Lin ZQ, Xie B, Zou YZ *et al.* Intelligent development environment and software knowledge graph. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 32(2): 242–249 Mar. 2017. DOI 10.1007/s11390-017-1718-y

Intelligent Development Environment and Software Knowledge Graph

Ze-Qi Lin [1,2], Bing Xie [1,2,*], *Senior Member, CCF*, Yan-Zhen Zou [1,2], *Member, CCF, ACM, IEEE*
Jun-Feng Zhao [1,2], *Member, CCF*, Xuan-Dong Li [3], *Fellow, CCF, Member, ACM, IEEE*
Jun Wei [4], *Senior Member, CCF*, Hai-Long Sun [5], *Senior Member, CCF, Member, ACM, IEEE*
and Gang Yin [6], *Member, CCF*

[1] *Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education Beijing 100871, China*

[2] *Beida (Binhai) Information Research, Tianjin 300450, China*

[3] *National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China*

[4] *Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China*

[5] *School of Computer Science and Engineering, Beihang University, Beijing 100191, China*

[6] *College of Computer, National University of Defense Technology, Changsha 410073, China*

E-mail: {linzeqi, xiebing, zouyz, zhaojf}@pku.edu.cn; lxd@nju.edu.cn; weijun@iscas.ac.cn; sunhl@act.buaa.edu.cn jack_nudt@163.com

Received November 6, 2016; revised November 28, 2016.

Abstract Software intelligent development has become one of the most important research trends in software engineering. In this paper, we put forward two key concepts — intelligent development environment (IntelliDE) and software knowledge graph — for the first time. IntelliDE is an ecosystem in which software big data are aggregated, mined and analyzed to provide intelligent assistance in the life cycle of software development. We present its architecture and discuss its key research issues and challenges. Software knowledge graph is a software knowledge representation and management framework, which plays an important role in IntelliDE. We study its concept and introduce some concrete details and examples to show how it could be constructed and leveraged.

Keywords intelligent development environment, software big data, software knowledge graph, semantic search

1 Introduction

Software intelligent development is defined as providing intelligent assistance for software development[1]. Recent years, software intelligent development has become one of the most important research trends in software engineering with the rapid development of software big data and artificial intelligence technology. The project of Big Data Based Software Intelligent Development Methods and Environments is a recent key project funded by China's Ministry of Science and Technology as a part of the National Key Research and Development Program of Cloud Computing and Big Data in 2016. It aims to promote the intelligence level of software development technologies and platforms via software big data mining, analysis and understanding.

Intelligent assistance for software development can be divided into two main categories: intelligent recommendation and intelligent question-answering. Intelligent recommendation means recommending relevant software resources based on user context in various software development activities[2]. Many recommendation strategies are learned from software big data. Intelligent question-answering means finding answers of users' questions from software resources automatically. This

Regular Paper
Special Section on MOST Cloud and Big Data
This work is supported by the National Key Research and Development Program of China under Grant No. 2016YFB1000800, and the National Science Fund for Distinguished Young Scholars of China under Grant No. 61525201.
*Corresponding Author

is a knowledge-intensive task, requiring semantic understanding and inference for software big data.

In this paper, we put forward two key concepts — intelligent development environment (IntelliDE) and software knowledge graph — for the first time. IntelliDE is an ecosystem in which software big data is aggregated, mined and analyzed to provide intelligent assistance in the life cycle of software development. It proposes that software intelligent development passes through three stages: data aggregation, knowledge acquisition, and intelligent assistance. Based on this idea, IntelliDE illustrates how to develop the well-rounded technology architecture, software ecosystem and service environment for software intelligent development. Software knowledge graph is a software knowledge representation and management framework. In IntelliDE, complex software knowledge is extracted from software big data; thus we propose software knowledge graph to ensure that software knowledge could be represented uniformly, connected together and easy to reuse.

Section 2 is about IntelliDE. We introduce its architecture and discuss its key research issues and challenges. Section 3 presents the definition and key ideas of software knowledge graph, and introduces some details and examples about how we construct and leverage it at the present stage. Section 4 concludes the paper.

2 IntelliDE

Modern software development activities have generated web-scale software data, especially open source software projects on the Internet. The wide practice of enterprise engineering also provides rich domain specific data. These software data are large-scale, multi-source, heterogeneous, distributed and fast-growing, containing rich and complex software knowledge. At the same time, the rapid development of information extraction and machine learning technologies makes automatic software knowledge extraction easier and faster. Therefore, software intelligent development is becoming one of the most popular research issues in software engineering.

