Construction and Application of the User Behavior Knowledge Graph in Software Platforms

Fuhua Shang, Qiuyu Ding, Ruishan Du*, Maojun Cao and Huanyu Chen

School of Computer and Information Technology, Northeast Petroleum University, China
E-mail: ruishan_du@163.com
**Corresponding Author*

Received 13 November 2020; Accepted 11 December 2020;
Publication 09 March 2021

Abstract

The analysis of user behavior provides a large amount of useful information. After being extracted, this information is called user knowledge. User knowledge plays a guiding role in implementing user-centric updates for software platforms. A good representation and application of user knowledge can accelerate the development of a software platform and improve its quality. This paper aims to further the utilization of user knowledge by mining the user knowledge that is implicit in user behavior and then constructing a knowledge graph of this behavior. First, the association between a software bug and a software component is mined from the user knowledge. Then, the knowledge entity extraction and relationship extraction are performed from the development code and the user behavior. Finally, the knowledge is stored in the graph database, from which it can be visually retrieved. Relevant experiments on CIFLog, an integrated logging processing software platform, have proved the effectiveness of this research. Constructing a user behavior knowledge graph can improve the utilization of user knowledge as well as the quality of software platform development.

Journal of Web Engineering, Vol. 20_2, 387–412.
doi: 10.13052/jwe1540-9589.2027

Keywords: User behavior knowledge graph, user knowledge extraction, graph database, knowledge graph construction.

1 Introduction

During the process of software development, knowledge flows between developers, projects, and users [1]. After the software is released, its users often play a role in promoting the improvement of the software platform by guiding software updates [2]. At the same time, as the development scale of software continues to expand, software requirements increase and become more complex. If software developers, particularly developers of large-scale platforms, want to improve the user experience, meet the needs of users, and expand the user community, they need to deepen their understanding of users [3]. Under this premise, knowledge extraction and the proper use of behavioral information generated at the user level is a better method for improving the quality and efficiency of software development, and for making the software platform more user-centric.

However, during the development of the large-scale software platform Integrated Logging Platform CIFLog [4], the project team found that the platform was suffering from problems in the areas of knowledge extraction and utilization of user behavior information, which affected the process of software maintenance. Through investigation, we find that the current research on the application of user behavior information can be divided into three types: user intention analysis based on user behavior [5–7], personalized recommendation based on user behavior [8–10], and credibility analysis of user behavior [11–13]. Furthermore, while the research on software knowledge representation in the field of software engineering is still focused on user demand representation [14–16], software development knowledge modeling [17, 18], and software development process knowledge management [19, 20]. However, research on user behavior knowledge extraction and representation is still lacking.

The knowledge included at the user level is divided into two types. The first is the user's active feedback information. Such feedback can go directly to the developers. Communicating user expectations for the software and hearing what they have to say about bugs can quickly improve the software development project. For this kind of information, it is generally easy to build rules, so artificial knowledge is usually adopted to build rules, or modify and perfect on the software platform directly. The second type is tacit knowledge, which requires mining user behavior information in order to

extract knowledge such as potential associations between components, bugs that have not been fixed, and the common usage habits of users. This article focuses on tacit knowledge.

In summary, in order to increase the acquisition of user knowledge and mine user-generated knowledge, and to improve the developer's utilization of such knowledge for the purpose of improving the software platform, this paper proposes a user knowledge mining method. This method also applies knowledge graph technology, which can help developers intuitively search, browse, and manage user knowledge. For this article, user knowledge refers to software platform-related knowledge that has a greater value density, including explicit knowledge and tacit knowledge. This research was carried out on CIFLog, and the effectiveness of the work in this paper was demonstrated through application examples.

The second section of this paper analyzes the problems and difficulties of user knowledge acquisition and representation and selects relevant problem-solving technologies. Related solutions are provided in Section 3. User knowledge mining based on user behavior information is discussed in Section 4. The construction scheme and application of a user behavior knowledge graph is explained in Section 5. The conclusions and future work of this paper appear in Section 6.

2 Related Work

The research on the application of user behavior information can be divided into three areas: user intention analysis based on user behavior, personalized recommendation based on user behavior, and credibility analysis of user behavior.

The related research on user intention analysis based on user behavior is as follows: Kang constructed two-phase reanalysis model for better understanding user intention [5]. Xin conducted related research on the relationship between user behavior intent in mobile applications and services, and clarified selected models related to user intent research [6]. Burkhardt introduced a new approach for classifying users' search intentions in big data applications [7].

The related research of personalized recommendation based on user behavior is as follows: Priya used consumer behavior analysis to generate personalized ontology system [8]. Lei used causal association rule and collaborative filtering based on user behavior to improve personalized

recommendation [9]. Shaymaa analyzed user online behavior information to give them personalized recommendation [10].

Relevant research on user behavior credibility analysis is as follows: Singal estimating the trustworthiness of websites by evaluating the actual behavioral metrics of the users [11]. Alrubaian evaluated the behavioral credibility of users on Twitter by build a new model [12]. Zhou proposed a shilling behavior detection structure based on abnormal group user findings and rating time series analysis [13].

Despite all of the research on user behavior analysis and application, there is still a lack of analysis of user behavior for software platforms. The research on software knowledge representation in the field of software engineering can be divided into three areas: user demand representation [14–16], software development knowledge modeling [17, 18], and software development process knowledge management (KM) [19, 20].

