A Multi-agent Approach for Semantic Annotation of Source Code Artefact

Pornpit Wongthongtham[1,*] Udsanee Pakdeetrakulwong[2,*] Suksawat Sae-Lim[2], Worachet Uttha[2], Sutarat Chaonafang[2], Suphitcha chanrueang[2], Supakit Nakpomchin[2], Somkiat Chormuan[2], Naruapon Suwanwijit[2]

[1]School of Information Systems, Curtin University, Western Australia, Australia
[2]Nakhon Pathom Rajabhat University, Nakhon Pathom, Thailand

Abstract

A large volume of software project information is produced in software projects. Manually transforming or mapping them into a semantically rich form for shared understanding is time-consuming, laborious, tedious and prone to error. Hence, it is important to use a systematic approach to automate the knowledge capture of software project information. In this paper, the active Software Engineering Ontology through Multi-agent System (SEOMAS) approach for automated knowledge capture of software project information is proposed. The agents utilise the Software Engineering Ontology (SE Ontology) to capture knowledge from software development artefacts during the daily software development activity. The captured knowledge is populated as new instances in the SE Ontology repository to allow project team members and software agents to access it. It has been demonstrated that the captured knowledge can be put to practical use to clarify any ambiguity in remote communication and to facilitate effective and efficient coordination and knowledge sharing within a software development project.

Keywords: Multi-agent, Ontology, Software Development

1. Introduction

Various types of software project information produced throughout the software development life cycle describe different levels of abstraction and perspectives of a software system. Nonetheless, they are in syntactic format that does not facilitate the understanding of the concepts or meaning. The syntactic representation of software project information produces several issues such as ambiguities, difficulty in data integration, limitation of information retrieval, etc. These problems are more significant in a multi-site software development environment where project team members are dispersed across several locations and face-to-face communication (e.g., formal or information meeting) is limited. As a result, software project information should be transformed into semantic representation to alleviate the aforementioned issues. Some existing approaches have been introduced to capture the semantics of a software project. However, many of them are based on manual approaches or require effort from project team members to carry out additional steps in the knowledge capturing process because they are not integrated in a software development process. The manual capturing of knowledge of software project information is time-consuming, labour-intensive, tedious and error-prone task. In order to tackle these issues, there is the need for a systematic approach that can automatically capture knowledge of software project information and that is seamlessly integrated in a software development process. Once this information has been captured and conceptualised, it can be semantically interlinked with other relevant information. It can then be used to clarify any ambiguity in communication and to enable knowledge sharing among team members. This knowledge is also in machine-readable format which means that it can be understood by software agents. As a result, the agents can make use of this knowledge to assist project teams with their software development activities such as managing project issues, monitoring software project status, suggesting solutions or experts.

2. Materials and methods

2.1. Related works

Knowledge assimilation is the process of capturing and representing the domain-specific knowledge in a formal conceptual model [1]. When large amounts of knowledge need to be captured, an important point is the assimilation of extracted knowledge by means of systematic approaches that do not require great amount of human effort. The captured knowledge can be conceptually represented using the ontological model. In the

* Corresponding author; e-mail: [1*]ponnie.clark@curtin.edu.au, [2*]udsanee@webmail.npru.ac.th

literature, several studies have proposed the use of ontology-based semantic annotation, or semantic annotation for short, to express a formal representation of the resource's content by connecting it to concepts defined in an ontology. In the software development domain, the semantic annotation process is used to tackle problems regarding inappropriate, incomplete, and inconsistent syntactic descriptions of software development artefact properties and qualities. Qiang, Ming and Zhiguang [2] implement a semantic annotation-based software knowledge-sharing space to improve the level of knowledge sharing and facilitate collaborative work among project members. Ontologies are used to create a link between software artefact contents and the abstract knowledge in the space. However, the annotation process is done manually by team members. Zygkostiotis, Dranidis and Kourtesis [3] propose a manual approach to semantically annotate Java source code using domain ontologies for the purpose of software reuse. This approach makes use of the standard annotation facility equipped with the release of Java 5.0 to add metadata to source code elements. In [4], the authors discuss the use of semantic annotations in requirements document templates to support the management and evolution of requirements. The semi-automatic annotation process is based on the conceptualisation captured in the defined software requirement ontology. In [5], the authors propose KnowBench, a semantic-based knowledge management system to assist developers to reuse code or knowledge about solving problems that had been previously addressed in the organisation. The source code is captured by means of both manual and semi-automatic annotation. Damljanovic, Amardeilh and Bontcheva [6] introduce an automatic approach to enhance semantic access to software artefacts (e.g., software document, source code) using the semantic annotation process. This approach is based on the text analysis technique. Taglialatela and Taglino [7] propose an approach to enrich the semantic description of source code by semantically annotating it with a common domain ontology. The goal is to develop a semantic-based search and retrieval of software artefacts in order to facilitate software reuse. The annotation mechanism is based on the analysis of the source code comments which are added by a developer. The annotation process is automatic. However, the quality of the annotation result depends on the quality of the code comments. Tichy, Köerner and Landhäußer [8] propose an approach to automatically create software models from natural language texts with semantic annotation. In [9], the authors present a concept whereby automated software composition is supported by semantic modelling and making use of the annotation process and semantic extensions through knowledge-based techniques.