However, existing studies in software intelligent development are usually conducted case by case. Different researchers study different intelligent assistance tasks (e.g., code completion[3], defect prediction[4] and bug location[5]). For different tasks, specific datasets are collected separately, and different intelligent assistance tools are created. We call this problem as diverse-tooling[6]. Diverse-tooling makes it difficult to reuse these software data, knowledge and tools in software development. Therefore, a uniform ecosystem for software intelligent development is needed.

For this purpose, we propose intelligent development environment (IntelliDE). Fig.1 shows its architecture. In this ecosystem, software intelligent development passes through three stages: data aggregation, knowledge acquisition, and intelligent assistance. Software knowledge is extracted from software big data, and then leveraged to promote the intelligence level of software development environment. In Subsection 2.1∼Subsection2.3, we discuss key research issues and challenges in these three stages respectively.

2.1 Data Aggregation

Issue. Software big data is web-scale, distributed, multi-source, heterogeneous, dynamic and fast-growing. Therefore, IntelliDE needs an ultra-large-scale, up-to-date and easy-to-access software data repository to solve the problem of insufficient data supply for software engineering researchers.

Challenges. IntelliDE faces three main research challenges in data aggregation: data collection, data fusion, and data update.

1) *Data Collection.* We need to study the forms and features of software big data on the Internet so that IntelliDE can sense them pertinently, download them accurately and store them organically.

2) *Data Fusion.* Different aspects of software are hidden in multi-source and heterogeneous software data. Therefore, we need to study how to discover the relationships between different software data sources so that researchers can analyze them jointly.

3) *Data Update.* Keeping the software data repository up-to-date is an important task since software big data is dynamic and fast-growing. Therefore, we need to study software big data update strategies (e.g., incremental update and partial update).

2.2 Knowledge Acquisition

Issue. Complex software knowledge is hidden in large-scale software data, and software knowledge is the key to providing intelligent assistance for software development. Therefore, IntelliDE needs the support of knowledge acquisition, i.e., we need to study how to construct a knowledge base from the software data repository.

Challenges. IntelliDE faces two main research challenges in knowledge acquisition: knowledge extraction,

