knowledge management is a challenge too [33–35]. Knowledge capturing is a critical phase of the AK management process in agile GSE environment [36]; however, the agile developers' attitudes [37] towards this phase cause documentation debt [11]. Since an agile GSE environment leads to a lack of captured AK, the knowledge management phases of sharing/disseminating and acquiring/applying are also affected because AK is shared and acquired based on inappropriate documentation or even tacit knowledge. Souza et al. [38] determined that current architectural derivation methods are heavily based on the architects' tacit knowledge, and this does not offer sufficient support for inexperienced architects or novice developers. The limitations of the current derivations methods are the following: undetailed decision-making process, inconsistency between the artifacts (especially those referring to micro-architecture), and semantic loss, and lack of traceability between models (between/within macro-architecture and micro-architecture).

### *2.2. Approaches to Address the Lack of Explicit Architectural Knowledge*

One of the approaches to address the lack of explicit AK at the micro-architecture level is mining issue trackers, version control repositories (generally from open source software projects), or any other document which could contain AK. The main objective is to extract large sets of architectural design concepts [39], or automatically recover design decisions from projects' artifacts [40], which can be textual or based on unified modeling language [41]. However, none of these works consider that in agile environments exist documentation debt; thus, the mining sources used in these works could not exist o could be incomplete or outdated. Furthermore, mining repositories could be inefficient on small projects with a non-rigid discipline regarding the commits and bug fixing comments. Further, challenges associated with the volume, velocity, and variety of the data set and other challenges about extensive data systems were identified.

Another proposal to address the lack of explicit AK is using tags recommendation systems to classify work items during the development cycle. Tagging has been used in software engineering for information finding, team coordination, and cooperation, and helping to bridge socio-technical issues [42,43]. Automatic tag recommendation in software engineering started with the TAGREC method [44] to recommend tags for work items in IBM Jazz, based on the fuzzy set theory. More recently, neural networks had been used to recommend tags for question and answers (Q&A) sites [45], finding that this technique could infer the context of the posts and consequently it could recommend accurate tags for new posts. On the same line, Mushtaq et al. [46] proposed a framework to obtain explicit and implicit knowledge from community-based Q&A sites (e.g., StackOverflow) through mining techniques to support AK management; however, the authors did not report any implementation. In the Parra et al. work [47], a process to extract relevant keywords from video tutorials was proposed, which uses the video title, description, audio transcription, as well as external resources related to the video content, such as StackOverflow publications and its tags. This method showed that it could automatically extract tags that encompass up to 66% of the information developers consider relevant to outline the content of a software development video.

Another way of tagging is presented by Jovanovic et al. [48], called semantic tagging, which uses Wikipedia information to establish the tagging process semantically and to provide a methodical approach to apply tagging in software engineering content. In the same sense, TRG [49] proposes to discover and enrich tags focused on open source software through using a semantic graph to model the semantic correlations between the tags

and words of the software descriptions. A similar approach was used by Alqahtani and Rilling [50] who proposed an ontology-based software security tagger framework to automatically identify and classify cybersecurity-related entities and concepts in the text of software artifacts.

Most of the above-cited works use the traditional approaches in tags recommendation (EnTagRec, TagMulRec, and FastTagRec). They were compared against deep learning-based approaches (TagCNN, TagRNN, TagHAN, and TagRCNN) [51], obtaining that the performance of TagCNN and TagRCNN was more suitable in terms of performance than the traditional approaches. However, all these approaches do not consider the "cold start" problem [52], i.e., the absence of previous information about the target object.

The presented works evidence that Q&A sites, wikis, or any other similar platform can be considered as a source of AK, as was stated by Soliman et al. [53], who analyzed and classified StackOverflow posts, and these were contrasted with software engineers. They obtained that there is valuable AK particularly related to technological decisions. However, these discussions do not include specific project AK such as business rules, architectural decisions based on particular conditions of the project, etc. Therefore, organizations need to unify their AK sources, to have a similar resource as Q&A systems. CAKI [54] is a proposed solution to the necessity of internal AK, which works as a continuous integration of organization-internal and external AK sources, with enhanced semantic reasoning and personalized capabilities.

There are different approaches to enhance AK managing, focused on the extraction, classification, and searching of knowledge. Many efforts are focused on extracting/classifying AK from public Q&A services sites which concentrate enormous volumes of information, experiences, and knowledge related to technical aspects of software engineering, using different artificial intelligence techniques and data mining. However, AK is not stored in those Q&A services, such as discussions about project business rules, architectural decisions that depend on the project conditions, and even AK related to the project deployment. This AK is usually stored in the UTEM logs of the software company [9], and these information sources are our focus with the knowledge condensation concept. To help classify the unstructured AK in the UTEM logs, we chose assisted tagging to recommend adequate tags to project stakeholders. In the following section, we describe the Slack service developed to recommend tags using NLP models.

## 3. Architectural Knowledge Condensation and the Slack Tagging Service

The AK condensation concept was proposed in a previous work [18] as a response to reduce the AK vaporization in agile GSD, taking advantage of the preference of sharing AK through UTEM in these kinds of environments [9,10]. To instantiate this concept, we have to ensure the following items [18]:

- Accessible UTEM logs. All the stakeholders involved in an agile GSE project must access the information contained in the UTEM log files to consult the AK that is shared among the development team.
- UTEM log classification mechanism to structure the shared information to ease the AK retrieval. This mechanism must be based on a semantic scheme representing the AK shared through UTEM.
- AK searching mechanism. All the stakeholders could use the semantic scheme to find valuable AK with less effort in the structured UTEM logs.

In the aim to instantiate the AK condensation concept we chose Slack, an application that eases communication and collaboration within teams, which is booming in agile development, and researchers have even studied how developers use Slack [55–57]. Slack can be considered a UTEM since the messages shared between the interested parties have no fixed structure to ease the retrieval of AK. Given the popularity of Slack, we decided to develop a Slack tagging service so that the concept of knowledge condensation could eventually be evaluated in a real agile GSE environment. While Slack uses social tagging as part of its functionality, and there are some add-ons based on the use of tags, these present

the problems associated with free tagging: tag explosion, interpretation differences of a tag's meaning, incomplete context to understand a tag, etc. [20]. Furthermore, there are no assistants that implement intelligent elements to suggest tags.

Hereunder, we describe the operation of the service in terms of the elements of the concept of knowledge condensation [18], excepting the element named "Architectural knowledge searching mechanism", since it is out of the scope of this study (an example of the implementation of this mechanism can be seen in our past work [18]). In the last part of this section, we present a scenario to explain how the AK condensation concept could work in an agile GSE environment and the potential benefits of its implementation.

### *3.1. Accessible Messages of the UTEM Log Files*

We developed a component in Node.js, which extracts the messages periodically (or at request) using the Slack API and then stores the messages in a common repository (Algolia (https://www.algolia.com/, accessed on 29 October 2021). The main objective is to keep the Slack messages (previously classified by tags—see next section) and another UTEM's messages located in one single point so that the AK can easily be found.

### *3.2. UTEM Log Files Classification Mechanism*

In a previous work [18], we learned that tagging is feasible to classify the AK stored in the UTEM log files. With this focus, we proposed that developers tag the UTEM messages they consider relevant in terms of AK during their daily interactions to recover them in the future. To avoid tags explosion phenomena [20] (occurring when free tagging is allowed), we chose a semi-fixed tagging in a past work [18]. This type of tagging has a meta-tags base model (obtained from a previous study [9], see Figure 1), which represents the AK shared in UTEM in an agile GSE environment. Each meta-tag is related to user-tags which developers register in a web application at the beginning of each agile development cycle after they agree about which tags they will use during that cycle. The meta-tag model could be changed by another model that fits in other environments, such as the work of Martinez et al. [58] who formalize that model as an ontology for general use in software projects. The main objective of this model is to have a semantic anchor to recover AK in the future.

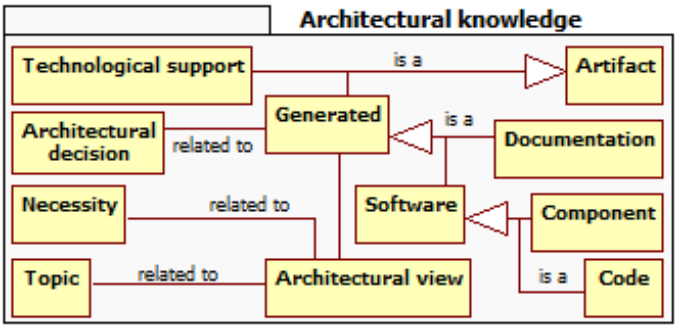


**Figure 1.** Meta-tags model, representing the architectural knowledge shared in agile GSE environments (adapted from [9]).

To ease the tags recalling during the interactions through Slack, we implemented a tagging system on which developers have to select the option "Tag! Tagger" from the options that appear for each message. Then, a dialogue box appears to allow the selection from a tags' list previously recorded, including meta-tags (see Figure 2A). This selection list includes an auto-completion feature that filters the available tags as the developer writes. It is also possible to enter customized (free) tags in a text field. In the final selection list, developers can view the tags recommendation list, which comes from the language models (see Figure 2B). On this list, the elements are the suggested tags. These elements are ordered regarding how semantically close the tag is to the message to be tagged. It is

worth remarking that the tagging system has three different ways to tag a message only for evaluation purposes.