The research on user demand expression is as follows: Meng Xiangwu summarized the research progress of mobile user demand acquisition technology [14]. Wang Chen used multi-dimensional user demand acquisition and representation using the ontology technology [15]. Wang et al. built a distributed database to deal with issues related to the changing needs of users [16].

The research on software development knowledge modeling is as follows: Maouche's research was multi-scale knowledge modeling towards software engineering [17]. Ouriques explored the management of corporate knowledge by the KM strategy adopted in agile software development [18].

The research on knowledge management in software development process is as follows: Vasconcelos adopted software development knowledge representation and management to software evolution [19]. Goncalves built demand model to manage software knowledge [20].

The above research has leveraged knowledge about the software development process in order to improve the process. However, there is no existing research that aims to improve and upgrade software platforms by extracting user knowledge based on user behavior, and then applying it to the software platform. This article proposes to extract implicit user knowledge based on the behavior information generated by the software platform, and to represent such knowledge by constructing a user behavior knowledge graph, in order to improve the utilization rate of user knowledge for software developers. At the same time, the knowledge graph of user behavior makes it easier for developers to retrieve and use the software platform knowledge.

3 Problem Analysis

User knowledge is divided into explicit knowledge and tacit knowledge. Of the two, explicit knowledge is easier to understand, while tacit knowledge requires analysis and processing of user behavior information in order to obtain knowledge with greater value density. Through the analysis of user behavior, the tacit knowledge that we can obtain includes the association relationship between software platform components such as functional modules, buttons and interfaces, the bugs not found during software testing, and, through the association relationship, obtaining common operations for more users.

The generation of user behavior is based on the user's interaction with the software platform. Different users on the same software platform show different behaviors due to differences in their behavior habits. Even the same user often exhibits different behaviors under the influence of external factors. Therefore, it is necessary to obtain the common information mined from user behavior, and it is very important to select the appropriate analysis algorithm.

At the same time, user behavior information and software platform knowledge entities are complex and diverse, and the relationship between them is complicated. Therefore, it is necessary to use appropriate models to represent knowledge entities and relationships. The characteristics of the property graph [21] model are suitable for our requirements for user knowledge and software platform knowledge representation. These characteristics consist of nodes, directed edges, and attributes; each node has a unique identifier that corresponds to the defined entity type; each vertex has a key-value pair to represent the attribute; nodes and nodes are connected by edges; each side has a unique identifier, each edge must have a head node and a tail node, and each side has key-value pairs to define attribute combinations.

Based on the attribute graph model, different nodes in the software knowledge graph for user knowledge constructed in this paper correspond to different knowledge entities; the directed edges in the graph represent semantic associations; and the key-value pairs in each software knowledge entity correspond to the entity nodes.

The attribute graph model is the underlying implementation model of many graph databases. The top four graph databases [22] in the DB-Engines Ranking list are Neo4j, OrientDB, Titan, and Virtuoso. After comparing these four databases, this study selected the Neo4j database due to its maturity and its support of the attribute graph model and rich query language as the underlying storage of the software knowledge graph for user knowledge.

When constructing a user behavior knowledge graph based on user behavior information, four problems need to be solved.

(1) User knowledge entity extraction. The user entity includes a variety of knowledge entities, including functional modules, buttons and interfaces. The types of knowledge entities that graph user behavior information to the underlying code include classes and methods. For different types of knowledge entities, their forms and characteristics are also different.
(2) Software component association rule mining. Based on user behavior information, the association rules between components in the software platform are mined. Because the component entities included in the software platform are complex and diverse, and because each user's familiarity with the software platform is different, the value of the user behavior information is also different.
(3) Establishment of the relationship between the user knowledge entities and the software knowledge entities. Software platform development code determines the relationships between the software knowledge entities, and the user behavior information generated by the users of the software platform also implies the relationship between the components.
(4) User knowledge retrieval and display. This involves determining how to use the user behavior knowledge graph to provide software developers with the knowledge they need.

4 Solution

The overall framework for constructing a user behavior knowledge graph is shown in Figure 1.

First, an appropriate algorithm is selected to obtain two types of user knowledge: software component associations and bugs based on user behavior information. After that, the graph is constructed.

The data source of the graph is the user knowledge and software platform development source code obtained based on the user behavior information generated by the software platform. First, the above two types of user knowledge are extracted from their respective knowledge graphs. This process is called software knowledge extraction. Then, according to the rules obtained from mining, association relationships are established among knowledge entities, and the independent knowledge graphs from these two resources are organized to form the user behavior knowledge graph. Finally, the knowledge

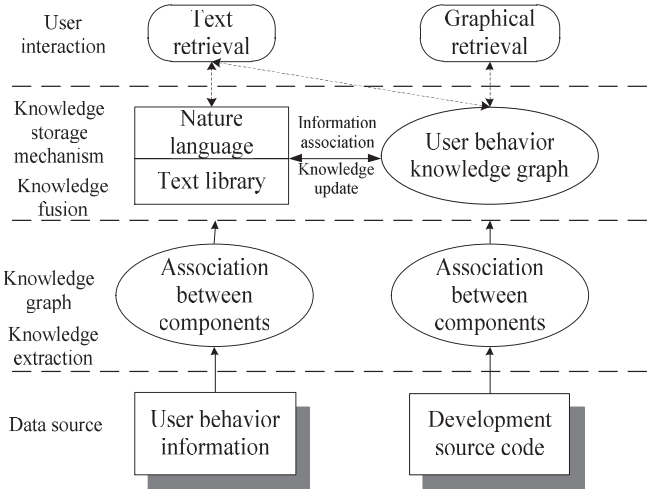

Figure 1 User behavior knowledge graph framework.

retrieval and presentation mechanism is established to meet the developer's retrieval requirements. This paper focuses on the graphical retrieval mechanism, which provides the required knowledge and related knowledge quickly and visually.