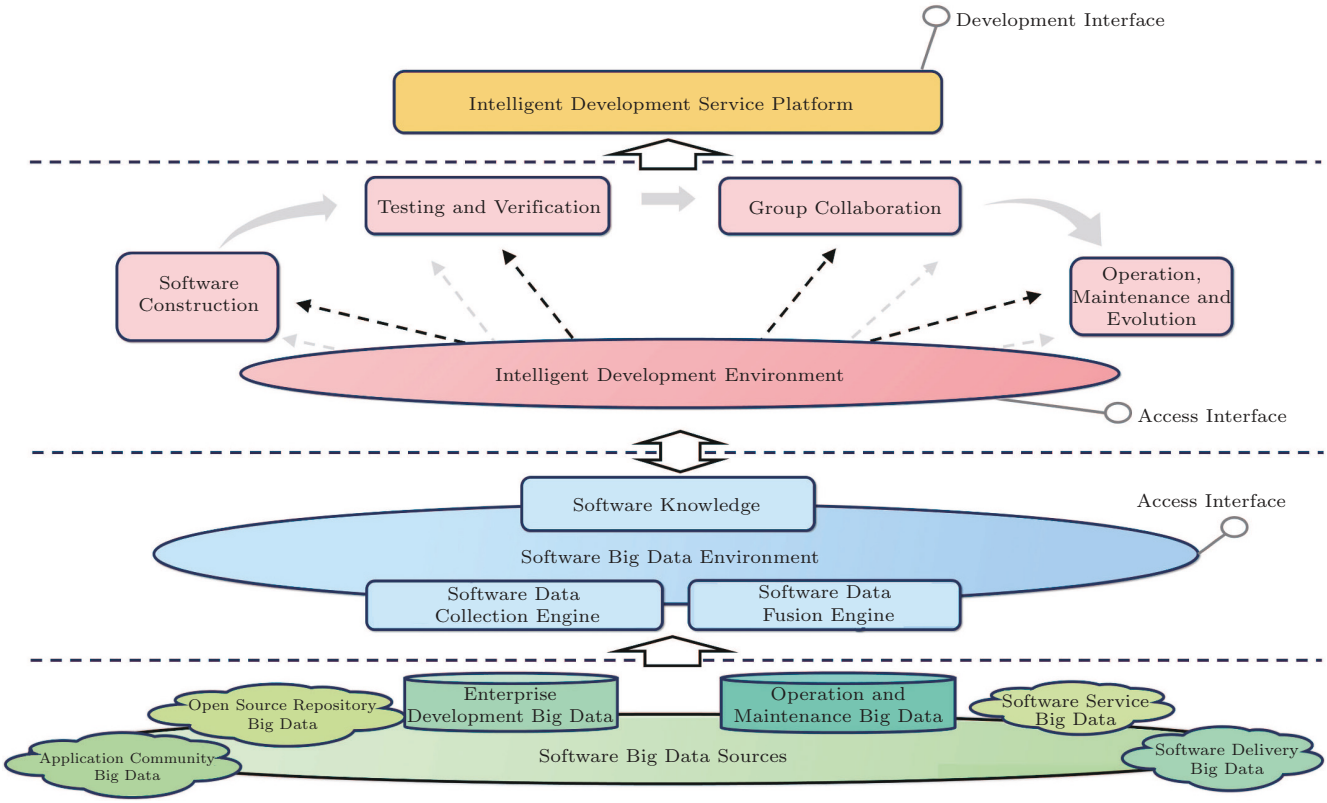


Fig.1. Architecture of IntelliDE.

and knowledge representation and management.

1) *Knowledge Extraction.* IntelliDE needs to extract various kinds of software knowledge from software big data automatically. Therefore, we need to study various software knowledge extraction algorithms. These extraction algorithms are mainly based on program analysis, natural language processing, data mining, machine learning and information retrieval, etc., but are not limited to them.

2) *Knowledge Representation and Management.* To ensure that the intelligent development environment could leverage the knowledge efficiently and systematically, we need to study how software knowledge could be represented uniformly, linked together and queried easily. We propose software knowledge graph to solve this challenge, and the detailed solution is presented in Section 3.

2.3 Intelligent Assistance

Issue. In IntelliDE, software knowledge extracted from software big data is leveraged to provide intelligent recommendation services and intelligent question-answering services for software developers. Software developers are concerned about various tasks in different stages in the life cycle of software development, and providing intelligent assistance for different tasks requires different software knowledge and different knowledge processing strategies. Therefore, IntelliDE needs a series of intelligent development services, leveraging software knowledge through different algorithms to solve different software engineering tasks intelligently.

Challenges. We conclude the main concerned intelligent development services in IntelliDE as four categories: 1) software construction, 2) testing and verification, 3) group collaboration, and 4) operation, maintenance, and evolution.

1) *Software Construction.* IntelliDE needs to provide intelligent development services for different software construction activities (e.g., domain analysis, programming and software refactoring). These services include software project knowledge visualization and browsing, software document semantic retrieval, intelligent code synthesis and completion, software refactoring recommendation, etc.

2) *Testing and Verification.* IntelliDE needs to provide software big data based software testing and verification technology and methods, including test cases generation, inspecting code by bounded model check-

ing, identifying warnings output from static analysis, debugging software, etc.

3) *Group Collaboration.* IntelliDE needs to analyze the skill characteristics of developers and the potential collaborative relationship among them, leveraging software knowledge (especially historical collaboration knowledge) to improve developer recommendation, task assignment, resource recommendation, etc.

4) *Operation, Maintenance, and Evolution.* IntelliDE needs to solve problems in the scope of whole software lifecycle, such as the fragmentation between software development and system maintenance, and the incapability of fast adaptation to changing requirements. Relevant intelligent development services include intelligent load testing, anomaly analysis, service interface design and evolution, etc.

3 Software Knowledge Graph

In this section, we present software knowledge graph, the software knowledge representation and management framework in IntelliDE.

3.1 Definition

Knowledge graph means a graph made up of nodes and directed edges, which is used to represent domain knowledge[7]. Software knowledge graph is defined as a graph for representing relevant knowledge in software domains, projects, and systems. In a software knowledge graph, nodes represent software knowledge entities (e.g., classes, issue reports and business concepts), and directed edges represent various relationships between these entities (e.g., method invocation and traceability link). Different nodes and relationships have different properties to describe their inner features. For example, a method entity's properties include its name, return type, parameter list, access modifier, and description.