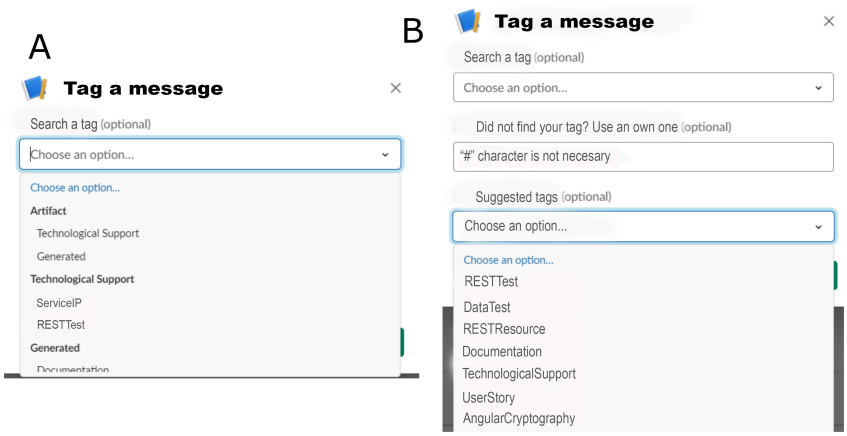


**Figure 2.** Graphic user interfaces of Slack tagging service. (**A**) Auto-completion assistance to select a tag; (**B**) tags recommendation list of the natural language processing (NLP) assistance comes from the language models. The middle textbox is used to type tags freely (only for evaluation purposes).

### *3.3. Scenario of Architectural Knowledge Condensation*

This part exemplifies how developers could use the Slack tagging service in an agile GSE scenario where the knowledge condensation concept is implemented.

First, developers (local and remote) during a kick-off meeting of a development cycle (e.g., Scrum sprint meeting) agree on the tags to use during the next cycle depending on the topics covered by the user stories to be developed. Next, developers register the agreed tags in the tag management Web application (see Figure 3B), and the conventional flow of an agile methodology continues.

In an agile environment, it is common that developers neglect AK documentation activities, as working software and individuals' interaction are preferred over comprehensive documentation, process, and tools [6]. In addition, there generally exists time pressures in agile GSE environments [36], and developers show a lousy attitude towards documentation [37]. These situations generate documentation debt [11], which is addressed by constantly interacting between remote and local team members via UTEM to acquire and share AK. During those interactions, developers can tag the AK they feel necessary to retrieve later (see Figure 3B), supported by the tagging service for Slack or another UTEM, such as email, Trello, Skype (see our past implementation in [18]). The tagged messages are stored in the respective UTEM logs, which the gathering service frequently reads and sends the messages to a repository (see Figure 3A).

Finally, due to the documentation debt, it is frequent that developers do not remember some architectural details (e.g., technical specifications, detailed design, application deployment issues, etc.) either during the same development cycle or in a later one [9]. It is even common that they do not remember the exact date or the communication media used to share AK with their counterparts. However, since team members tagged their interactions and these are located in a common repository, the project AK could be retrieved using the AK search engine presented in [18], which is out of the scope of this paper (see Figure 3C).

Although we describe this scenario sequentially, the technology elements involved (tagging assistants, tag management web application, gathering service, AK search engine, etc.) can come into play at any time. For example, tag registration and/or deletion could be performed at any time during the development cycle, as long as there is agreement among team members. Furthermore, although tagging is fundamental to this implementation of knowledge condensation, knowledge retrieval can be performed even before any message is tagged, either by retrieving by period, author, recipient, etc. However, not tagging conversations can lead to the loss of semantics and context of knowledge over time since

tags and the meta-tag structure to which they are related provide a semantic anchor that would offer a way to remember or intuit the context of the conversation being consulted.

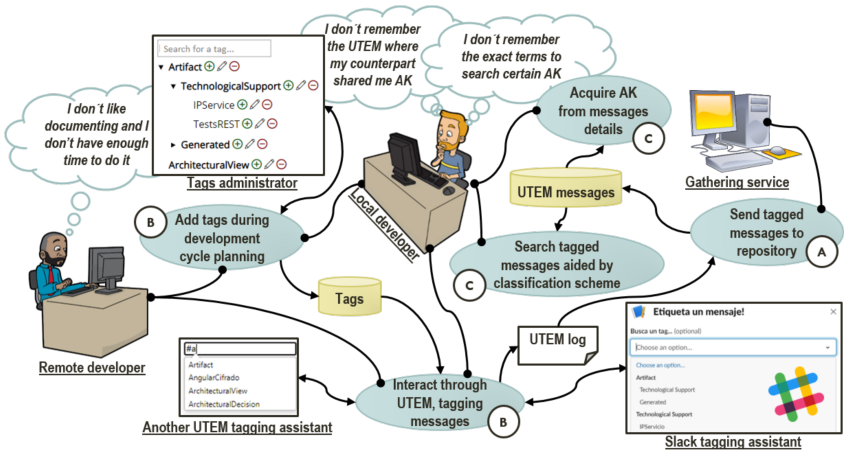


**Figure 3.** Rich picture of an AK condensation scenario based on [18]. It includes activities (ovals) corresponding to the three elements of the AK condensation concept, A = Accessible UTEM logs information, B = UTEM logs Classification mechanism, C = AK searching mechanism. Bulleted lines represent links between activities and who performs them. Arrowed lines represent links between activities and artifacts. There are three types of artifacts: resulting artifacts (activities' outgoing arrows), source artifacts (activities' in-going arrows), and interacting artifacts (linked with double arrow lines).

### *3.4. Potential Benefits of the Architectural Knowledge Condensation*

Listed below are the potential benefits of fully implementing the architectural knowledge condensation concept.

- Reduction of interruptions. Since full implementation of this concept provides a search engine for AK, a team member who has a question about any AK topic could first use the search engine before interrupting one of their or her teammates to ask a question.
- Reduced time to find AK. AK condensation offers a common point to search for shared knowledge in different UTEM, in addition to offering search filters that most UTEM do not offer, such as search by period, by author, by the recipient, by UTEM source, and by tag.
- Reduction of development tasks time. As a consequence of the two previous points, it is expected that the time to complete development tasks will be reduced since the project's AK will be more accessible through the AK search engine.

A crucial point to obtain the benefits listed above is that the AK classification mechanism is efficient and adapted to the dynamics of an agile environment to ensure the adoption of the technological elements that implement such mechanisms. That is why in this paper, we focus on evaluating how well an NLP tagging service suggests tags according to the context, intending to ease the interactions tagging. Furthermore, it is of our interest to evaluate how quickly the tagging action can be performed since the time to accomplish a tag is an essential factor to the future adoption of any software tool [26].

## 4. Language Models Development and Integration to Slack Tagging Service

As we mentioned before, we incorporated NLP techniques and statistical language models into the Slack tagging service. Language models use probabilities for their operation and specifically assign a value to a sequence of words [59]. An n-gram is a sequence of $n$ terms, which can be words or characters, e.g., "story user" is a sequence of two words and is called 2-gram or bigram. In this way, we can have different configurations of n-grams. N-gram models can compute the probability of a sequence of words (text segment) by

estimating each word's probability in an n-gram given the previous terms. Equation (1) computes the probabilities for any string of terms.

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})\ldots P(w_n|w_{1:n-1})$$
$$P(w_{1:n}) \approx \prod_{i=1}^{n} P(w_n|w_{1:n-1}) \quad (1)$$

In the aim of integrating the use language models to suggest tags during Slack interactions, we executed the following phases: (1) corpus development, (2) language models development, and (3) language models and Slack tagging service integration.

### *4.1. Corpus Development*

Language models require initial data (corpus) to be developed; thus, we took the UTEM logs generated in our previous study [18], in which participants interacted through Skype following a script. In the referenced study, participants tagged Skype messages following the script indications, and then, we determined which messages were tagged coherently. It is important to highlight that participants used the same set of tags used in this study, and those tags were related to the meta-tags model presented in the previous section (see Figure 1).

As the next step, we selected the coherently tagged messages to determine which tags were used more frequently. We obtained 291 messages tagged with the following tags (Table 1 the tags distribution): Encryption, Documentation, UserStory, RESTTest, RESTResource, TechnologicalSupport and TestData. It is important to point out that Documentation and TechnologicalSupport are meta-tags. Thus, we had the base to develop seven language models, one per each frequent tag with this data. We used this corpus to train the language models generated by each category (tags).

**Table 1.** Distribution of the 291 tagged messages selected from our previous study [18].

| Tag/Meta-Tag | Number of Messages |
|---|---|
| UserStory | 88 |
| RESTResource | 61 |
| RESTTest | 43 |
| Documentation | 36 |
| Encryption | 26 |
| TestData | 23 |
| TechnologicalSupport | 16 |

### *4.2. Language Models Development*

The developed method seeks to capture features of the seven tags selected from the original corpus. As the input of our language model, we used the lemma and the grammatical class. In the lemmatization process, the words' inflections are removed, and the words' root is obtained. For example, the word "computing" has the word "compute" as its lemma. On the other hand, the grammatical categories are the different classifications in which words are grouped, for example, nouns, articles, adjectives, verbs, pronouns, prepositions, conjunctions, and interjections. For example, the original comment from a developer in the #UserStory category "the resource you consult is the same as story 2" provides for the word story "story NN". The NN grammatical class represents a noun. The probability of #UserStory example as bigrams could be computed with the Equation (2). This probability is affected by the appearance of each term in the entire collection of messages (corpus).

$$P(W_n|W_{n-1}) = \frac{C(W_{n-1}W_n)}{C(W_{n-1})} \quad (2)$$

The Equation (2) is called maximum likelihood estimation or MLE. The MLE estimates for an n-gram model's parameters by getting counts from a corpus and normalizing the counts; therefore, the values lie between 0 and 1.