Four aspects are elaborated on in Section 5: user knowledge mining, software knowledge entity extraction, software knowledge entity association establishment, and software knowledge retrieval and display.

5 User Knowledge Mining Based on User Behavior Information

5.1 Knowledge Mining Algorithm

This software platform used in this paper is CIFLog. Therefore, the user knowledge mining based on the user behavior information generated by CIFLog is mainly used to mine the association rules between the components in the software platform. The main goal of knowledge extraction in this paper is to mine the relationships between components, including the relationship between modules, the relationship between modules, the relationship between buttons, and the relationship between modules. The association rules are divided into two categories according to the application environment: numerical types and Boolean. In this paper, the association rules between software platform components are Boolean.

The most important step in the process of mining association rules between software platform components is to find frequent item sets with confidence greater than the minimum confidence threshold in the user behavior information. First, find all frequent items whose support degree meets the minimum support threshold. Then, find a set of items with a confidence greater than the minimum confidence threshold. After performing these two steps, strong associations between software platform components can be found.

Common association rule algorithms include Apriori algorithm and FP-growth algorithm. By comparing the two algorithms, this paper chooses FP-growth algorithm for mining association rules among software platform components. Because Apriori algorithm has the following disadvantages: Excessive number of scans to the database may generate a large number of candidate item sets. However, the tree structure used by FP-growth algorithm makes it not produce a large number of candidate sets in the face of huge user behavior data, and it does not cause huge I/O burden by repeatedly scanning the user behavior information database. It only needs to scan the database twice. Under the background of this paper, FP-growth algorithm is more efficient, so this paper chooses FP-growth algorithm to mine Boolean association rules from the database.

5.2 Knowledge Mining Process Based on User Behavior

First, the behavior information of different users is processed. Each user has a table in the database. The table id is the time when the user uses the software (it is a valid reference content for subsequent reasoning), and the table content is the behavior information of the software user. The user exits the software as a mark and deletes the invalid operation by repeatedly clicking a certain button or clicking on a blank space, without getting any feedback from the software. These are called invalid operations.

After pre-processing, the association rules have been mined based on the obtained information. Let $P = \{p_1, p_2, \ldots, p_n\}$ be a collection of all components in the software platform. Data set D is a collection of database transactions storing user behavior information, and any transaction T is a collection of some components in the software platform, $T \subseteq P$. Let A be a set of components that make up the software platform, then transaction T contains A as $A \subseteq T$. The association rules are expressed in the form of implications. For example, in the software platform, button A and button B are related to the implied expression of $A \Rightarrow B$, where $A \subseteq P$, $B \subseteq P$, and $A \cap B = \phi$.

Two important concepts in relation mining are support and confidence. The support degree s is the ratio of the union of the software platform component item set A and the software platform component item set B in the database transaction set D, which stores the user behavior information:

$$s = P(A \cup B) = \frac{count(A \cup B)}{|D|} \tag{1}$$

Confidence c is the ratio of transactions that also contain B in transactions that contain both A in A and transactions that contain only A:

$$c = P(B|A) = \frac{support(A \cup B)}{support(A)} = \frac{count(A \cup B)}{count(A)} \tag{2}$$

In relation mining, a set of items whose support degree is greater than or equal to the minimum support level threshold is called a frequent (or large) item set. First, the FP-Growth algorithm is used to traverse the database transaction set, the frequent item sets are obtained, and then arranged in descending order according to the support degree to form a frequent item header table. Then, the transaction set is traversed again, the FP-tree is created, the pointer in the header table is filled, and the mining head is drilled. The initial suffix pattern in the table constructs a conditional pattern base consisting of a set of prefix paths, then establishes a conditional pattern tree, and then recursively mines all frequent K-item sets [23]. The association rule mining finds the top-K term in the frequent item sets with confidence greater than the minimum confidence threshold. This process is also the mining process of strong association rules. The mined association rules will be stored and represented in the form of knowledge for use by developers.

The confidence between the strong association rules $A \Rightarrow B$ items is defined as follows [24]:

$$confidence(A \Rightarrow B) = \frac{\eta(A \cup B)}{\eta(A)} \tag{3}$$

where confidence $(A \Rightarrow B)$ is the confidence between the $A \rightarrow B$ item sets, and $\eta(A)$ is the number of transactions containing A in the user behavior information database. The higher the confidence, the greater the likelihood that B will appear in A.