Knowledge in software knowledge graph is divided into two categories: primitive knowledge and derivative knowledge.

Primitive knowledge is defined as software knowledge added into software knowledge graph through data structure parsing. For example, we may parse the abstract syntax trees (ASTs) of source code files, and then code entities (such as classes, methods and fields) and relationships between them (such as method invocation and inheritance hierarchy) could be added into software knowledge graph. And we may parse the structure of mail archive files, and then mail entities (such as mails and mail users) and relationships between them (such as mail sending, mail receiving, and mail replying) could be added into software knowledge graph.

Derivative knowledge is defined as software knowledge added into software knowledge graph through mining existing software knowledge in it. For example, we may extract business concept entities through processing natural language sentences in software document entities. We may recover traceability links between document entities and code entities. And we may further leverage these traceability links to recover traceability links between business concept entities and code entities.

Based on the definition, we construct software knowledge graph automatically from software big data. The construction process contains two phases: data parsing and knowledge extraction. The data parsing phase adds various kinds of primitive knowledge into a software knowledge graph through parsing various software data. Then the knowledge extraction phase adds various kinds of derivative knowledge into software knowledge graph via various knowledge extraction algorithms.

The software knowledge graph should be ever-evolving. When new data parsing modules or knowledge extraction modules are developed and integrated in this framework, new kinds of software knowledge could be accumulated in the software knowledge graph. A software knowledge ecosystem could be formed based on this framework, and IntelliDE could leverage it to provide intelligent assistance for software development.

3.2 Software Knowledge Graph in IntelliDE

In this subsection, we describe the software knowledge graph construction platform we have implemented in IntelliDE at the present stage. Currently, we focus on constructing a software knowledge graph for a given software project. Fig.2 shows a logical overview of this platform.

3.2.1 Data Parsing

When a software developer wants to construct a software knowledge graph for a software project, he/she should add relevant software data to our platform via its web front-end. The added software data are allowed to be large-scale, multi-source and heterogeneous. Currently, data formats supported by this platform include: 1) source code files (e.g., Java); 2) version control systems (e.g., VCS, SVN and Git); 3) mailing lists

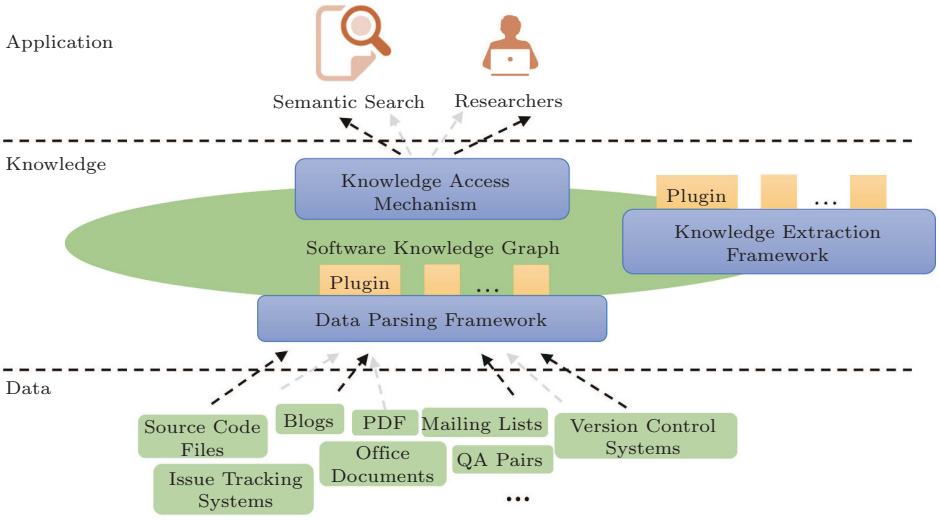


Fig.2. Logical overview of the software knowledge graph construction platform. OA: question-answer.

(e.g., mbox); 4) issue tracking systems (e.g., JIRA and BugZilla); 5) Microsoft office (e.g., DOC, DOCX) and PDF documents; 6) html-format tutorials, API documentation, user forum posts and blogs; 7) online social question-answering pairs (StackOverflow XML dumps).

We provide a data parsing framework in the software knowledge construction platform. This framework provides a data parsing plugin interface so that different data parsing plugins could be developed and integrated together.