Let us see an example using bigrams with corpus of three sentences:

1. $*B*$ them. I am programming $*END*$
2. $*B*$ programming I am $*END*$
3. $*B*$ I do not like to design $*END*$

Below we show the result for some bigrams of this short corpus:

$$P(I \mid *B*) = \frac{2}{3} = 0.67,$$

the word I appears two times after $*B*$ and $*ENDS*$ appears three times.

$$\begin{aligned} P(*END* \mid programming) &= \frac{1}{2} = 0.5 \\ P(programming \mid *B*) &= \frac{1}{3} = 0.33 \\ P(programming \mid am) &= \frac{1}{2} = 0.5 \\ P(am \mid I) &= \frac{2}{3} = 0.67 \end{aligned}$$

To compute the probability of the first sentence, we would have the following expression:

$$\begin{aligned} P(S3) &= P(I \mid *B*) + P(am \mid I) + P(programming \mid am) \\ &= +P(*END* \mid programming) \\ &= 0.67 \times 0.67 \times 0.5 \times 0.5 = 0.000112225 \end{aligned}$$

The process estimates the n-gram probability by dividing the observed frequency of a particular sequence by the observed frequency of a prefix. This ratio is called a relative frequency [59]. The above example provides an overview of how language models work. This task is automated by the SRILM (http://www.speech.sri.com/, accessed on 29 October 2021) tool.

We created seven language models using SRILM version 1.7.3, a toolkit for building and applying language models, primarily used in speech recognition, statistical tagging and segmentation, and machine translation. In previous work [60], it was possible to identify that the language models had a better performance than the bag of words (BoW) method, a technique widely used to represent text data. In [61], a study on short texts in different languages showed results of 82.4 precision on average. Recently, pre-trained models are using a deep learning approach [62]; however, their implementation requires sizeable computational processing capabilities. Language models have not been used for the task addressed in this work, as indicated by some studies cited in the related work.

The configuration used for the creation of the models was as follows:

- N-grams-2 (bigrams): the sequence of terms was determined after performing an analysis between N-grams of sizes 2, 4, and 6.
- We used Kneser–Ney discounting (kndiscount) as a smoothing method.
- Gtmin: specify the minimum counts for N-grams to be included in the language model, we used gt2min.

We used all the terms to train the seven models. Repeated text segments were also automatically removed to avoid overfitting. Then, the text was lemmatized with Freeling (http://nlp.lsi.upc.edu/freeling/, accessed on 30 October 2021) with the default config file for Spanish, which also allowed to obtain the part-of-speech (POS) tags for the text. Below we provide an example of input for the training process of the user story model "ver que la historia del usuario este enfocada a un asistente medico/see that the user's story is focused on a medical assistant":

> *ver VMIP1S0 que CS el DA0FP0 historia NCFP000 de SP usuario NCMS000 estar VMIP3P0 enfocar VMP00PF a SP uno DI0MS0 asistente NCCS000 médico AQ0MS00*
>
> see VB that IN the DT user NN 's story NN is VBZ focused VBN on IN a DT medical JJ assistant NN

Definition of grammatical class:
VB: Verb, base form/IN: Preposition or subordinating conjunction/DT: Determiner/NNs: Noun, plural/NN: Noun, singular or mass/VBZ: Verb, 3rd person singular present/VBN: Verb, past participle/JJ: Adjective

The example above shows the lemmatized words (word lemma) and their respective grammatical classes. The generated language models can be downloaded from https://github.com/gborrego/autotagsakagsd (accessed on 16 November 2021).

Finally, when the models were generated, each new text evaluated generates a numerical value that indicates the closeness of the text to the trained model. The following section details the use of language models.

### *4.3. Language Models and Slack Tagging Service Integration*

Once we developed the language models, we integrated them with the Slack tagging service. To achieve this, we developed a REST web service in Python (recommendation service) to access the seven language models from any system capable of working with the HTTP protocol.

The operation of the seven language models together with the Slack tagging service is outlined in Figure 4, and described below:

1. An user clicks the button to start tagging a message, and the Slack tagging service does an HTTP request to the recommendation service sending the message text.
2. The recommendation service receives the text and sends it to each of the seven language models. Each model calculates a numerical value called perplexity, which expresses the confidence of the sentence tested in the language models. It is worth mentioning that the perplexity value is expressed between 0 and 1, where values close to zero indicate a powerful closeness. The measure used to compare the similarity in the language models was "average perplexity per word".
3. Once the recommendation service obtained the perplexity values of the seven language models; the values are ordered so that the first position represents the tag that fits better to the received text, according to the language models. Then, the service return to Slack an ordered list of tags, to be shown in the selection tag window (see Figure 2).

Although the described process seems cumbersome, the response time of the recommendation service is good enough not to cause any inconvenience to the end-user.

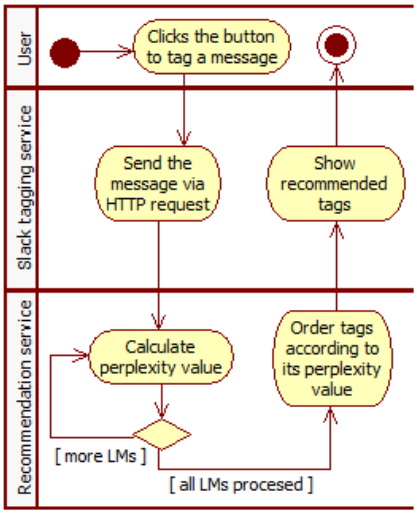


**Figure 4.** Activity diagram in unified modeling language, representing the integration of the Slack tagging service and the language models.

## 5. Materials and Methods

In this section, we present our method (based on the Wohlin et al. [63] guidelines) to evaluate the Slack tagging service. This method was based on the one followed in our past work [18]; even we reused part of the instrumentation in this experiment.

### *5.1. Objective*

To analyze the use of the Slack tagging service, to determine the semantic correctness of the suggested tags concerning the context of the tagged message, as well as comparing the time spent to tag message in a both-way: assisted by auto-completion and assisted by NLP, from the point of view of professional developers and students in the context of agile and global software engineering.

### *5.2. Planning*

According to the Wohlin et al. [63] guidelines, the planning phase consists of the following: experiment context definition, description of the subject selection, study design, variables and hypotheses definition, and a description of all the instruments required in the study.

#### 5.2.1. Context and Subjects Selection

This study had two contexts: academic and industrial. The academic context was at a computer lab at two public universities. The industrial context was in different software development companies, selecting a room where they could not be interrupted during the evaluation activities.

We chose the subjects of both contexts by convenience. The participants in the academic context were students enrolled in either a software engineering program or an information technology program. We took care that they already had been in courses related to agile methodologies and web development since the evaluation's context scenario was a Web project driven by an agile method. The industrial context participants were professional software developers with experience in agile/distributed development and web application projects.

#### 5.2.2. Study Design, Variables Selection, Furthermore, Hypotheses Formulation

This study is a quasi-experiment with a within-subject design because all the treatments were applied to all the participants. All the participants were mentally situated in a context scenario to interact through Slack with a counterpart; they used the tagging service (with predefined user tags) and followed a chatting guide that contains eight marks

suggesting what to tag, without specifying the tag to use. There were two types of marks: one indicates using a tag recommended by the Slack tagging service, and another indicates selecting a tag from the tags catalog using the auto-complete feature.

The independent variable is represented by the different ways of choosing tags, using either the tags recommendation feature or the tags auto-completion feature. The dependent variables were the semantic correctness of the chosen tag (see Section 5.3.3 to know how we determined the tag correctness) and the time to select a tag. Besides, we observed the number of new tags that the participants created during the evaluation and the perceived ease of use and perceived usefulness. Based on the previous variables, we stated two hypotheses for this study.

- $H_{0Correctness}$. There is no significant difference in the number of correctly tagged messages by using the tags recommendation feature or the auto-completion feature.
- $H_{0Time}$. There is no significant difference in time spent tagging a message by using the tags recommendation feature or the auto-completion feature.

It is worthwhile to notice that the hypothesis $H_{0Correctness}$ corresponds to the RQ1 and the hypothesis $H_{0Time}$ corresponds to the RQ2.

#### 5.2.3. Instrumentation

Below, we present the instruments developed to conduct this study.

- Context scenario. This scenario was used in our past work [18], and it concerned two agile developers from different companies and locations working on the same project (medical appointments system), one of whom required information about a RESTful service that the other was developing. They had documentation debt, and consequently, they had to acquire the project AK by asking questions to each other. A complete description of this scenario is located at https://github.com/gborrego/autotagsakagsd (accessed on 16 November 2021).
- Chatting guides. Each pair of participants had to follow two guides (one per scenario role) to simulate a technical conversation using Slack regarding the context scenario. The guides had marks indicating when to tag using either the recommendation or auto-completion features. It is important to highlight that these guides were based on the chatting scripts used in our past work [18] and the interactions based on these scripts comprised the corpus with which we developed the language models of this study. Furthermore, it is worth remarking that using a chatting guide does not mean that we wrote exact phrases to be copied by the participants during their interactions. This chatting guide contained hints about what to request or respond to cause variations in the participants' writing, but with the same hints' semantic; in this way, we could test the robustness of the language models. Both chatting guides are located at https://github.com/gborrego/autotagsakagsd (accessed on 16 November 2021).
- Slack tagging service, and were presented in Section 3. The service also registers when it was activated and when it was closed to obtain the time spent tagging a message.
- Messages gathering program. It is a program developed in Node.js which uses the Slack API to extract the messages of public channels and then sends them to the Algolia repository.
- Extended TAM questionnaire. We prepared a questionnaire in Google Forms which was based on the Technology Acceptance Model [64] (TAM) using a Likert-7 scale. Just as in our previous work [18], we extended this questionnaire adding items such: how the Slack component integrates to the daily work, and another in which we asked about enhancements that the participants would consider adding to the component to answer the RQ3. Furthermore, this questionnaire collected the following demographic data: age, years of experience in agile development, years of experience in distributed/global software development. The complete questionnaire could be viewed at https://github.com/gborrego/autotagsakagsd (accessed on 16 November 2021).