The degree of lift reflects the relevance of the pre- and post-rules and excludes rules that are unrelated or less relevant. The degree of elevation

Table 1 User knowledge entity association information

Strong association	Constituent elements = {Relationship between the same entity types, Relationship between different entity types}
Relationship between the same entity	Constituent elements = {(Function module, Function module), (Interface, Interface), (Button, Button)}
Relationship between different entity types	Constituent elements = {(Function module, Interface), (Function module, Button), (Interface, Button)}

between the sets of items A and B indicates the extent to which the item set A is "upgraded" to B. This is calculated using the following formula [24]:

$$lift(A, B) = \frac{P(A \cup B)}{P(A)P(B)} = \frac{confidence(A \Rightarrow B)}{support(B)} \tag{4}$$

If $\text{lift} > 1$, the occurrence of item sets A and B is positively correlated, and the appearance of A has a positive effect on the appearance of B; if $\text{lift} = 1$, A and B are independent, and there is no correlation; if $\text{lift} < 1$, then A and the occurrence of B are negatively correlated.

After performing the above steps, it is necessary to combine the information mining and knowledge screening. Based on different rules formed by frequent pattern analysis, the rules of users are discovered at different levels, and this knowledge is then stored. The associated information between the mined user knowledge entities is shown in Table 1.

6 Knowledge Entity Extraction

6.1 Design and Construction of the Schema Layer

Before the knowledge can be extracted, the schema layer needs to be constructed. The knowledge extraction is then performed according to the defined schema layer. This process is equivalent to modeling the overall knowledge framework. Therefore, the schema layer needs to define and constrain the relationship between classes and classes, that is, define and constrain the concepts and relationships between the concepts contained in the knowledge graph.

This paper constructs the schema layer of the user behavior knowledge graph and defines four basic classes: function modules, interfaces, buttons and bugs. The properties for the base class are defined as follows:

(1) The properties of the function module include Chinese name, English name, interface, function, pre-module, and post-module.

(2) The properties of the interface include Chinese name, English name, module, button, and function.
(3) The properties of the button include Chinese name, English name, the interface to which it belongs, response event, front button, and back button.
(4) The properties of the bug include the number, the prompt information, the trigger method, and the function.

According to the definition of the four categories, this paper constructs three semantic relationships: e_HasInterface, e_HasButton, and e_HasBug. An e_HasInterface relationship exists between the function module and the interface. An e_HasButton relationship exists between the interface and the button. There are e_HasBug relationships between the function module, the interface, and the button.

Note that the entity referred to by the e_HasBug relationship here refers to the location where the bug occurred. The location of the bug should be in the implementation of the interface and buttons. However, from the user or non-professional perspective, it seems that there is a bug in the interface or button, so in this article, the e_HasBug relationship is defined in the most intuitive way from the user's point of view. The constructed schema layer is shown in Figure 2.

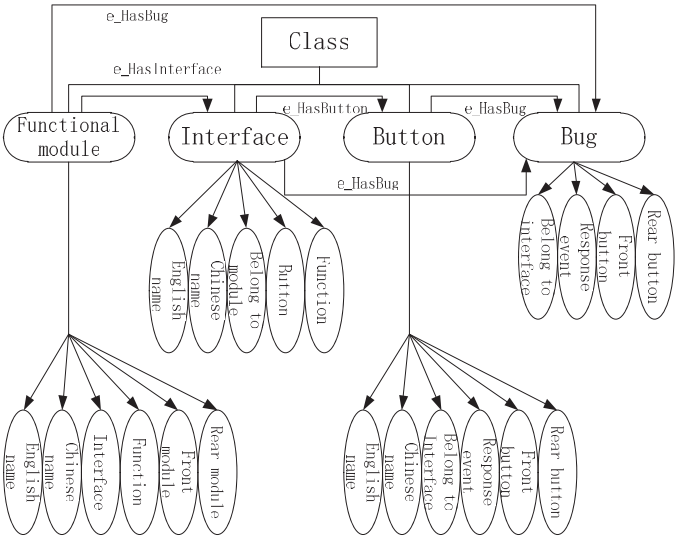

Figure 2 User behavior knowledge graph framework.

6.2 Knowledge Entity Extraction for User Behavior Information

The knowledge entity extraction is performed after obtaining the user behavior information. The user behavior information document contains not only the software component name, but also the name it graphs to in the underlying code.

After obtaining the user behavior information, it is necessary to first classify the user behavior information according to the functional modules and extract the user behavior information corresponding to different functional modules. In the software platform, different buttons in the UI interface correspond to different functional modules, according to which user behavior information is classified. If the user's behavior information involves multiple functional modules, such information is useful for mining the association relationship between modules, and the processing of such information depends on which module the subject of its operation belongs to.

The user behavior information documentation format is defined in Table 2.

The entities extracted based on user behavior documentation and their attributes are shown in Table 3.