3.2.2 *Knowledge Extraction*

After the data parsing phase adds primitive knowledge into the software knowledge graph, we use various knowledge extraction algorithms to add derivative knowledge into the software knowledge graph. There is a knowledge extraction framework in the software knowledge graph construction platform. It provides a knowledge extraction interface so that different knowledge extraction algorithms could be developed as plugins and integrated together. Currently, the knowledge extraction plugins we implement include: 1) document-to-code traceability link recovery[8], 2) document-to-document lexical similarity estimation[9], 3) issue-to-commit link recovery, 4) stakeholder identification, 5) source code latent topic modelling[10], 6) API usage example extraction[11], etc.

3.2.3 *Storage and Query*

We use Neo4j①, a popular graph database, to store software knowledge graph. Cypher②, a declarative query language that allows for expressive and efficient querying of graph data, is used to query software knowledge graph. Besides Cypher, we also use Java API③ provided by Neo4j to access the software knowledge graph. Cypher is more concise than Java API, but Java API can implement more complex graph algorithms.

3.3 Application

IntelliDE aims to provide the support of intelligent recommendation and intelligent question-answering for software development. In this subsection, we introduce software text semantic search, a current preliminary study, as an example to show how software knowledge graph could be leveraged to assist software developers.

3.3.1 *Software Text Semantic Search*

Text search is defined as the matching of some free-text user queries against a set of free-text documents. Since text is the common form of information representation among various software artifacts at different abstraction levels (such as identifiers, comments, documentation, issue reports, commit message, developer discussions, and user communications), text search is widely used in software development, maintenance, and reuse[12]. However, keyword matching based text search

①https://neo4j.com/, Jan. 2017.
②http://neo4j.com/docs/cypher-refcard/current/, Jan. 2017.
③http://neo4j.com/docs/java-reference/current/, Jan. 2017.

techniques suffer from the problem that they cannot understand the semantic meaning of free-text. Therefore, we propose a software knowledge graph based software text semantic search approach. Developers are allowed to input free-text queries. For each query, we return a ranked list of text items (i.e., text properties in the software knowledge graph) semantically relevant to it, and for each text item in the list, we visualize a subgraph of the software knowledge graph to demonstrate how this text item is semantically relevant to the query.

The basic idea of this approach is that well-structured source code knowledge can be used as a bridge to mine semantic relevance between text items[13-14]. The approach is composed of three phases.

Code Location. We link free-text to relevant code entities in the software knowledge graph via feature location technique[15].

Entity Matching. We measure the structural similarity between different code entities based on TransR[16], a graph embedding technique. The structural similarity re-ranks text search results.

Subgraph Extraction. We extract a subgraph from the software knowledge graph to visualize the relationships between a query and a related text item. This phase is accomplished based on a minimum spanning tree based algorithm[17].

3.3.2 Case Study

We construct a software knowledge graph for Apache Lucene④, a popular open source software project, as an example to demonstrate our software text semantic search approach. Relevant data contain: source code files (800 000+ lines), version control logs (1.2+ GB), emails (244 000+), issue reports (5 200+), official web pages (500+), blogs (1 200+), StackOverflow Q&A (question-answer) pairs (3 800+), etc. We extract 64 932 software knowledge entities and 276 065 relationships from these software data. Then, a software text semantic search engine is provided, and a semantic search example is discussed.

The example user query is "how to get all field names using class IndexReader". The semantic search result is shown in Fig.3. The left of this figure lists a ranked list of text items semantically relevant to the query. For each text item in the ranked list, the semantic search engine demonstrates a subgraph of the software knowledge graph for it. For example, consider the text item ranked in the second place, which is a StackOverflow Q&A pair. The user can click it and then corresponding subgraph would be shown in the right of the semantic search web front-end interface, as shown in Fig.3. In the subgraph, nodes in the "query" column represent the query and some keywords in it. Similarly, nodes in the "document" column represent the Q&A pair and some keywords in it. The subgraph shown in the "code" column illustrates how different keywords are semantically related through the software knowledge graph. For example, the keyword "IndexReader" occurring in the query is semantically relevant to the keyword "AtomicReader" occurring in the Q&A pair, since class AtomicReader is the super-class of class IndexReader. Though the query and the Q&A pair share few common keywords, the software knowledge graph bridges them semantically. Therefore, this Q&A pair is returned to help the user.