### *5.3. Operation*

In this part, we describe the four different stages of the study operation: the preparation stage, execution stage, data collection stage, and data validation stage.

#### 5.3.1. Preparation

We verified that the computers to be used in this study had an Internet connection and a web browser that could execute the Slack messenger. We created a Slack workspace, and we installed the tagging service. Then, we added the user tags which correspond to the context scenario. The user tags were the same that we used in our previous study [18]: ServiceIP, RESTTest (related to TechnologicalSupport meta-tag); RestApikey, RestSecurity, Encryption, TestData, RESTResource, RESTResponse (related to Code meta-tag); AngularEncryption (related to Component meta-tag); and UserStory (related to Documentation). Furthermore, we checked that the tags corresponding to the trained NLP models appear in the respective list.

#### 5.3.2. Execution

We had 52 participants which 30% were professional developers (26.4 years old on average, s.d. = 3.6) from three different Mexican companies, who declared an average of 2.7 years of experience in agile development (s.d. = 2.09) and 2.06 years (on average) of experience in global/distributed software development (s.d. = 2.5). The rest of the participants were students (21.6 years old on average, s.d. = 2.7) from two different Mexican universities. The evaluation sessions consisted of three parts which we explain below.

- Introduction (duration 10 min). We explained to the participants the study sessions along with their objectives. We organized the participants in pairs (to chat between them), and then we asked their email addresses to register them on the Slack workspace. We created a public channel for each pair of participants (to isolate the pairs' conversations), and we helped them complete the Slack registration. We gave the participants a short training session regarding how to use the tagging service (3 min, approx.), and they quickly explored the available tags (2 min, approx.). Then, we described the scenario in which they would be located to carry out the tasks, and we assigned a role to each pair member: either the developer working on a RESTful service or the developer who wished to use it. We gave them the corresponding guides, and we explained to them that the guides had marks indicating when to use the recommendation feature. Each pair member sat in a different part of the session room, ensuring they had no visual contact as if they were geographically distributed. We asked them to avoid talking to each other to emulate an environment of geographic distribution better.
- Interacting through Slack (duration 25 min). Following the corresponding guide, the participants used Slack to chat, and the tagging service aided them. We explained that they could paraphrase the messages since we gave them only a guide, not a script. We also told the participants that they could write a new tag (unregistered/invalid tag) if they could not find one that fitted a certain message on the options shown by the tagging service.
- Finalization (duration 3 min). When the participants had finished chatting through Slack, they answered the TAM-based questionnaire. Finally, we executed the message-gathering program to send all the channel conversations to a common repository.

#### 5.3.3. Data Collection and Data Validation

The tagged messages from each conversation were collected from the repository, and we executed a script to determine whether the used tags were semantically correct or not. This process consisted of comparing the tagged messages against the expected tags, accounting that the meta-tags hierarchy of each tag was also considered correct. We noticed that participants also tagged messages that were not marked. Two expert persons manually evaluated whether the tag fitted the message semantics in those cases.

Besides, we extracted from the tagging service log file the closing and opening time of the tagging dialog to obtain the time spent to select a tag for each participant. Furthermore, we obtained the answers to the TAM questionnaire directly from Google Forms. Finally, we discarded those tagged messages that did not correspond to the chatting guide, such as the initial testing messages and those written after the final line of the chatting guide. The curated data set of the gathered messages is located in the following link: https://github.com/gborrego/autotagsakagsd (accessed on 16 November 2021).

## 6. Results

We obtained 356 tagged messages, where 50.6% were tagged using the auto-completion feature and 49.4% were tagged using the recommendation feature. We present the results organized as follows: one section per each stated hypothesis, another section in which we present the TAM results, and a section where we present additional observations of this study.

### *6.1. Tagging Correctness*

In this part, it is worth remembering that the participants did not register any tag for this controlled study, as it is supposed to happen in a knowledge condensation implementation. For this reason, we allowed the participants to create tags when they considered that the predefined tags/meta-tags did not fit a message. We obtained that 14% of the messages were tagged with a tag created by the participants (new tags), from which 20% were incorrect, and 80% were correct. Furthermore, 25% of messages with user-created tags were considered synonyms of the predefined tags. Thus, to calculate the number of messages tagged correctly using the auto-completion feature, we summed the messages correctly tagged with the existing tags and those correctly tagged using a synonym tag. Moreover, we added the number of correctly tagged messages using the recommendation feature. It is worth remembering that we excluded the messages tagged with a new tag and included synonyms tags. Figure 5A shows that participants had more messages correctly tagged when they used the NLP recommendation feature.

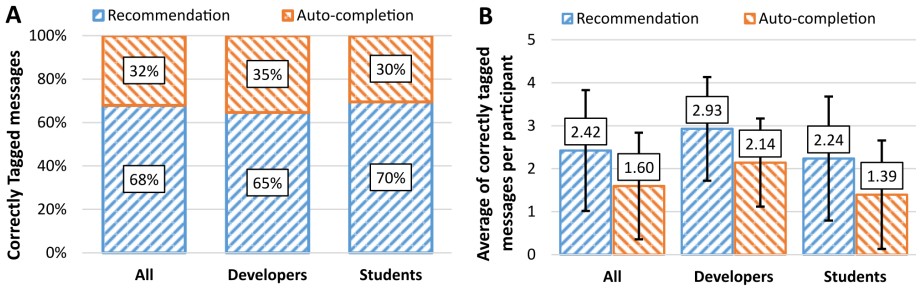


**Figure 5.** Tagging correctness using the recommendation feature and auto-completion feature, classified by participants profile. (**A**) Percentage of correct tagging on both ways. (**B**) Average of correct tagging by participant (whiskers represent standard deviation).

On the other hand, if we group the data by participants, the above-stated difference is also evident on the average of correctly tagged messages per participant on both participant profiles (see Figure 5B). We verified whether both data sets (average of correctly tagged messages using recommendation and auto-completion) were normally distributed by applying the Kolmogorov–Smirnov normality test. We obtained that the set of the tagged messages using the auto-completion feature did not fit the normal curve ($D = 0.18621$, $p$-value = 0.04731). Thus, we applied the Wilcoxon signed-rank test ($\alpha = 0.05$) to determine whether the difference between the average of correctly tagged messages per participant using recommendation and auto-completion feature was significant. We obtained that the difference is statistically significant ($W = 237.5$, $p$-value = 0.00452), hence, this result provides evidence to reject the null hypothesis $H_{0Correctness}$. With this result, we can answer

the RQ1 saying that the participants tagged messages more coherently when they used the NLP recommendation.

### *6.2. Time Spent to Tag Messages*

We obtained that the participants spent less time tagging when the recommendation feature was used ($avg = 11.03$ s, $std = 5.23$), either in general, or per each participant profile (see Figure 6A). In this case, we applied a $T$-test since both sets of data were normally distributed according to the Kolmogorov–Smirnov test (Time using recommendation: $D = 0.11737$, $p$-value = 0. 48686—Time using auto-completion: $D = 0.12835$, $p$-value = 0.34096). By applying $T$-test ($\alpha = 0.05$), we obtained that the difference between the time spent to tag by auto-completion and to tag using the recommendation feature was significant ($T = -5.10628$, $p$-value < 0.00001).

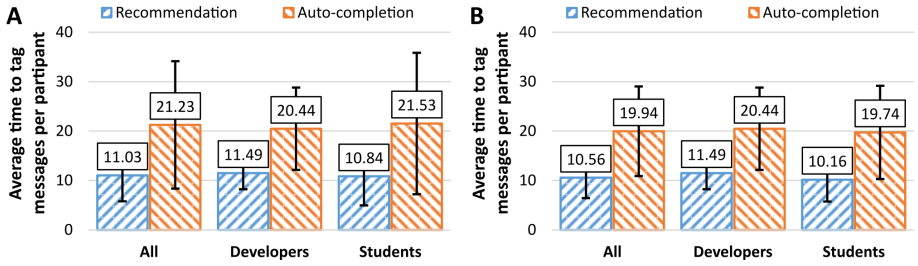


**Figure 6.** Time (in seconds) participants spent tagging a message using the recommendation and auto-completion features, classified by participants profile. (**A**) Results with outlier values, and (**B**) results without the outlier values (whiskers represent standard deviation).

To deeply explore this result, we discarded the outlier values of both data sets. Those values were obtained by applying the Grubbs' test with $\alpha = 0.05$ (Time using recommendation: $Z = 4.2314$, outlier-value = 33.18—Time using auto-completion: $Z = 5.0222$, outlier-value = 85.90); the difference was still visually present, as is shown in (Figure 6B). Removing the outlier values gives us that the average time was 10.56 s ($std = 4.13$) using the recommendation feature, and 19.94 s ($std = 9.06$) using auto-completion feature. We applied again $T$-test ($\alpha = 0.05$) and the difference was also statistically significant ($T = -6.48889$, $p$-value < 0.00001), thus, we can reject the null hypothesis $H_{0Time}$. With this result, we can answer the RQ2 saying that the participants tagged messages faster when they used the NLP recommendation.