6.3 Software Knowledge Entity Extraction for the Underlying Code

This research is focused on CIFLog, which was developed in the Java language. Therefore, this study is based on Java source code for software knowledge entity extraction. According to the Java programming specification,

Table 2 User behavior documentation

Item	Explanation
UserID	Unique identifier for each user
Function module	Function module where the user operates
Behavior information	User actions, including clicking buttons or performing operations on the interface
EName	Components' name in code
CName	The name of the component displayed to the user in the foreground
Date	The time when the user was operating
Note	If an exception occurs, write it in a note

Table 3 User behavior information

Entity	Attributes Belong to Entity
UI knowledge entity collection	{Function module, Interface, Button}
Function module	{English name, Chinese name, Interface, Function}
Interface	{English name, Chinese name, Module, Button, Function}
Button	{English name, Chinese name, Interface, Response event}
Function module 1	{[Button a]-> [Interface a]-> [Interface b]->..... [Button n/Interface n]}
Function module 2	{[Button 1]-> [Interface 1]-> [Interface 2]->..... [Button N/Interface N]}
.....	.....

the development source code consists of packages, classes, interfaces, and methods. This paper uses the QDox parser [25] to extract classes, interfaces, and method definitions from the software platform development code, and then obtains other information through recursive methods.

QDox is a small-footprint, high-speed parser that extracts metadata from a given Java source. When loading a Java file or a folder containing Java files into QDox, it automatically performs iterations. The QDox process employs different methods for extracting different kinds of metadata from the source code. These include:

(1) The getPackage() method lists all available packages for a given source.
(2) The getClasses() method lists all available classes in the package.
(3) The getMethods() method lists all available methods in the class.
(4) The getReturns() method returns the return type of the method.
(5) The getParameters() method lists all the parameters available to the method.
(6) The getType() method returns the type of the method.
(7) When the getComment() method is used with a package, class, and method, it returns the corresponding comment.

The output information of QDox stores the metadata in the form of a string. The information we extract is filled according to the top-level framework defined by the schema layer. When the user searches the entity, a large amount of association information of the entity can be obtained, or the system can quickly lock to the corresponding entity when the description

Table 4 Source code software knowledge

Software Knowledge Entity	Attribute Information
Software Knowledge Entity Collection	{Package, Class, Interface, Method}
Package	{Name, Comment}
Class	{Name, Belonging package, Modifier, Parent class, Implementation interface, Comment}
Interface	{Name, Belonging package, Parent interface, Comment}
Method	{Name, Signature, Class or interface, Modifier, Parameter, Return value type, Comment}

information is input. The specific extracted entities and their attributes are shown in Table 4.

6.4 Software Knowledge Entity Association Establishment

There is a complex relationship between software knowledge entities. In this paper, the relationship is established from two levels: top and bottom. The top level is the user level. First, a simple relationship is filled at the user level, and then the relationship between the top-level UI level and the underlying code is analyzed and built. The code level is the bottom layer. The top layer can be mapped to the bottom layer, and the bottom layer constitutes the display and function implementation of the top layer.

In terms of the top-level relationship, in the schema layer, we have defined the relationship between e_HasInterface, e_HasButton, and e_HasBug. In addition to these three kinds, there are strong correlation inferences based on FP-Growth algorithm, that is, rules obtained by FP-growth algorithm are used to fill the relationship between software knowledge entities. This paper further divides the strong associations into two types: strong association between the function modules and strong association between the buttons. Due to the nature of the interface, it contains more than one button and function, and it is essentially the function of the carrier and the response event of the button. Therefore, it does not have much practical significance to mining the strong association relationship, and so it is not considered here.

Because of the large number of associations between the software underlying code knowledge entities, the definition of the relationship in this part is more complicated. According to the Java programming specification, this paper divides relationships between software entities into dependency

Table 5 Software platform development source code entity relationship

Relationship Item	Attribute Information
Dependency relationship	{(Class, Class), (Interface, Interface)}
Association relationship	{(Class, Class), (Class, Interface)}
Aggregate relationship	{(Class, Class), (Class, Interface)}
Inheritance relationship	{(Class, Class), (Interface, Interface)}
Implementation relationship	{(Class, Interface)}
Call relationship	{(Method, Method)}
Contains relationship	{(Project, Module), (Project, Package), (Module, Package), (Package, Class), (Package, Interface), (Class, Method)}
Method parameter	{(Method, Class), (Method, Interface)}
Return value	{(Method, Class), (Method, Interface)}
Reference relationship	{(Method, Class)}

relationship, association relationship, aggregation relationship, inheritance relationship, implementation relationship, call relationship, inclusion relationship, method parameter, return value, and reference relationship. For example, dependencies mainly occur in class and interface entities, mainly in the form of a class relying on another class, or an interface relying on another interface, namely {(class, class), (interface, interface)}. However, association relationships mainly occur between classes and between classes, as well as between classes and interfaces. And the QDox parser is used to parse the Java file, and the regular expression is read into the data stream iterative search in order to find the above association relationship.

The relationship attributes are represented in the form of a set in Table 5.

6.5 Knowledge Retrieval and Display

This study is based on the use of the graph database to achieve the visual retrieval of user knowledge. After evaluating four databases, this paper selected the Neo4j graph database, which has an efficient and extensible declarative query language and development model. Neo4j employs Cypher, a declarative, easy-to-read graph query language that can be retrieved directly from the knowledge base.

The relational database selects the PostgreSQL open source relational database for its scalability, strong internal function support, execution of complex SQL and multiprocessing capability. In addition, during the data

Table 6 Software knowledge entity statistics

Statistical Item	Quantity
Number of entity types	832
Number of relationship types	3927
Number of attributes	3470
Number of functional modules	19
Number of user interfaces	51
Number of buttons	189
Number of bugs	27
Number of packages	22
Number of classes	181
Number of methods	647
Number of interfaces	46

migration process to the graph database, PostgreSQL performs better than other relational databases when using the ETL tools in Neo4j.