In the literature review, a significant amount of research contains proposals for semantically annotating software project-related information. A number of works have contributed to source code semantic annotation. However, most of the reviewed approaches are based on manual and semi-automatic annotation. The manual approaches are considered inappropriate because they are tedious, time consuming, and error-prone, especially when a large volume of software artefacts is generated within a project. The semi-automatic annotation approaches can be a good solution; however, they still require human intervention at some annotation level. Some works have proposed the automatic approach. However, most of them are based on text analysis techniques so that they are applicable only to textual artefacts (e.g., software requirement specification, software documents); they are not suitable for the semantic annotation of certain types of artefacts such as source code. In addition, most of the reviewed works regarding semantic annotation approaches in the software engineering domain focus only on semantic annotation which is intended to create semantic descriptions of software resources. Fewer works have paid attention to populating the ontology which is the task of adding new instances of concepts to the ontology. The new instances could be derived from the semantic annotation.

2.2. Conceptual framework

Because software development-related information generated within a software project is in syntactic form, its structure is not conducive to an understanding of the semantics, and therefore may create ambiguities (e.g. incorrect or different interpretations). Source code is considered as the main, centrally located artefact and is critical in software development; therefore, the need to capture its semantics in order to facilitate remote communication, coordination and knowledge sharing is obvious. Hence, given the volume of source code that needs to be dealt with, it is imperative to have a systematic approach for automating semantic annotation and ontology population tasks to ease the burden of manual tasks. This approach should be automated, or should require minimum human effort. In this paper, the SEOMAS framework is proposed. The agent annotates software project information according to the corresponding concepts and then generates new instances which are subsequently populated into the ontology repository. The aforementioned processes can be done by software agents with minimum human intervention. In addition, the utilisation of agents can speed up the process because they are able to act in parallel. To sum up, an ontology-based multi-agent approach will encourage team members to share their knowledge by offering automated and transparent support to semantically capture software project information when they are working on software development process. In this work, the Agent Unified Modelling Language (AUML), a standard representation by FIPA to describe agent communication and protocols [10, 11] has been chosen as an appropriate methodology to capture agent concepts and their interactions. The proposed framework is intended to provide active support to assist software team members

with software engineering knowledge when they are working on multi-site distributed software development projects. It comprises four agent types with brief descriptions of their roles as follows.

- **User agent (UA)** is a mediator between a user and the system. A user employs his/her user agent to perform tasks on his/her behalf.
- **VersionControl agent (VA)** is responsible for managing the version control repository. In this research, this agent focuses on the import of new source code file(s) into the version control repository.
- **Annotation agent (AA)** is responsible for annotating software project information that is imported into the version control repository.
- **Ontology agent (OA)** is responsible for accessing and manipulating the SE Ontology domain and instance knowledge. It also manages the ontology population according to the semantic annotation process.

In order to summarise the semantic annotation and the ontology population process performed by the SEOMAS agents as described above, the whole process is shown graphically in Figure 1.