### *6.3. Additional Observations*

We noticed that 7.3% of the messages were tagged using the recommendation feature when it was not indicated in the guide, where 57% of the 7.3% were correctly tagged. It is important to remember that the language models were developed using tagged messages obtained from our previous study [18]. This fact indicates that the language models used to detect the context of the messages are robust enough to support different ways of writing mixed messages of the same topic. The average of correct and incorrect tagged messages per participant was 0.89 ($std = 0.90$) and 0.67 ($std = 0.68$), respectively. We applied the Kolmogorov–Smirnov test of normality and we obtained that both sets, correct and incorrect tagged messages, were normal (correct messages: $D = 0.23407$, $p$-value = 0.23742—incorrect messages: $D = 0.28579$, $p$-value = 0.0855). Hence, we applied a $T$-test ($\alpha = 0.05$), obtaining that the difference between both sets is not significant ($T = 0.83299$, $p$-value = 0.205331), which is an expected result since we did not train the language models to support the kind of messages where participants tagged.

In addition, when participants used the recommendation feature, and they tagged correctly, 85% of the time, the selected option was in the first position of the tag's list (see Figure 7A). This difference is significant according to the goodness of fit test based on $\chi^2(\alpha = 0.05)$, assuming a distribution on which the first position has 75% and the rest

25% ($\chi^2 = 7.714$, $p$-value = 0.00548). On the other hand, when the participants used the recommendation feature, and they tagged incorrectly, 68% of the times the correct option was located in the first position of the list (see Figure 7B). This difference is significant according to the goodness of fit test based on $\chi^2(\alpha = 0.05)$, assuming a distribution on which the first position has 58% and the rest 42% ($\chi^2 = 4.599$, $p$-value = 0.032). This result means that the language models were preciser than the participants when tagging was unexpected. The correctly recommended tags were the following: DataTest, Documentation, TechnologicalSupport, AngularEncryption, RestResource, and UserStory.

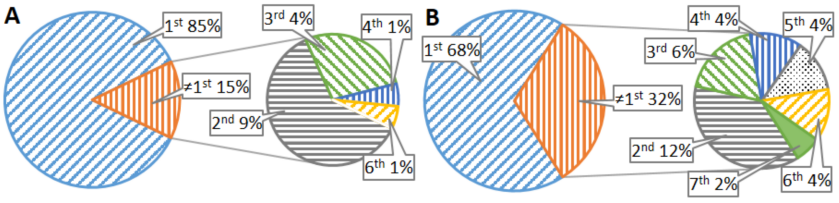


**Figure 7.** List position of the correct option on the recommendation feature when participants tagged correctly (**A**) and incorrectly (**B**) while using the recommendation component.

Finally, we obtained 176 messages tagged using the recommendation feature, without the unexpected tagged messages, wherein 112 messages the correct tag was in the first position of the recommendation list. With these data we can calculate a $Precision = 112/176 = 0.6363$ of NLP recommendation method. On the other hand, if we include the unexpected tagged messages, we have 204 tagged messages, wherein 125 messages the correct tag was in the first position of the recommendation list, i.e., now $Precision = 125/204 = 0.6127$. Furthermore, the difference between correct and incorrect recommendations is significant according to the goodness of fit test based on $\chi^2(\alpha = 0.05)$, assuming a distribution on which the correct recommendations has 56% and the rest 44% ($\chi^2 = 4.165$, $p$-value = 0.04126). The difference between correct and incorrect recommendations is also significant according to the goodness of fit test based on $\chi^2(\alpha = 0.05)$, assuming a distribution on which the correct recommendations has 54% and the rest 46% ($\chi^2 = 4.346$, $p$-value = 0.0371).

### *6.4. Qualitative Results*

Participants responded to the TAM questionnaire after chatting through Slack using the chatting guide. We gathered all the questionnaire answers (based on Likert-7), and we applied the Cronbach alpha test to determine its consistency. We obtained $\alpha = 0.9641$ with 95% of confidence, which means that the TAM results present an excellent internal consistency. Analyzing the obtained results, participants perceived the implemented Slack tagging service as very useful and easy to use in general (median of 6 in Likert-7 scale) (see Table 2), highlighting that for both measures (usefulness and ease of use), the mode was 7 in Likert-7 scale. In Table 2, we observe that all the Likert values obtained for usefulness and ease of use represent a positive perception (greater of 4), which means that most of the participants perceived the tagging service as useful. However, there were some outlier values: developers' ease of use perception with values of 2, 3, and 4, and students' perception on each aspect, with 1. These low values could be related to the received suggestions about the pre-loaded tags; participants mentioned would have liked to participate in the tags registration. Furthermore, participants commented that the way of tagging must be improved; however, it entirely depends on the Slack conditions to create complements or services. Remarkably, participants suggested tagging before sending a message, not when the message has already been sent, which is how most tagging services currently work. Some comments of the participants accompany these results: *"...tagging messages is a bit tedious... I do not think that people would use it in an agile development..."*, which indicates to us that the service operation has to be improved. Finally, we extended the TAM questionnaire with one question about how the tagging service integrates into

daily work. We obtained that participants think that this service integrates easily to the working conditions of agile GSE projects. However, we obtained a value lower than 5 (Likert-7 scale) on quartile 25 and some outliers values (see Table 2), which could be related to the above-reported problems too. To summarize, we can state that participants perceived the Slack tagging service as very easy to use, useful and that it could be easily integrated into the daily work. With the previous statement, we answer the RQ3.

**Table 2.** Results of the extended TAM questionnaire, by participant profile. All the values are based on a Likert-7 scale.

| | **Students** | | | **Developers** | | |
|---|---|---|---|---|---|---|
| | **Usefulness** | **Ease of Use** | **Integration** | **Usefulness** | **Ease of Use** | **Integration** |
| Mode | 7 | 7 | 7 | 7 | 7 | 7 |
| Median | 6 | 6 | 6 | 6 | 7 | 6 |
| Q.75 | 7 | 7 | 7 | 7 | 7 | 7 |
| Q.50 | 6 | 6 | 6 | 6 | 7 | 6 |
| Q.25 | 5 | 5 | 5 | 5 | 6 | 4.25 |

## 7. Threats to Validity

To understand how valid the results are and how researchers and practitioners can use them, we present the validity threats according to Wohlin et al.'s [63] specification.

### *7.1. Conclusion Validity*

This study had participants from different backgrounds and experiences using Slack. To balance the participants' knowledge about Slack, they received a brief training regarding using this application and how to use the tagging service. The participants were anonymous in this study. We did not know them before or after the evaluation; therefore, there was no reason to try to please us. Furthermore, we are aware that the evaluation period was relatively short; however, the results indicate a clear trend about the benefits of using the implemented Slack service.

### *7.2. Internal Validity*

The study's sessions were short to prevent the participants from getting bored or tired. We attempted to avoid learning effects by using a counterbalancing technique, i.e., we placed the participants in groups and presented the conditions (auto-completion and recommendation) to each group in a different order. The TAM questionnaire results showed an excellent internal consistency when applying the Cronbach Alpha test. Besides, all the participants were volunteers and showed interest in collaborating in this study. Regarding persistence effects, the study was executed with subjects who had never taken part in a similar study. Moreover, the participants did not have any previous knowledge of the context scenario since it was fictitious.

### *7.3. Construct Validity*

We measured the required time to tag a message using the tagging service, which registered when the tagging screen was shown and when it was closed by the participants. This time could have been affected by the participants' reading speed and by the time to decide a proper tag; however, we considered that participants had similar abilities, and the arrangement of the sets could have reduced this threat. To discover the participants' perceptions about how the tagging service integrates into their daily work, we extended the standard TAM questionnaire by adding one question with the same structure as TAM questions. This extension provided us with a structured means to obtain the participants' perceptions of topics that the conventional TAM questionnaire does not include.

### *7.4. External Validity*

We identified two main threats to external validity: subjects and tasks/materials. Regarding subjects' threats, we included students to have more controlled conditions to conduct the study in an academic context. Unfortunately, these students had no experience with real agile GSE projects; however, they had already taken courses concerning agile software development and had been in touch with GSE topics during their university education. We included professional developers with experience in agile GSE to enforce external validity. Concerning threats about tasks and materials, the chatting guides were based on a fictitious scenario but included real-world situations and characteristics.

## 8. Discussion

This section discusses our results in two ways: implications about the AK condensation in agile GSE and natural language models. Finally, we present a summary of the contributions of this paper.

### *8.1. Implications about Architectural Knowledge Condensation in Agile Global Software Engineering*

Literature reports that developers prefer to share AK with their counterparts using UTEM in agile GSE environments [10,19], mainly because of the language and culture differences. Thus, a mechanism to condense AK from UTEM logs is essential since these logs are the only source of AK many times [9,10]. A crucial part of the AK condensation is the classification mechanism, and it was implemented by using social tagging during UTEM interactions. Thus, efficient and coherent tagging is essential to retrieve AK in the future.

In this study, we observed that participants tagged coherently more frequently when they used the recommendation feature (RQ1). In our previous study [18], we obtained 62% of correctly tagged messages; in this study, we obtained 66%, where 32% corresponds to tagging by the auto-completion feature, and 68% corresponds to tagging by the recommendation feature. Thus, tag recommendations were essential to raising the correctness percentage. Regarding the spent time to tag a message, the use of the recommendation feature represents a significantly faster way to select tags (RQ2). In general, the time difference was nine second, which could represent the difference between a good or bad user experience [26], and consequently an essential factor to adopt the tagging service and the KC concept.