First, based on the defined schema layer, create a concept class and a relationship class based on the Neo4j creation mode, and then load the entity node information and relationships. In order to prevent duplicate information when importing data information, use Cypher query statements to determine whether to repeat and delete.

Using the Intelligence and HWGeoModel modules under CIFLog as examples, the statistics of the knowledge entity data obtained based on the knowledge extraction are shown in Table 6.

Based on user knowledge mining of these two modules, the strong association statistics obtained are shown in Table 7.

Using the association relationship between the modules as an example, through mining user behavior information, we find the following: that users often use the DataManagement module before using the HWGeoModel module; that they often use the Intelligence module while using the HWGeoModel module; and that while using DataManagement, they often use the Intelligence modules. Therefore, we can say that there is a strong relationship between them.

Using the example of finding the HWGeoModule using the Cypher language, the graphical visualization result of the "HWGeoModule" is shown in Figure 3. After the association rule mining based on user behavior

Table 7 Software knowledge entity statistics

Statistical Item	Quantity
{(Module), (Module)}	4
{(Module), (Button)}	30
{(Button), (Interface)}	38
{(Button), (Button)}	48
{(User interface), (User interface)}	12
Number of interfaces	46

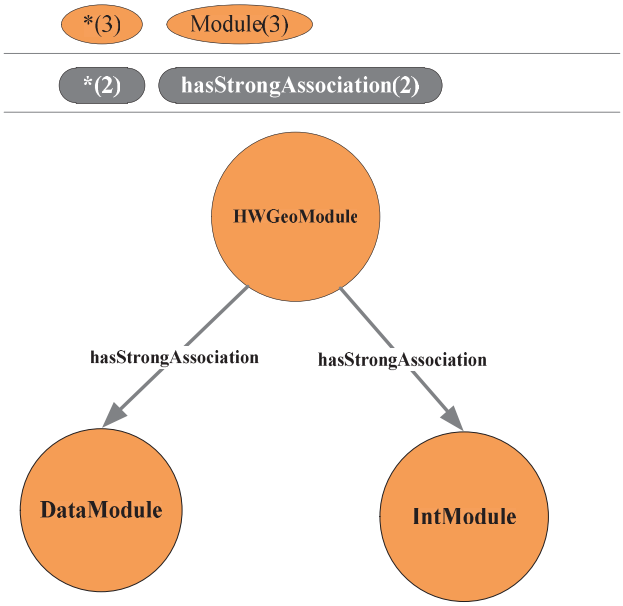

Figure 3 "HWGeoModule" module graphic visualization search results.

information, there are one-way strong association relationships between that module and "DataModule" and "IntModule,", and this knowledge has been stored in the user knowledge graph. When a strong association search for "HWGeoModule" is performed in Neo4j, "IntModule" and "DataModule" are output, and connected by directed edges, and the relationship represented by the edge is "strongAssociation".

Since the software platform source code knowledge is also integrated into the user knowledge graph construction process, the programmer can

use the user knowledge graph to carry out the relevant knowledge retrieval for software development. The following is an example of searching the "baseProperty" class, and the query result is as follows:

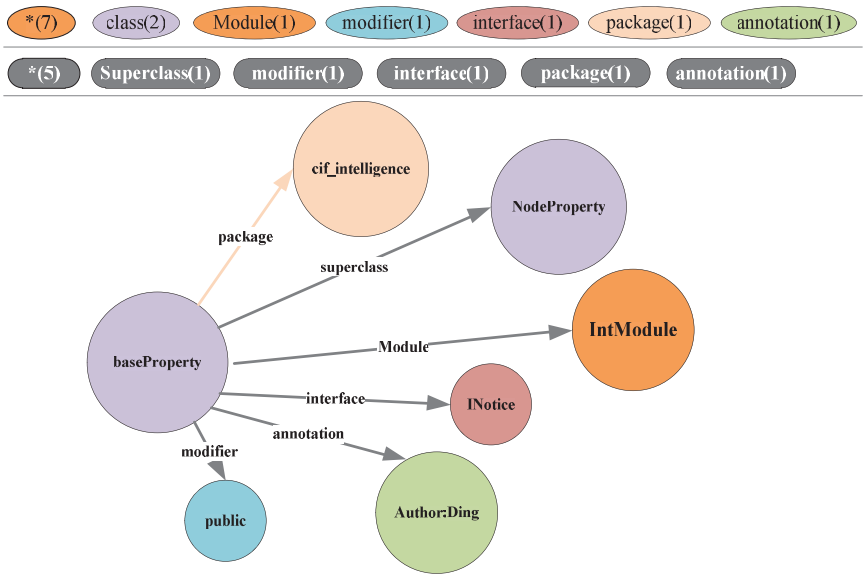

Figure 4 "baseProperty" class graphic visualization search results.

The search result modified in Figure 4 includes several entity nodes and relationship edges, and the information contained therein is "baseProperty" class and its attribute information: "baseProperty" belongs to "cif_intelligence" and the modifier is "public". The parent class is "NodeProperty", the implementation interface is "INotice", and the comment is "Author: Ding". Connected directed edges between nodes are used to indicate the relationship between them, such as: "hasSuperclass", "hasPackage", and "has Annotation". The different colors in the picture indicate that they belong to different categories, that is, they have different labels in the graph database.