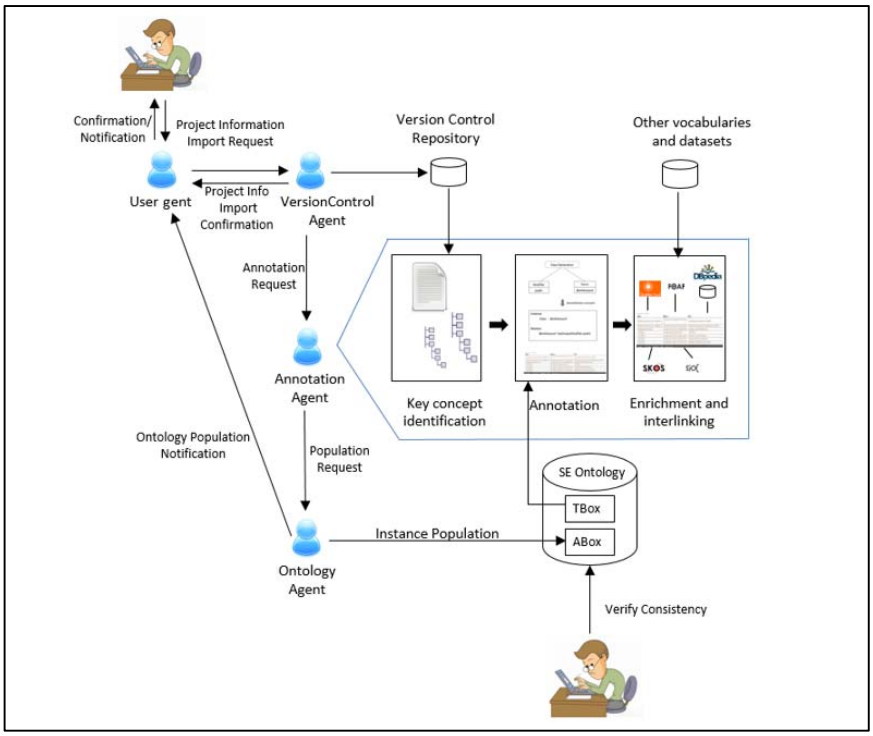

Figure 1 The automated knowledge capture by the SEOMAS approach

Due to space constraint, it is not practical to describe the complete internal aspect model of all agents. In this paper, the internal model of the annotation agent is chosen to be refined as followed.

The annotation agent is mainly responsible for semantically annotating software project information (i.e., source code artefact). It responds to an annotation request from the versioncontrol agent by carrying out a semantic annotation process in order to identify new instances of the Software Engineering Ontological concepts from the source code artefacts. The role associated with this agent is *SemanticAnnotator*. The *SemanticAnnotator* role is to semantically annotate source code artefacts with the appropriate concepts defined in the SE Ontology. The annotation agent fulfils its role with two main behaviours: *IdentifySourceCodeKeyConcepts and AnnotateSourceCode.*

2.2.1 IdentifySourceCodeKeyConcepts behaviours

An incoming request for source code annotation from a user agent is managed by the *IdentifySourceCodeKeyConcepts* behaviour. Two main steps are performed after a request has been identified:

1) Source code retrieval

This step is to retrieve the requested source code file from the version control repository.

2) Key concept identification

This step is to identify the key concepts that are being used in the source code. The source code is analysed and parsed to produce an abstract syntax tree (AST) which is a representation of the abstract syntactic structure of the source code written in a programming language, for example, classes, fields, methods, constructors, parameters as well as in-line comments (e.g., JavaDoc). For source code comments such as author, versions are also identified and parsed in order to obtain a meaningful term-based description of the source code.

2.2.2 AnnotateSourceCode behaviours

After the *IdentifySourceCodeKeyConcepts* behaviour accomplishes its task of key concept identification, the *AnnotateSourceCode* behaviour is initialised as indicated in the pre-condition [*sourcecodeIdentified*]. It annotates the source code elements with the appropriate concepts defined in the SE Ontology and other well-known ontologies and vocabularies, as well as to enrich and to interlink the annotated source code with similar concepts in other datasets (Figure 6-14). This behaviour comprises two main tasks:

1) Source code annotation

The identified source code elements and other software artefacts are assigned software engineering domain concepts that correspond to their semantic description specified in the SE Ontology. Examples of these concepts are Class, Field, Method, Parameter, Modifier, etc. The source code elements that are assigned to those concepts are used to construct statements in the format of RDF/OWL triples which comprise three elements, namely, subject, predicate, and object (subject, predicate, object). The subject part identifies the thing that the statement is about. The predicate part identifies the property or characteristic of the subject that the statement specifies. The object part identifies the value of the property or characteristic [12]. The RDF/OWL statement can be used to semantically describe:

- resource type of the source code elements such as (HelloWorld, type, Class),
- attribute of the source code elements such as (HelloWorld, isMainClass, "True"), or
  to define the relationship between source code elements such as (HelloWorld, hasMethod, main).

2) Enrichment and Interlinking

Other relevant domain ontologies and controlled vocabularies, namely, FOAF, DC, SKOS, SIOC are reused to enrich and interlink the semantic description of the annotated source code. For example, all the source code elements (e.g., class, package, interface, etc.) are annotated with the relationship *rdf:type* as Dublin Core Metadata Initiative (DCMI) Type *'Software'*[1]. If the name of an author is available in the source code, then this relationship is defined in the resulting RDF/OWL triple by using *foaf:name*. The use of existing domain ontologies can enhance the re-useability factor and promote data interoperability [13] as well as help to find semantic similarities with other similar entities described in different semantic repositories. Interlinking also includes the construction of semantic relationships between the annotated source code elements and other entities defined in other dataset on the Web, namely, Wikipedia. In other words, interlinking can enable extensive textual information related to the annotated source code elements or other project-related resources to be retrieved from the Wikipedia website. To extract structured information from Wikipedia and then transform it into RDF, DBpedia has been developed by the research community. The URI according to the format http://dbpedia.org/resource/*Name* corresponds with the URL of the source Wikipedia article, which has the pattern http://en.wikipedia.org/wiki/*Name* [14]. The annotation agent interlinks the annotated source code elements with the corresponding DBpedia entity by using the *owl:sameAs* property. This property is used to specify that the URIs of the annotated elements and those of DBpedia actually refer to the same entities. After the source code has been annotated with the SE Ontology domain concepts as well as enriched and interlinked with other ontologies and controlled vocabularies, the ontology agent inserts the annotated source code into the ontology as new instances.