Regarding the perceived ease of use and the perceived usefulness, the Slack tagging service obtains high values of the Likert-7 scale, even on quartile 25, which means that participants consider the service very useful and usable (RQ3). Furthermore, participants considered that tagging service could be integrated into the daily work activities (RQ3). Further, there are points to enhance regarding tagging a message; however, it heavily depends on Slack's conditions to develop a complement. This situation could happen to any other UTEM since they have those conditions for developing plugins to complement its operation. We know that determining any software's usability implies more than applying questionnaires; usability testing includes observations, video-recording analysis, interviews, focus groups, etc. However, the main focus of this paper was to evaluate the Slack service in terms of time and coherence at tagging messages, and the TAM questionnaire was only to have a preliminary view of the Slack service's usability and usefulness.

The results showed that the semi-fixed tagging approach supported by the tags recommendation feature is an efficient (precise and fast) and well-perceived way to classify AK during UTEM interactions in an agile GSE environment. Furthermore, the results could reduce the cognitive load to agile GSE developers during their daily work (reducing the effort to think how to tag) and ease the AK finding since the AK is going to be aligned to the classification scheme, on which the tags are based. In this way, a developer could recover AK from a past project, using the tags and meta-tags to explore the project AK.

All the presented results led us to think that a recommendation feature could be essential for better AK classification when tagging is used as a classification mechanism (part of the AK condensation concept). This is because the tagging correctness percentage could increase in general; thus, the projects' AK could be better classified.

Considering the results of our study, we can state that the AK condensation concept could be implemented in the industry in the following way:

- We must implement a mechanism to determine how well a message was tagged. This implementation could run when developers look for projects' AK, using the searching mechanism proposed by the AK condensation concept; thus, when they find AK, they also could qualify the tagging. It is important to notice that the searching mechanism has as a source a repository where the UTEM logs will be stored; thus, it can contain messages from email, code repositories (commit comments), instant messengers, etc.
- Another mechanism could take from the messages repository the tagged ones with a good qualification; thus, determining when is a good qualification would be crucial.
- The same mechanism could group the messages by tag, and finally, pass the grouped messages to a final process to update the current language models or generate new ones based on new tags.

In this manner, a development team could have an updated set of language models, which would be helpful to suggest tags during UTEM interactions to condense AK.

### *8.2. Implications about Natural Language Models in Agile Global Software Engineering*

The TAM evaluation provides encouraging results; developers and students identified the tag recommendation component as useful (based on language models and incorporated into the Slack tagging service) in software projects. The correctness percentage obtained by the recommender component gives evidence that the language models captured the sentence structure written by the participant. For instance, the language model identifies the word "story" with a probability of 0.75, and the phrase "user_story" is 0.76; that is, the terms used by the participants are very close to those used in the creation stage of the models.

However, for the terms that have a low probability, in some cases, they complement the probability of the grammatical class. For example, the model's question mark symbol ("?") has a probability of 0.05, but with its grammatical class, the probability increases to 0.97. The probabilities of the term and the grammatical class complement each other and give the tagging service robust performance, reflecting in the correctness percentage. The Kneser–Ney function softens those terms out of the vocabulary of the language model as a smoothing method. The precision results of the implemented models may indicate that using language models is promising. We observed that the small vocabulary of the corpus had a positive impact on the models, especially on tags where the frequency of repetition of the terms is high, such as "UserStory".

Concerning the precision of the language models, it could be considered a good score ($precision = 0.6363$ without including the unexpected tagged messages), taking into account the low amount of data (291 messages obtained from our previous study [18]) used to train the language models. For instance, works based on YouTube [47] or Stackoverflow [45,51,53] had available an extensive corpus (tagged posts and video tutorials of different years) to train the recommendation models. In particular, Parra et al. [47] reported that up to 66% of the tagged videos were considered relevant by developers. Thus, it could be interpreted that they obtained 0.66 of precision, which is pretty close to our precision value. Moreover, considering our acceptable results and our small corpus to train the language models, we could say that our approach presents certain tolerance to the "cold start" problem. However, we have to conduct more experiments to determine its robustness.

It is worth remarking that 68% of the times that participants tagged incorrectly, the correct option was the first element of the recommendation list. However, participants decided to choose another list element. This situation leads us to improve the language models of

the recommendation feature to explore a mechanism of semi/auto-tagging. In this way, when a developer wants to tag a message, s/he could use this mechanism to assign a tag suggested by a language model.

We addressed the above-presented problem by training a model for each category due to the size of the corpus used to build them. However, each model provides the system with a refined and specific analysis. One aspect that we can identify to improve is incorporating more elements in the training corpus to reach a higher percentage of correct ones. Increasing the number of language models could leverage the operation of the Slack tagging service and the tagging process in an agile GSE environment. We created the models to support the Spanish language; however, we can build models for other languages by collecting a new corpus, for instance, English. Another rapid but less effective option would be to map our corpus to English, assuming that developers' communication is standardized. Finally, because of the success of our simple models, we did not investigate more complex recent models like BERT [62]. Such models can improve performance but would increase computational cost.

### *8.3. Summary of Contributions*

The main contributions of our work could be summarized as four aspects, which are depicted below without a relevance order.

First, we implemented a Slack tagging service with an NLP recommendation-based feature that helps developers choose a correct tag regarding a conversation context in terms of AK. A coherent tagging is crucial to implementing the AK condensation concept. A coherent tagging would be the base to find AK when developers do not remember the exact phrases to search a UTEM interaction where AK was shared, accounting that UTEM logs usually are the only source of AK in agile GSE.

Second, the evaluation of the Slack tagging service provided statistical evidence to show that users tag faster ($T = -6.48889$, $p$-value $< 0.00001$) and with better semantic correctness, i.e., more coherently ($W = 237.5$, $p$-value = 0.00452), using the NLP recommendation feature than using the auto-completion one. As was mentioned above, coherent tagging is crucial to implement the AK condensation concept. Thus, having a service that helps accurately recommend relevant tags means a step towards industrial implementation of the AK condensation context. In the same sense, the implemented service allows to tag quickly, which could minimize the negative cost that interruptions cause [22,23], doing developers to redirect their attention from the main activity. Fast tagging represents a better usability perception, which also increases the software adoption [26].

Third, students and professional developers perceived the Slack tagging service as useful and easy to use. The evaluation was done using a TAM-based questionnaire, in which we obtained high Likert-7 values (e.g., 5 and 6) in low quartiles as Q.25. Furthermore, participants considered that the Slack tagging service could be easily integrated into the daily work of a software project. Although the participants' perception was good, we found aspects regarding the ease of use that could be improved if Slack offered more open elements for its extension. For the AK condensation concept, it is essential that the classification mechanism, implemented through the Slack tagging service, be embedded as naturally as possible in the developers' activities.

Fourth, the developed language models were robust enough to give coherent tag recommendations (participants tagged better using the NLP feature), even on parts of the chatting guide where we did not expect to be tagged. Thus, the developed language models could work with very different ways to write a given topic. This result encourages us to keep developing more language models to cover different AK topics to have diverse tag recommendations.

Fifth, we developed a corpus of 291 tagged messages contained in seven tags, which came from our past study [18]. This corpus could be used by academia to explore more possibilities to expand this work.

## 9. Conclusions

In agile GSE, a critical amount of AK is stored in the UTEM logs, and most of the time, these are the only AK source in a development team [9,10]. AK condensation is an approach that proposes a way to classify and recover AK from the UTEM logs. An implementation of this concept is based on tagging UTEM interactions to classify and ease AK retrieval.

In this paper, we presented the Slack tagging service (an implementation of an AK classification mechanism), which allows to tag messages based on a classification scheme obtained in a previous study [9]. The objective of this service was to help developers remember the available tags during their daily interactions, by auto-completing tags or by offering tags recommendations through NLP techniques and language models. Students and professional developers evaluated the Slack tagging service in a controlled experiment to compare both tagging mechanisms in terms of correctness and time to select a tag. The results statistically showed that more than 64% of the Slack messages were tagged correctly (i.e., tags were semantically related with the messages) using the tags recommendation, considering that the participants were unaware of the existing tags and the interaction context. Furthermore, when participants used the recommendation feature, they spent less time selecting a tag (statistically significant too), which is beneficial to the potential adoption of the Slack tagging service and the AK condensation concept. It is worth remarking that although the language models were based on a corpus in Spanish, we can use the same reported process to develop new language models based on a corpus in English, which is a common language in agile GSE interactions.

Further, the Slack tagging service was perceived as highly useful and usable, according to the TAM questionnaire results. However, we are aware that a more robust study is needed to determine the actual usability and user experience of the Slack service.

The obtained results are remarkable since the corpus volume to train the language models was significantly smaller than other approaches reported in the literature, which used sources as StackOverflow entries to train the recommendation models. Besides, our approach is focused on AK entirely dependent on software projects' circumstances and conditions, such as business rules clarifications, architectural decisions, and even deployment topics. Thus, we cannot use public Q&A services since these sources contain more general AK than the knowledge shared through UTEM during a software project.