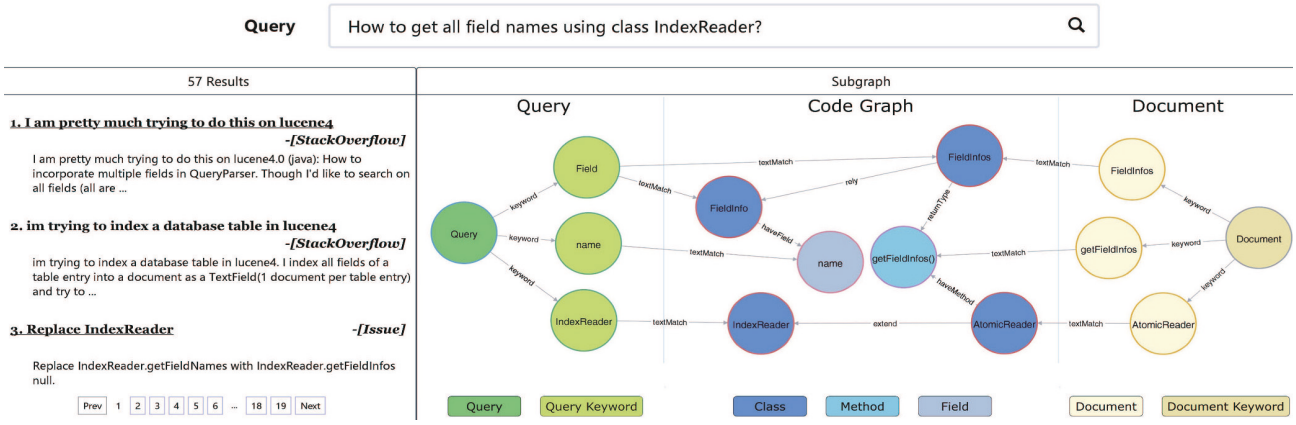


Fig.3. Example of software text semantic search.

④http://lucene.apache.org/, Jan. 2017.

In this example, we can see that software knowledge graph can be used to assist the semantic understanding of free-text in software effectively. It shows that the software text semantic search engine is a good start of the intelligent utilization of software knowledge graph for improving the efficiency and quality of software development activities.

4 Conclusions

In this paper, we proposed two key concepts— IntelliDE and software knowledge graph — for the first time. IntelliDE is an ecosystem which aims to lead the current integrated development environments (IDE) to intelligent development environment (IntelliDE), and the key research issues and challenges are concluded as: data aggregation, knowledge acquisition, and intelligent assistance. Software knowledge graph is a software knowledge representation and management framework. It is proposed as an infrastructure for knowledge acquisition in IntelliDE. Currently, our software knowledge graph contains not only primitive knowledge directly extracted from data sources (e.g., source code files and issue tracking systems), but also derivative knowledge (e.g., traceability links, API usage examples and latent topics). We provided a software text semantic search engine as an example application to show how software knowledge graph can provide the support of semantic understanding and inference in IntelliDE. And we hope that it will lead to intelligent question-answering in software domain in future.

References

[1] Kaiser G E, Feiler P H, Popovich S S. Intelligent assistance for software development and maintenance. *IEEE Software*, 1988, 5(3): 40-49.

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

[3] Raychev V, Vechev M, Yahav E. Code completion with statistical language models. *ACM SIGPLAN Notices*, 2014, 49(6): 419-428.

[4] Shepperd M, Bowes D, Hall T. Researcher bias: The use of machine learning in software defect prediction. *IEEE Transactions on Software Engineering*, 2014, 40(6): 603-616.

[5] Ye X, Bunescu R, Liu C. Learning to rank relevant files for bug reports using domain knowledge. In *Proc. the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Nov. 2014, pp.689-699.

[6] Trautsch F, Herbold S, Makedonski P, Grabowski J. Adressing problems with external validity of repository mining studies through a smart data platform. In *Proc. the 13th International Conference on Mining Software Repositories*, May 2016, pp.97-108.

[7] Liu Q, Li Y, Duan H, Liu Y, Qin Z G. Knowledge graph construction techniques. *Journal of Computer Research and Development*, 2016, 53(3): 582-600. (in Chinese)

[8] Dagenais B, Robillard M P. Recovering traceability links between an API and its learning resources. In *Proc. the 34th International Conference on Software Engineering*, June 2012, pp.47-57.

[9] Ye X, Shen H, Ma X, Bunescu R, Liu C. From word embeddings to document similarities for improved information retrieval in software engineering. In *Proc. the 38th International Conference on Software Engineering*, May 2016, pp.404-415.