3. Result and discussion

3.1. Prototype Implementation and Results

The prototypes are used as proof-of-concept experiments of the proposed framework. Java source code is selected for a proof-of-concept implementation. Jena, a Java framework for building Semantic Web applications, is used to make a connection between agents and the SE Ontology and to provide several functionalities such as create, read, modify triples in RDF/OWL. Qdox is used as a parser for the extraction of source code elements. JADE, Java Agent Development Framework [17], which is an agent middleware, is chosen to implement the agent platform and to provide a development framework. JADE is developed from Java and is completely based on the Foundation for Intelligent Physical Agents (FIPA) specifications [18]. Agent Communication Language (ACL) defined by FIPA is chosen as the language of communication between

agents. JADE provides various implemented FIPA-specified interaction protocols such as FIPA-Query, FIPA-Request and so on to construct agent conversation messages. JADE helps to integrate ontologies to represent the application domain through its content reference model [19]. The SE Ontology is registered to this model through the ontological elements, namely, predicates, concepts, and agent actions so that it can be accessed by JADE agents and used as the content of an ACL message.

The SEOMAS agents populates the SE Ontology by inserting new instances derived from the semantic annotation process into the ontology repository. For example, the Java source code, BankAccount.java[2] is semantically annotated and identified as instances of ClassType (Class), Constructor, and Method. In addition, because the BankAccount instance is enriched with the Software concept of Dublin Core Metadata Initiative (DCMI), so it is an instance of a Software class as well. The annotated source code elements are also enriched by interlinking them with other relevant data source in order to provide an extended view of them. The annotated Java class BankAccount is interlinked with the DBpedia dataset named http://dbpedia.org/page/Java_class_file. The link is created by using an owl:sameAs property to specifiy that the URI of the annotated element and that of the DBpedia dataset refer to the same resource. As a consequence, additional information about the Java class file can be obtained or queried from DBpedia website (http://dbpedia.org). Figure 2 depicts the instances and relationships of class BankAccount populated in the SE Ontology. The graph is generated by the OntoGraf plug-in.