Furthermore, our results reinforce the AK condensation concept feasibility, and that semi-fixed tagging could be an AK classification mechanism adequate for the agile GSE environment. Furthermore, these results lead us to conduct a long-term study in an agile GSE company to evaluate the concept adoption and evaluate how AK condensation could affect metrics such as time to finish tasks, interruptions quantity, team efficiency, and product quality. This long-term study will help us increase the training corpus's size for language models. Further, we will explore new ways to implement an AK classification mechanism, for instance, using ontologies or deep learning techniques. We will even explore new approaches to condense AK on any agile software development environment, either co-located or distributed or using new sources such as transcriptions, audio, or video, which are also used to share knowledge in agile GSE.

Another direction in our future work is to conduct a qualitative or mixed study to get the perception of software architects about the AK that could be condensed with our approach. This study should cover micro-architecture topics since our proposal focuses on the developers' daily interactions; however, this proposal could support architects' interactions, where macro-architecture topics could be more prevalent. Furthermore, developers could discuss architecture topics through UTEM that could be relevant at the macro-architecture level. Thus, a software architect's point of view will be relevant to determine the impact of the condensed AK, regardless its abstraction level.

**Author Contributions:** Conceptualization, Methodology, Software, Formal Analysis, Investigation, Writing—Original Draft, Writing—Review and Editing, Supervision, Project Administration, Visualization, G.B.; Conceptualization, Methodology, Software, Investigation, Data Curation, Writing—Original Draft, Writing—Review and Editing, S.G.-L.; Validation, Data Curation, Writing—Original Draft, Writing—Review and Editing, R.R.P. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported by the "Programa de Fortalecimiento a la Investigación 2021" (project numbers: *PROFAPI*_2021_0062 and *PROFAPI*_2021_0065) of the Instituto Tecnológico de Sonora. Furthermore, this work had support of the "Programa para el desarrollo profesional docente" for the first author (*ITSON-PTC*-114).

**Institutional Review Board Statement:** The study was approved by the Institutional Committee of Bioethics of the Instituto Tecnológico de Sonora (date of approval: 23 November 2020, approval code: 132).

**Informed Consent Statement:** Informed consent was obtained from all subjects involved in the study.

**Data Availability Statement:** All the data obtained in this study, as well as the generated languaje models, chatting scripts, and TAM questionnaire can be downloaded from https://github.com/gborrego/autotagsakagsd (accessed on 16 November 2021).

**Acknowledgments:** The authors are grateful to the participating companies: EMCOR, Sahuaro Labs and Tufesa, as well as the students of Instituto Tecnológico de Sonora and Universidad Tecnológica del Sur de Sonora, for the support provided to carry out the present study.

**Conflicts of Interest:** The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

## Abbreviations

The following abbreviations are used in this manuscript:

| | |
|---|---|
| AK | Architectural Knowledge |
| GSE | Global Software Engineering |
| NLP | Natural Language Processing |
| TAM | Technology Acceptance Model |
| UTEM | Unstructured Textual and Electronic Media |

## References

1. Vliet, H.V. Software architecture knowledge management. In Proceedings of the IEEE 19th Australian Conference on Software Engineering (ASWEC 2008), Perth, Australia, 25–28 March 2008; pp. 24–31. [CrossRef]
2. Kruchten, P., Documentation of Software Architecture from a Knowledge Management Perspective—Design Representation. In *Software Architecture Knowledge Management: Theory and Practice*; Springer: Berlin/Heidelberg, Germany, 2009; Chapter 3, pp. 39–57. [CrossRef]
3. Dalkir, K. *Knowledge Management in Theory and Practice*, 2nd ed.; MIT Press: Cambridge, MA, USA, 2011; p. 485.
4. Holmstrom, H.; Conchuir, E.O.; Agerfalk, P.J.; Fitzgerald, B. Global Software Development Challenges: A Case Study on Temporal, Geographical and Socio-Cultural Distance. In Proceedings of the International Conference on Global Software Engineering (ICGSE '06), Florianopolis, Brazil, 16–19 October 2006; pp. 3–11. [CrossRef]
5. Zahedi, M.; Shahin, M.; Ali Babar, M. A systematic review of knowledge sharing challenges and practices in global software development. *Int. J. Inf. Manag.* **2016**, *36*, 995–1019. [CrossRef]
6. Beck, K.; Beedle, M.; van Bennekum, A.; Cockburn, A.; Cunningham, W.; Fowler, M.; Grenning, J.; Highsmith, J.; Hunt, A.; Jeffries, R.; et al. Manifesto for Agile Software Development. Web, 2001. Available online: https://agilemanifesto.org/ (accessed on 20 October 2021).
7. Cockburn, A.; Highsmith, J. Agile Software Development:The People Factor. *Computer* **2001**, *34*, 131–133. [CrossRef]
8. Yanzer Cabral, A.R.; Ribeiro, M.B.; Noll, R.P. Knowledge Management in Agile Software Projects: A Systematic Review. *J. Inf. Knowl. Manag. (JIKM)* **2014**, *13*, 1450010. [CrossRef]
9. Borrego, G.; Moran, A.A.L.; Palacio, R.R.; Rodriguez, O.M.O.; Morán, A.L.; Palacio, R.R.; Rodríguez, O.M.; Moran, A.A.L.; Palacio, R.R.; Rodriguez, O.M.O. Understanding architectural knowledge sharing in AGSD teams: An empirical study. In Proceedings of the 11th IEEE International Conference on Global Software Engineering, Irvine, CA, USA, 2–3 August 2016; pp. 109–118. [CrossRef]

10. Clerc, V.; Lago, P.; Vliet, H.V. Architectural Knowledge Management Practices in Agile Global Software Development. In Proceedings of the IEEE Sixth International Conference on Global Software Engineering Workshop, Helsinki, Finland, 15–18 August 2011; pp. 1–8. [CrossRef]
11. Tom, E.; Aurum, A.; Vidgen, R. An exploration of technical debt. *J. Syst. Softw.* **2013**, *86*, 1498–1516. [CrossRef]
12. Bider, I.; Söderberg, O. Becoming Agile in a Non-Disruptive Way. In Proceedings of the 18th International Conference on Enterprise Information Systems (ICEIS 2016), Rome, Italy, 25–28 April 2016; SCITEPRESS—Science and Technology Publications, Lda.: Setubal, Portugal, 2016; pp. 294–305. [CrossRef]
13. Bosch, J. *Software Architecture: The Next Step*; EWSA; Volume 3047, Lecture Notes in Computer Science; Oquendo, F., Warboys, B., Morrison, R., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; pp. 194–199.
14. Holz, H.; Melnik, G.; Schaaf, M. Knowledge management for distributed agile processes: Models, techniques, and infrastructure. In Proceedings of the IEEE Enabling Technologies: Infrastructure for Collaborative Enterprises, Linz, Austria, 11 June 2003; pp. 291–294. [CrossRef]
15. Uikey, N.; Suman, U.; Ramani, A. A Documented Approach in Agile Software Development. *Int. J. Softw. Eng.* **2011**, *2*, 13–22.
16. Rios, N.; Mendes, L.; Cerdeiral, C.; Magalhães, A.P.F.; Perez, B.; Correal, D.; Astudillo, H.; Seaman, C.; Izurieta, C.; Santos, G.; et al. Hearing the Voice of Software Practitioners on Causes, Effects, and Practices to Deal with Documentation Debt. In *Requirements Engineering: Foundation for Software Quality*; Madhavji, N., Pasquale, L., Ferrari, A., Gnesi, S., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 55–70.
17. Ryan, S.; O'Connor, R.V. Acquiring and sharing tacit knowledge in software development teams: An empirical study. *Inf. Softw. Technol.* **2013**, *55*, 1614–1624. [CrossRef]
18. Borrego, G.; Morán, A.L.; Palacio, R.R.; Vizcaíno, A.; García, F.O. Towards a reduction in architectural knowledge vaporization during agile global software development. *Inf. Softw. Technol.* **2019**, *112*, 68–82. [CrossRef]
19. Estler, H..; Nordio, M.; Furia, C.A.; Meyer, B.; Schneider, J. Agile vs. Structured Distributed Software Development: A Case Study. In Proceedings of the 2012 IEEE Seventh International Conference on Global Software Engineering, Porto Alegre, Brazil, 27–30 August 2012; pp. 11–20.
20. Bagheri, E.; Ensan, F. Semantic tagging and linking of software engineering social content. *Autom. Softw. Eng.* **2016**, *23*, 147–190. [CrossRef]
21. Nagwani, N.K.; Singh, A.K.; Pandey, S. TAGme: A topical folksonomy based collaborative filtering for tag recommendation in community sites. In Proceedings of the 4th Multidisciplinary International Social Networks Conference (MISNC '17), Bangkok, Thailand, 17–19 July 2017. [CrossRef]
22. Bailey, B.P.; Konstan, J.A. On the need for attention-aware systems: Measuring effects of interruption on task performance, error rate, and affective state. *Comput. Hum. Behav.* **2006**, *22*, 685–708. [CrossRef]
23. Altmann, E.; Trafton, J.; Hambrick, Z. Momentary Interruptions Can Derail the Train of Thought. *J. Exp. Psychol. Gen.* **2014**, *143*, 215–226. [CrossRef]
24. Czerwinski, M.; Horvitz, E.; Wilhite, S. A Diary Study of Task Switching and Interruptions. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04), Vienna, Austria, 24–29 April 2004; Association for Computing Machinery: New York, NY, USA, 2004; pp. 175–182. [CrossRef]
25. Bailey, B.P.; Iqbal, S.T. Understanding Changes in Mental Workload during Execution of Goal-Directed Tasks and Its Application for Interruption Management. *ACM Trans. Comput.-Hum. Interact.* **2008**, *14*, 1–28. [CrossRef]
26. Nielsen, J. *Usability Engineering*; Interactive Technologies; Elsevier Science: Amsterdam, The Netherlands, 1994.
27. Kruchten, P.; Lago, P.; Vliet, H.V. Building Up and Reasoning About Architectural Knowledge. In Proceedings of the Second International Conference on Quality of Software Architectures, Västerås, Sweden, 27–29 June 2006; pp. 43–58._8. [CrossRef]
28. Bass, L.; Clements, P.; Kazman, R. *Software Architecture in Practice*, 2nd ed.; Addison-Wesley Professional: Boston, MA, USA. 2003; p. 560.
29. ISO/IEC/IEEE. Systems and Software Engineering—Architecture Description. In *ISO/IEC/IEEE 42010:2011(E)*; IEEE: New York, NY, USA, 2011. [CrossRef]
30. Kruchten, P. *The Rational Unified Process: An Introduction*, 3rd ed.; Addison-Wesley Longman: Reading, MA, USA, 2003.
31. Clements, P.; Kazman, R.; Klein, M. *Evaluating Software Architectures: Methods and Case Studies*; SEI Series in Software Engineering; Addison-Wesley: Boston, MA, USA, 2001.
32. Vogel, O.; Arnold, I.; Chughtai, A.; Kehrer, T. *Software Architecture: A Comprehensive Framework and Guide for Practitioners*; Springer: New York, NY, USA, 2011. [CrossRef]
33. Dorairaj, S.; Noble, J.; Malik, P. Knowledge Management in Distributed Agile Software Development. In Proceedings of the IEEE 2012 Agile Conference, Dallas, TX, USA, 13–17 August 2012; pp. 64–73. [CrossRef]
34. Jiménez, M.; Piattini, M.; Vizcaíno, A. Challenges and Improvements in Distributed Software Development: A Systematic Review. *Adv. Softw. Eng.* **2009**, *2009*, 710971. [CrossRef]
35. Awar, K.; Sameem, M.; Hafeez, Y. A model for applying Agile practices in Distributed environment: A case of local software industry. In Proceedings of the 2017 International Conference on Communication, Computing and Digital Systems (C-CODE 2017), Islamabad, Pakistan, 8–9 March 2017; pp. 228–232. [CrossRef]
36. Sneed, H.M. Dealing with Technical Debt in agile development projects. In Proceeding of the 6th International Conference (SWQD 2014), Vienna, Austria, 14–16 January 2014; pp. 48–62. [CrossRef]