[10] Hua Z B, Li M, Zhao J F, Zou Y Z, Xie B, Li C. Code function mining tool based on topic modeling technology. *Computer Science*, 2014, 41(9): 52-59. (in Chinese)

[11] Wang L J, Fang L, Wang L Y, Li G, Xie B, Yang F Q. APIExample: An effective web search based usage example recommendation system for Java APIs. In *Proc. the 26th IEEE/ACM International Conference on Automated Software Engineering*, Nov. 2011, pp.592-595.

[12] Marcus A, Antoniol G. On the use of text retrieval techniques in software engineering. In *Proc. the 34th IEEE/ACM International Conference on Software Engineering, Technical Briefing*, June 2012.

[13] Chan W K, Cheng H, Lo D. Searching connected API subgraph via text phrases. In *Proc. the 20th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, Nov. 2012, Article No. 10.

[14] Mcmillan C, Poshyvanyk D, Grechanik M, Xie Q, Fu C. Portfolio: Searching for relevant functions and their usages in millions of lines of code. *ACM Transactions on Software Engineering and Methodology*, 2013, 22(4): Article No. 37.

[15] Marcus A, Sergeyev A, Rajlich V, Maletic J I. An information retrieval approach to concept location in source code. In *Proc. the 11th Working Conference on Reverse Engineering*, Nov. 2004, pp.214-223.

[16] Lin Y K, Liu Z Y, Sun M S, Liu Y, Zhu X. Learning entity and relation embeddings for knowledge graph completion. In *Proc. the 29th AAAI Conference on Artificial Intelligence*, Jan. 2015, pp.2181-2187.

[17] Schuhmacher M, Ponzetto S P. Knowledge-based graph document modeling. In *Proc. the 7th ACM International Conference on Web Search and Data Mining*, Feb. 2014, pp.543-552.

Ze-Qi Lin received his B.E. degree in computer science from Peking University, Beijing, in 2014. He is currently pursuing his Ph.D. degree in computer science at Peking University, Beijing. His main research interests include software engineering, software reuse, knowledge engineering, data mining, etc.

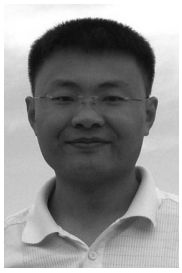
Bing Xie received his Ph.D. degree in computer science from National University of Defense Technology, Changsha, in 1998. He is a professor and Ph.D. supervisor in Peking University, Beijing. His main research interests include software engineering, formal method, software reuse, etc.

Yan-Zhen Zou received her Ph.D. degree in computer science from Peking University, Beijing, in 2010. She is an associate professor in Peking University, Beijing. Her main research interests include software engineering, software reuse, information retrieval, etc.

Jun-Feng Zhao received her Ph.D. degree in computer science from Peking University, Beijing, in 2005. She is an associate professor in Peking University, Beijing. Her main research interests include software engineering, software reuse, Web services, cloud computing, ubiquitous computing, etc.

Xuan-Dong Li received his B.S., M.S., and Ph.D. degrees all in computer science from Nanjing University, Nanjing, in 1985, 1991 and 1994 respectively. Since 1994 he has been in the Department of Computer Science and Technology at the Nanjing University where he is currently a professor. He is also a professor of the National Key Laboratory for Novel Software Technology, Nanjing University. His research interests include formal support for design and analysis of reactive, disturbed, real-time, and hybrid systems, software testing and verification, and model-driven software development.

Jun Wei received his B.S. degree in 1992 and his Ph.D. degree in 1997, both in computer science and from Wuhan University, Wuhan. He was a visiting researcher in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, in 2000. He is a professor in the Institute of Software, Chinese Academy of Sciences (ISCAS), Beijing. His area of research is software engineering and distributed computing, with emphasis on middleware-based distributed software engineering.



Hai-Long Sun received his Ph.D. degree in computer software and theory from Beihang University, Beijing, in 2008. Currently he is an associate professor in the School of Computer Science and Engineering of Beihang University, Beijing. His research interests include software engineering, crowdsourcing and distributed systems. He is a senior member of CCF and a member of ACM and IEEE.

Gang Yin received his Ph.D. degree in computer science from National University of Defense Technology (NUDT), Changsha, in 2006. He is currently an associate professor in NUDT, Changsha. His main research interests include software engineering, big data and cloud computing, etc.