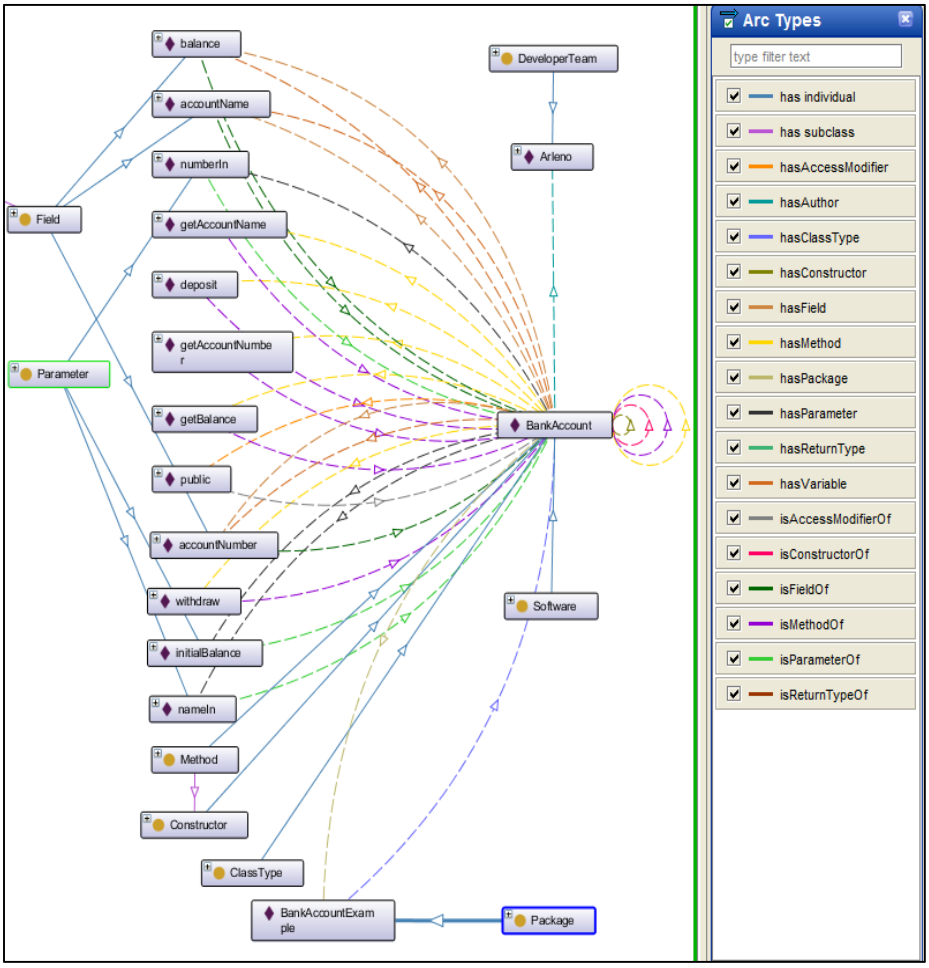

Figure 2 OntoGraf presentation of the BankAccount class instance.

The practical uses of the SEOMAS approach for evaluation purpose are based on a vehicle registration system being developed by a multi-site team located across various sites. Software developers communicate, coordinate, manage and share software development project information captured in the SE Ontology through the collaborative agents during the bug resolution process. The vehicle registration system is being developed in a multi-site software development environment. Software developers are dispersed across four sites, namely, Perth, Bangalore, Dublin, and Shanghai. All the Java classes are annotated and populated in the SE Ontology repository

[2] http://homepages.uel.ac.uk/A.Kans/pm1/week3.pdf

by means of the SEOMAS approach. They are also semantically interlinked with other relevant software project information captured in the SE ontology, e.g., project description, project team information, source code commit, bug reports, etc. In other words, software project-related software information will not appear in isolation, but will be part of a large group of related information.

When Alex, a developer, requests a change to the method getMakeYear of the Vehicle interface by modifying a method return type through the SEOMAS platform, the ontology agent can make him aware of the potential impact to other software components. In object-oriented system development, a subclass is dependent on the super class that it inherits or the interface that it implements; therefore, a change in the super class or the interface will impact on its subclass. Figure 3 presents the recommendation of potentially affected artefacts sent to Alex. MotorBike and Car class are suggested as affected classes when the getMakeYear method is modified because they implement the Vehicle interface. VehicleRegistration is also suggested as the affected class because it is the main call which invokes either Car or MotorBike class. Figure 4 illustrates messages sent to notify the authors of those potentially affected artefacts to be aware of the change in the Vehicle interface. In this example, the manipulation platform does not only assist team members to manage the software project information captured in the Software Engineering Ontology, but it also provides useful and precise situational knowledge regarding the change impact analysis to improve team members' awareness and alert them to the need for coordination.

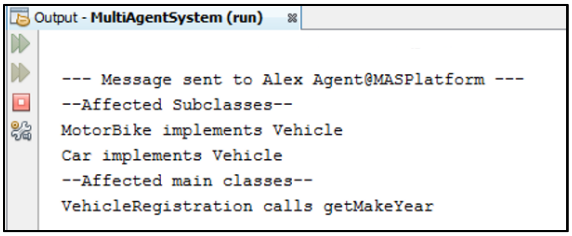

Figure 3 Recommendation of potentially affected artefacts

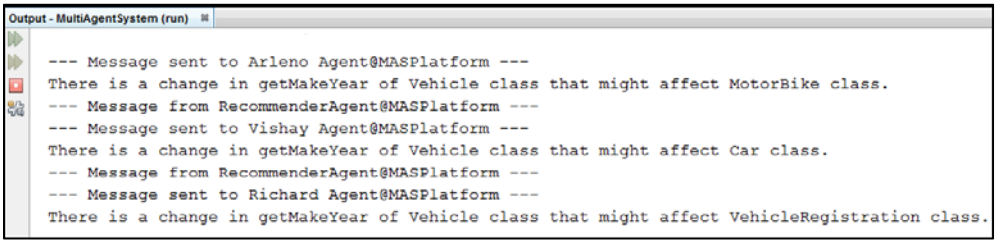

Figure 4 Messages to notify the authors of potentially affected artefacts

3.2. Evaluation

In this section, the evaluation of automated knowledge capture of source code artefacts is demonstrated through the case study derived from [20]. Table 1 describes the bug resolution process mentioned in the case study when the SEOMAS framework is not utilised.