37. Clear, T. Documentation and Agile Methods: Striking a Balance. *SIGCSE Bull.* **2003**, *35*, 12–13. [CrossRef]
38. Souza, E.; Moreira, A.; Goulão, M. Deriving architectural models from requirements specifications: A systematic mapping study. *Inf. Softw. Technol.* **2019**, *109*, 26–39. [CrossRef]
39. Mirakhorli, M.; Chen, H.M.; Kazman, R. Mining Big Data for Detecting, Extracting and Recommending Architectural Design Concepts. In Proceedings of the 1st International Workshop on Big Data Software Engineering, Florence, Italy, 23 May 2015; pp. 15–18. [CrossRef]
40. Shahbazian, A.; Kyu Lee, Y.; Le, D.; Brun, Y.; Medvidovic, N. Recovering Architectural Design Decisions. In Proceedings of the 15th International Conference on Software Architecture, Seattle, WA, USA, 30 April–4 May 2018; pp. 95–104. [CrossRef]
41. Sobernig, S.; Zdun, U. Distilling architectural design decisions and their relationships using frequent item-sets. In Proceedings of the 13th Working IEEE/IFIP Conference on Software Architecture, Venice, Italy, 5–8 April 2016; pp. 61–70. [CrossRef]
42. Treude, C.; Storey, M.A. Work item tagging: Communicating concerns in collaborative software development. *IEEE Trans. Softw. Eng.* **2012**, *38*, 19–34. [CrossRef]
43. Storey, M.A.; Ryall, J.; Singer, J.; Myers, D.; Cheng, L.T.; Muller, M. How software developers use tagging to support reminding and refinding. *IEEE Trans. Softw. Eng.* **2009**, *35*, 470–483. [CrossRef]
44. Al-kofahi, J.M.; Tamrawi, A.; Nguyen, T.T.; Nguyen, H.A.; Nguyen, T.N. Fuzzy Set Approach for Automatic Tagging in Evolving Software. In Proceedings of the 2010 IEEE International Conference on Software Maintenance (ICSM), Timisoara, Romania, Timisoara, Romania, 12–18 September 2010; pp. 1–10. [CrossRef]
45. Liu, J.; Zhou, P.; Yang, Z.; Liu, X.; Grundy, J. FastTagRec: Fast tag recommendation for software information sites. *Autom. Softw. Eng.* **2018**, *25*, 675–701. [CrossRef]
46. Mushtaq, H.; Malik, B.H.; Shah, S.A.; Siddique, U.B.; Shahzad, M.; Siddique, I. Implicit and explicit knowledge mining of Crowdsourced communities: Architectural and technology verdicts. *Int. J. Adv. Comput. Sci. Appl.* **2018**, *9*, 105–111. [CrossRef]
47. Parra, E.; Escobar-Avila, J.; Haiduc, S. Automatic tag recommendation for software development video tutorials. In Proceedings of the 2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC), Gothenburg, Sweden, 27 May–3 June 2018; pp. 222–232. [CrossRef]
48. Jovanovic, J.; Bagheri, E.; Cuzzola, J.; Gasevic, D.; Jeremic, Z.; Bashash, R. Automated semantic tagging of textual content. *IT Prof.* **2014**, *16*, 38–46. [CrossRef]
49. Wang, T.; Wang, H.; Yin, G.; Ling, C.X.; Li, X.; Zou, P. Tag recommendation for open source software. *Front. Comput. Sci.* **2014**, *8*, 69–82. [CrossRef]
50. Alqahtani, S.S.; Rilling, J. An Ontology-Based Approach to Automate Tagging of Software Artifacts. In Proceedings of the International Symposium on Empirical Software Engineering and Measurement, Toronto, ON, Canada, 9–10 November 2017; pp. 169–174. [CrossRef]
51. Zhou, P.; Liu, J.; Liu, X.; Yang, Z.; Grundy, J. Is deep learning better than traditional approaches in tag recommendation for software information sites? *Inf. Softw. Technol.* **2019**, *109*, 1–13. [CrossRef]
52. Martins, E.F.; Belém, F.M.; Almeida, J.M.; Gonçalves, M.A. On cold start for associative tag recommendation. *J. Assoc. Inf. Sci. Technol.* **2016**, *67*, 83–105. [CrossRef]
53. Soliman, M.; Galster, M.; Salama, A.R.; Riebisch, M. Architectural knowledge for technology decisions in developer communities: An exploratory study with StackOverflow. In Proceedings of the 13th Working IEEE/IFIP Conference on Software Architecture, Venice, Italy, 5–8 April 2016; pp. 128–133. [CrossRef]
54. Musil, J.; Ekaputra, F.J.; Sabou, M.; Ionescu, T.; Schall, D.; Musil, A.; Biffl, S. Continuous Architectural Knowledge Integration: Making Heterogeneous Architectural Knowledge Available in Large-Scale Organizations. In Proceedings of the IEEE International Conference on Software Architecture, Gothenburg, Sweden, 3–7 April 2017; pp. 189–192. [CrossRef]
55. Lin, B.; Zagalsky, A.E.; Storey, M.A.; Serebrenik, A. Why Developers Are Slacking Off: Understanding How Software Teams Use Slack. In Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion, San Francisco, CA, USA, 27 February–2 March 2016; pp. 333–336. [CrossRef]
56. White, K.; Grierson, H.; Wodehouse, A. Using Slack for Asynchronous Communication in a Global Design Project. In Proceedings of the International Conference on Engineering and Product Design Education, Oslo, Norway, 7–8 September 2017; pp. 346–351.
57. Chatterjee, P.; Damevski, K.; Pollock, L. Exploratory Study of Slack Q&A Chats as a Mining Source for Software Engineering Tools. In Proceedings of the 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR) (ICSE 2019), Montreal, QC, Canada, 26–27 May 2019; MSR 2019 MSR Technical Papers; pp. 490–501.
58. Martínez-García, J.R.; Castillo-Barrera, F.E.; Palacio, R.R.; Borrego, G.; Cuevas-Tello, J.C. Ontology for knowledge condensation to support expertise location in the code phase during software development process. *IET Softw.* **2020**, *14*, 234–241. [CrossRef]
59. Jurafsky, D.; Martin, J.H. *Speech and Language Processing*, 2nd ed.; Prentice-Hall: Upper Saddle River, NJ, USA, 2009.
60. González-López, S.; Borrego, G.; López-López, A.; Morán, A.L. Clasificando conocimiento arquitectónico a través de técnicas de minería de texto. *Komput. Sapiens* **2018**, *1*, 29–33.
61. Vatanen, T.; Väyrynen, J.J.; Virpioja, S. Language Identification of Short Text Segments with N-gram Models. In Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10), Marseille, France, 11–16 May 2010; European Language Resources Association (ELRA): Valletta, Malta, 2010.
62. Devlin, J.; Chang, M.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv* **2018**, arXiv:1810.04805.

63. Wohlin, C.; Runeson, P.; Hst, M.; Ohlsson, M.C.; Regnell, B.; Wessln, A. *Experimentation in Software Engineering*; Springer: Berlin/Heidelberg, Germany, 2012.
64. Davis, F.D. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Q.* **1989**, *13*, 319–340. [CrossRef]