6.6 Knowledge Update Mechanism

Because user behavior information is generated continuously, new user knowledge is also generated continuously, and the software platform is accordingly also updated and maintained. Therefore, the knowledge in the user behavior knowledge graph needs to be updated, which is also the reason

for using the combination of a relational database and a graph database in this paper.

The knowledge graph update is divided into two levels: the pattern layer and the data layer. The software platform discussed in this article, having been released, is relatively stable, and there will not be too many changes to the framework. Therefore, this paper focuses on the entity update of the data layer.

The relational database is directly related to the software platform background code and user behavior information document, and the user behavior document storage method is made more convenient through the relational database. Therefore, this study adopted the method of establishing association between relational database and graph database, and carried out data migration from relational database to graph database, so as to update knowledge in this way. In this paper, PostgreSQL database is used to correlate with Neo4j database, and data migration is adopted to update knowledge. When new rules are generated, they are mapped to the database and relational updates are made in the same way.

7 Conclusions and Future Work

This paper proposes a method for constructing a user behavior knowledge graph based on user behavior information and software platform development source code. It also describes in detail the construction process of this knowledge graph based on CIFLog. The knowledge graph is shown to improve the utilization of user knowledge in both the update and maintenance phases of the software platform, and to improve the acquisition, retrieval, and use of user knowledge by developers.

First, based on the user behavior information, the association rules and the user knowledge are mined. Then, the software entity is extracted from the user behavior information and the software platform development source code. Next, the knowledge is extracted from the development source code. The top-down approach builds the schema layer of the software platform developed by the Java project, and then builds the entire user behavior knowledge graph framework. This framework defines the semantic relationship between the concepts in the knowledge graph, and then performs the entity according to it. Next comes the extraction of relationships and attributes, and the establishment and filling of relationships between extracted entities, including entities at the UI level and graphing to entities in the development source code. Finally, the knowledge is stored in the Neo4j graph database, and

the formal search provided by Neo4j's declarative query language Cypher is used for the developer's graphical search of user knowledge and software platform knowledge.

This article has several purposes: to realize the mining, representation, and utilization of user knowledge; to improve the utilization of user knowledge by constructing a user behavior knowledge graph; and to upgrade the software platform based on user needs. When using this graph, in addition to intuitively and orderly searching and managing the user knowledge of the software platform, it can also infer user behavior trajectories facing different functions based on the user knowledge contained in the user behavior knowledge graph. Its application significance lies in the ease of operation of the optimized software, and the role it plays in interface design and operation guidance. This research group has already started the application work, such as developing the intelligent operation guidance function for CIFLog. In addition, related experiments have been completed in the test module. The experiment proves that the intelligent guidance function that was developed based on the user behavior knowledge graph has excellent expressiveness on software platforms for new users or users who are not familiar with some functions.

Building a systematic and long-term knowledge base is a complex task because it contains not only intuitive knowledge, but also empirical hidden knowledge. Therefore, in future work, we will continue to study the update mechanism of the knowledge graph, infer the development trend of software platforms based on existing knowledge, build intelligent operation guides, and improve the utilization of user knowledge graph, while applying it to more software development projects.

Acknowledgments

The research described in this paper was fully supported by Heilongjiang Provincial Philosophy and Social Science Foundation of China under Grant No. 19SHE280.

References

[1] F. Wang, J.P. Liu, B. Liu, T.Y. Qian, Y.H. Xiao, Z.Y. Peng, 'Survey on construction of code knowledge graph and intelligent software development', Journal of Software, vol. 11, no. 18, Nov. 2019, pp. 1–20.

[2] P. Huang, A. Tafti, S. Mithas, 'Platform Sponsor Investments and User Contributions in Knowledge Communities: The Role of Knowledge Seeding', MIS Quarterly, vol. 42, no. 1, 2018, pp. 213–240.
[3] K. Dhindsa, D. Carcone, S. Becker, 'Toward an Open-Ended BCI: A User-Centered Coadaptive Design', Neural computation, 2017, 29(10): 2742–2768.
[4] L. Ning, Wang, 'CIFLog: the 3rd generation logging software based on Java-NetBeans', Acta Petrolei Sinica, vol. 34, no. 1, 2013, pp. 192–200.
[5] Kang, Sangwoo, and J. Seo, 'Two-phase reanalysis model for understanding user intention', Pattern Recognition Letters, 42(2014): 35–39.
[6] M. Xin, Y. Zhang, S. Li, et al., 'A Location-Context Awareness Mobile Services Collaborative Recommendation Algorithm Based on User Behavior Prediction', International Journal of Web Services Research, vol. 2, no. 14, 2017, pp. 45–66.
[7] Burkhardt, Dirk, et al., 'Search Intention Analysis for Task- and User-Centered Visualization in Big Data Applications', Procedia Computer Science 104(2017): 539–547.
[8] S. Lakshmi Priya, S. Varatharajan, 'Parallel algorithm based consumer behavior analysis for generating personalized ontology system', International Journal of Managment, IT and Engineering, 2013, 3(5).
[9] Lei, Wu, F. Qing, and J. Zhou, 'Improved Personalized Recommendation based on Causal Association Rule and Collaborative Filtering', International Journal of Distance Education Technologies, 14.3(2016): 21–33.
[10] Khater, Shaymaa, D. Gracanin, and H.G. Elmongui, 'Personalized Recommendation for Online Social Networks Information: Personal Preferences and Location-Based Community Trends', IEEE transactions on computational social systems, 99(2017): 1–17.
[11] Singal, Himani, and S. Kohli, 'Trust Necessitated through Metrics: Estimating the Trustworthiness of Websites', Procedia Computer Science, 85(2016): 133–140.
[12] Majed Alrubaian, Muhammad Al-Qurishi, Mabrook Al-Rakhami, Mohammad Mehedi Hassan, Atif Alamri, 'Reputation-based credibility analysis of Twitter social network users', Concurrency and Computation: Practice and Experience, 2017, 29(7).
[13] Zhou Wei, Wen Junhao, Qu Qiang, Zeng Jun, Cheng Tian, 'Shilling attack detection for recommender systems based on credibility of group users and rating time series', PloS one, 2018, 13(5).