Table 1 Bug resolution process described in [20]

No.	Date	Actor	Actions
1.	3 Aug 2009	Richard@ Perth	Richard filed a bug report in the project issue tracking system with high priority.
2.	4 Aug 2009	Richard@ Perth	Richard filed another bug report with an urgent request hoping to increase its priority and draw greater attention from developers.
3.	4 Aug 2009	Vishay@ Bangalore	Vishay came up with a quick fix and added a comment at the end of the report, putting the report into the status of "re-evaluation pending".
4.	11 Aug 2009	Arleno@ Shanghai	Arleno filed a duplicate bug which was soon recognized as a repeated report two days later.
5.	15 Aug 2009	Arleno@ Shanghai	Arleno discussed with his team members and supervisor, who added comments to the report and directed their concerns back to the Bangalore Lab
6.	17 Aug 2009	Larry@ Bangalore	Larry provided another bug fix solution
7.	17 Aug 2009	Michael@ Dublin	Michael picked up the fix and pointed out that Larry's fix might produce deadlocks in another related component and suggested reverting back to the first fix.

No.	Date	Actor	Actions
8.	18 Aug 2009	Larry@ Bangalore	Larry fixed the bug based on Michael's instruction
9.	24 Aug 2009	Michael@ Dublin	Michael checked the fix and marked the bug report status as "resolved" and closed the bug.
10.	24 Aug 2009	Lisa@ Shanghai	Lisa suggested that the latest fix resulted in a connection timeout.
11.	25 Aug 2009	Larry@ Bangalore	Larry asked Lisa to explain the affected component
12.	25 Aug 2009	Michael@ Dublin	Michael fixed the bug, and explained his fix.
13.	29 Aug 2009	Richard@Perth	Richard closed the bug as “resolved”.
Total	27 days	6 actors	13 actions

From Table 1, it can be seen that even though the bug was not too complicated and needed only a simple modification to fix the problem, it took 27 days to finalise the resolution which might cause a project delay. Difficulties arose from the lack of common semantics. First, the information related to the bug was dispersed among several software repositories with no links to indicate that they were related to each other. Therefore, the same bug report was filed repeatedly. Second, the bug was initially fixed by developers who had no expertise in this area, resulting in several iterations of invalid fixes. Without the knowledge support to match the bug with the expert, the bug-fixing time could be prolonged. Finally, the inadequate sharing of project information and knowledge, such as the dependencies among software components, can delay the bug fixing. As discussed above, Larry did not know what the affected component was, so he needed someone to clarify this information because there was no available and explicit reference that he could access.

In order to address the abovementioned issues, software project information (e.g., source code, bug reports, communication threads) should be captured so that software development knowledge becomes conceptualised, organised, and can be semantically linked among related knowledge. The SEOMAS framework can help to automate knowledge capture process by means of the semantic annotation and the ontology population tasks which are seamlessly integrated into the software development process (e.g., version control). Once this software project information has been captured and integrated, it is available for sharing among software project teams to facilitate software development activities or to address project issues by, for example, assisting with a bug resolution process as described in Table 2.

Table 2 Bug resolution process with supporting from the SEOMAS approach

No.	Date	Actor	Actor Actions	Agent	Agent Actions
1.				VersionControl agent Annotation agent Ontology agent	1. The versioncontrol agent imported a new software project information file into the version control repository. 2. The annotation agent annotated software development artefacts to identify new instances. 3. The ontology agent populated the SE Ontology with new instances.
2.	3 Aug 2009	Richard@ Perth	Before filing a bug report, Richard checked whether the bug had been reported through the query platform	Richard’s user agent	Richard’s user agent sent a query request to the ontology agent
3.	3 Aug 2009			Ontology agent	The ontology agent retrieved existing bug reports related to the problem class and sent them back to the user agent.
4.	3 Aug 2009	Richard@ Perth	Richard filed a new bug report with high priority.		
5.	3 Aug 2009			Ontology agent	The ontology agent 1. identified Michael@ Dublin as the most likely person to be able to solve the new filed bug report; 2. attached Michael@

No.	Date	Actor	Actor Actions	Agent	Agent Actions
					Dublin as the potential fixer into the bug report; 3. sent a message to notify Michael@Dublin to draw his attention to the new bug report that may need his expertise to resolve.
6.	3 Aug 2009	Michael@ Dublin	Michael received a message to notify him of a new bug report.	Michael's user agent	Michael's user agent translates a message from the ontology agent and display to Michael
7.	3 Aug 2009			Ontology agent	The ontology agent provided Michael with: 1. information about the problem class and its related software compo-nents; and 2. history of all previous bugs reported to the problem class and how they were fixed.
8.	4 Aug 2009	Michael@ Dublin	1. Michael fixed the bug based on information from the ontology agent. 2. Michael marked the bug report status as "resolved".		
9.	4 Aug 2009			Ontology agent	The ontology agent sent a message to notify Richard that the status of the bug had been changed to "resolved".
10.	5 Aug 2009	Richard@ Perth	Richard read the message, verified the resolution, and then closed the bug.		
Total	3 days	2 actors	6 actions by real user	6 agents	12 actions by agents
Total number of actions			18 actions		

As demonstrated in Table 2, the bug resolution process involves bug understanding, bug triage, and bug fixing as well as additional steps to avoid the recurrence of similar bugs in the future. It is considered as one of the most complex activities particularly in a multi-site distributed software development project because it requires significant collaboration of information from various sources (e.g. bug reports, software components, forum discussions) and various stakeholders. From the comparison provided in Table 1 and Table 2, it is evident that the SEOMAS framework can help multi-site distributed software development teams to resolve the bug issues by improving the effectiveness and the efficiency of communication and coordination as well as enabling knowledge sharing as follows.

1. Before filing a bug report, the ontology agent can help a software developer to locate related bug reports based on their associated concepts defined in the SE Ontology and its instances. Then s/he can view a list of existing bugs reported to a particular class and determine whether the current bug is a duplicate. In this case, duplicated bug reports could be identified early and avoided. This can reduce the unnecessary information overload and considerably reduce confusion as well as help to prevent tedious conflict.

2. After a bug has been filed, the ontology agent can recommend a person who is most likely able to resolve the bug issue, and sends a message to alert him about the new bug report that potentially needs his expertise to resolve. This can help to match a bug to a potential fixer or consultant to avoid the inadequate fixes from someone without expertise with this particular bug.

3. When the bug is being fixed, the ontology agent can provide relevant information that is necessary for fixing the bug such as the history of bugs reported to the problem class and their resolution, or related software components and their owners. Then the developer can know what dependencies exist and check with relevant people before making a change to prevent unintended side effects from a change made.

4. When a developer makes a change to the source code, he is also proactively informed about the components that potentially may be affected by a change. This can reduce unintended side effects from the impact of the bug fixing, and avoid future problems.

5. The ontology agent sends a message to notify the bug reporter as soon as the bug status is changed to "resolved". The reporter then knows that the issue that he reported has been resolved, so he can verify the

solution. Once he is satisfied with the solution, the bug report can be closed. The SEOMAS agents can improve real-time awareness of team members and enable efficient coordination without overloading them.

Parameters for Efficiency Measurement

In the above scenarios, the efficiency of bug resolution by utilising the SEOMAS framework is measured by three parameters, namely, time to complete the task, the number of team members involving in the bug resolution, and the number of team members' actions.

1. Time to complete the task

Without the support of SEOMAS, the estimated time that would be taken to resolve a single bug issue is 27 days. However, when SEOMAS is utilised, it takes only three days to fix the same bug. This significant reduction in time is due to the fact that source code artefacts and other software-related project information (e.g., bug reports, archived communications) are all captured and can be integrated to generate interconnections among them. The ontology agent can utilise this interlinked knowledge space to deliver useful and timely information to development teams. The delivered information is also based on previous historical data in the software project. Information such as a match between a bug and expert, related software components and related bugs, can assist developers to diagnose and fix the bug more effectively and efficiently. Therefore, the response time required to correct failures to complete the bug resolution task is reduced.

2. The number of team members involving in the bug resolution

As seen in Table 1, six team members are involved in the bug resolution process. Even though the bug is not a complicated one and may require only a simple modification by an expert, without utilising the SEOMAS platform, it goes around across multiple sites which leads to several iterations of inappropriate fixes from someone without expertise in fixing this kind of bug; moreover it unnecessarily prolongs the bug resolution process. With the support from the SEOMAS framework, fewer team members are involved in the bug resolution process because the number of people reporting duplicate bugs can be reduced and the bug can be directly assigned to the appropriate team member who has the expertise required to resolve the issue instead of going around to several people.

3. The number of team members' actions

In the bug resolution scenario without support from the SEOMAS framework, it can be seen that there are a number of unnecessary actions from the team members. For example, personnel are filing duplicated bugs or iteratively fixing the same bug. This is because the information and interactions which relate to the bug are stored in various software artefacts without links between them. When the SEOMAS platform is utilised, the source code is annotated using meta-data that is semantically rich to enable it to be interlinked with other relevant information. Hence, the development artefacts are all related, not independent. Therefore, the ontology agent can help to locate related problems and deliver them to the team members to prevent the same bugs from being reported multiple times. Therefore, the number of team members' actions is decreased from thirteen actions to six actions.

From Table 2, it can be seen that the total number of actions with SEOMAS support is higher than without the SEOMAS support in Table 1. This is because several actions are performed by the SEOMAS agents to achieve their goal and to enable team members to perform their tasks more efficiently. These actions include the translation between team members and their user agents, identifying expert and recommending useful information about related software components, sending messages sent to relevant team members. However, these actions are autonomously performed by the agents and do not impact on team members' performance.