[14] Meng XW, Wang F, Shi YC, et al., 'Mobile User Requirements Acquisition Techniques and Their Applications', Journal of Software, vol. 3, no. 25, Dec. 2013, pp. 439–456.
[15] C. Wang, W. Zhao, J. Wang, et al., 'Multidimensional customer requirements acquisition based on ontology', Computer Integrated Manufacturing Systems, vol. 4, no. 22, Jan. 2016, pp. 908–916.
[16] Y. Wang, S. Yu, T. Xu, 'A User Requirement Driven Framework for Collaborative Design Knowledge Management', Advanced Engineering Informatics, vol. 33, 2017, pp. 16–28.
[17] M. Maouche, and M. Bettaz, 'Towards a Software Engineering Approach to Multi-Scale Modeling and Simulation', International Journal of Software Engineering & Its Applications, 2016, 10(11): 205–218.
[18] R.A.B. Ouriques, K. Wnuk, T. Gorschek, et al., 'Knowledge Management Strategies and Processes in Agile Software Development: A Systematic Literature Review', International Journal of Software Engineering and Knowledge Engineering, 2019, 29(3): 345–380.
[19] De Vasconcelos, Jose Braga, et al., 'The application of knowledge management to software evolution', International Journal of Information Management 37. 1pt. A(2017): 1499–1506.
[20] Goncalves, Joshua, and A. Krishna, 'Incorporating Change Management Within Dynamic Requirements-Based Model-Driven Agent Development', The Computer journal, 2017, 60(7): 1044–1077.
[21] W.P. Li, J.B. Wang, Z.Q. Lin, J.F. Zhao, Y.Z. Zou, B. Xie, 'Software Knowledge Graph Building Method for Open Source Project', Journal of Frontiers of Computer Science and Technology, vol. 11, no. 6, 2017, pp. 851–862.
[22] Arshad, Muhammad U., et al., 'Efficient and Scalable Integrity Verification of Data and Query Results for Graph Databases', IEEE Transactions on Knowledge & Data Engineering, 99(2017): 866–879.
[23] P.J. Foote, D.G. Stuart, R. Elmore-Yalch, 'Exploring customer loyalty as a transit performance measure', Transportation Research Record: Journal of the Transportation Research Record, vol. 1783, 2001, pp. 93–101.
[24] R. Agrawal, R. Srikant, 'Fast algorithms for mining association rules', Proc of International Conference on Very Large Databases. 1994, pp. 487–499.
[25] G. Gopinath, S. Sagayaraj, 'To Generate the Ontology from Java Source Code', International Journal of Advanced Computer Science and Applications, vol. 2, no. 2, 2011.

Biographies

Fuhua Shang is a professor at the School of Computer and Information Technology, Northeast Petroleum University. He is a senior member of China Computer Federation. He received his M.S. in Computer Application from Harbin Institute of Technology, China in 1990 and a Ph.D. degree in Computer Major from Harbin Institute of Technology, China in 2007. His main research areas include artificial intelligence, knowledge representation, and deep learning.

Qiuyu Ding is a Ph.D. candidate at Harbin Institute of Technology, China. She received her B.S. in Computer Science and Technology from Northeast Petroleum University, China in 2017, and purchase an M.S. in Software Engineering from there in 2020. Her main research directions include natural language processing, knowledge representation, and artificial intelligence.

Ruishan Du is an associate professor at the School of Computer and Information Technology at Northeast Petroleum University. He received his M.S. degree from Northeast Petroleum University in 2007 and is currently studying for his doctorate there. His main research areas include artificial intelligence, knowledge graph, and deep learning.

Maojun Cao, associate professor, received his M.S. degree in computer engineering from Northeast Petroleum University in 2008 and his Ph.D. in earth exploration and information technology from China Petroleum Exploration and Development Research Institute in 2018. Focusing on the field of oil and gas logging processing interpretation and artificial intelligence for 15 years, he also studies software engineering and large-scale software platform development, logging processing and interpretation software and 3D attribute modeling, knowledge mapping, etc.

Huanyu Chen is a M.S. candidate at Northeast Petroleum University, China. He received his B.A. degree in 2014 from Northeast Petroleum University. His main research areas include machine learning and knowledge graph.