4. Conclusions and Future work

In this paper, the SEOMAS framework for semantic annotation to automate knowledge capture of source code artefacts is proposed. The agents utilise the SE Ontology to capture knowledge from software development artefacts during the daily software development activity. The captured knowledge is populated as new instances in the SE Ontology repository to allow project team members and software agents to access it. In the future, the SEOMAS framework can be extended to capture the semantics of other types of software artefacts. The extension can cover the semantic annotation of both structured information (e.g., UML diagrams, issue tracking, commit data) and unstructured information (e.g., requirement documents, bug reports, forum discussion).

References

[1] D. E. Forbes, A Framework for Assistive Communications Technology in Cross-Cultural Healthcare. School of Information Systems, Curtin Business School, Curtin University, 2013.

[2] L. Qiang, C. Ming, and W. Zhiguang. A semantic annotation based software knowledges sharing space. **Network and Parallel Computing**, 2008. NPC 2008. IFIP International Conference on, doi: 10.1109/npc.2008.55.

[3] Z. Zygkostiotis, D. Dranidis, and D. Kourtesis. Semantic annotation, publication, and discovery of Java software components: an integrated approach. In Proceedings of the 2nd Workshop on Artificial Intelligence Techniques in Software Engineering (AISEW 2009), Thessaloniki, Greece: CEUR-WS. org. 168-178.

[4] L. d. O. Arantes, and R. d. A. Falbo. An infrastructure for managing semantic documents. In Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2010 14th IEEE International, October 25-29, 2010; 235-244. IEEE.

[5] D. Panagiotou, and G. Mentzas, Leveraging software reuse with knowledge management in software development. **International Journal of Software Engineering and Knowledge Engineering**. 2011; 21(5);693-723; 2011.

[6] D. Damljanovic, F. Amardeilh, and K. Bontcheva, CA manager framework: creating customised workflows for ontology population and semantic annotation. In Proceedings of the Fifth International Conference on Knowledge Capture, Redondo Beach, California, USA, 2009, 177-178.

[7] A. Taglialatela, and F. Taglino, A semantics-based approach to software reuse. In the Fifth Interop-Vlab.It Workshop on Complexity of Systems, Complexity of Interoperability in conjunction with itAIS 2012., Rome, Italy.

[8] W. F. Tichy, S. J. Köerner, and M. Landhäußer. Creating software models with semantic annotation. In Proceedings of the Third Workshop on Exploiting Semantic Annotations in Information Retrieval, Toronto, ON, Canada, 2010; 17-18.

[9] P. Graubmann, and M. Roshchin. Semantic annotation of software components. In Software Engineering and Advanced Applications, 2006. SEAA '06. 32nd EUROMICRO Conference on, 2006; 46-53.

[10] M.-P. Huget, and J. Odell. Representing agent interaction protocols with agent UML. Agent-Oriented Software Engineering V: 5th International Workshop, AOSE 2004, New York, NY, USA, July 19, 2004. Revised Selected Papers, J. Odell, P. Giorgini and J. P. Müller, eds., pp. 16-30, Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

[11] M.-P. Huget, J. Odell, and B. Bauer. The AUML approach. **Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook,** F. Bergenti, M.-P. Gleizes and F. Zambonelli, eds., 237-257, Boston, MA: Springer US, 2004.

[12] D. Beckett, and B. McBride. RDF/XML syntax specification (revised). **W3C recommendation.** 2004; 10.

[13] J. Ashraf, O. K. Hussain, and F. K. Hussain. A Framework for Measuring Ontology Usage on the Web. **The Computer Journal. 2012.**

[14] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia - A crystallization point for the Web of Data. **Web Semantics: Science, Services and Agents on the World Wide Web.** 2009;7(3): 154-165.

[15] M. Robillard, R. Walker, and T. Zimmermann. Recommendation systems for software engineering. **Software, IEEE**. 2010; 27(4): 80-86.

[16] "QDox," October 11, 2014; http://qdox.codehaus.org.

[17] Y. Liao, M. Lezoche, H. Panetto, and N. Boudjlida. Semantic annotation model definition for systems interoperability. On the Move to Meaningful Internet Systems: OTM 2011 Workshops. 2011; 61-70.

[18] F. L. Bellifemine, G. Caire, and D. Greenwood. **Developing multi-agent systems with JADE**: John Wiley & Sons; 2007.

[19] G. Caire, and D. Cabanillas. Jade tutorial application – Defined content languages and ontologies.12 July 2016; http://jade.tilab.com/doc/tutorials/CLOntoSupport.pdf.

[20] P. Wongthongtham, T. Dillon, and E. Chang. State of the Art of Community-Driven SE Ontology Evolution. In Proceedings of Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on, doi: 10.1109/DASC.2011.170.