International Journal of Software and Informatics, ISSN 1673-7288
http://www.ijsi.org, ijsi@iscas.ac.cn, +86-10-62661048
IJSI, 2021, 11(2): 217–241, doi:10.21655/ijsi.1673-7288.00250

Research Article

Open Source Software Supply Chain for Reliability Assurance of Operating Systems

Guanyu Liang (梁冠宇)[1,2], Yanjun Wu (武延军)[1,3], Jingzheng Wu (吴敬征)[1,3], Chen Zhao (赵琛)[1,3]

[1] (Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2] (University of Chinese Academy of Sciences, Beijing 100190, China)
[3] (State Key Laboratory of Computer Science (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)
Corresponding author: Yanjun Wu, yanjun@iscas.ac.cn

Abstract Software reliability is one of the research hotspots in the field of software engineering, and failure rate analysis is a typical research method for software reliability. However, the software construction mode has evolved from a single mode to a large-scale collaborative mode represented by open source software. As one of the representative products, the operating system includes the open source software connected through combinations and dependencies to form a supply network of tens of thousands of nodes. Typical methods lack consideration of supply relationships and cannot accurately identify and evaluate the software reliability issues introduced as a result. This paper extends the concept of supply chain to the field of open source software and proposes a knowledge-based management method for software supply reliability in a collaborative model. The ontological body is designed for the open source software ecosystem firstly, and then the knowledge graph of the open source software is constructed to achieve the extraction, storage and management of knowledge; driven by knowledge, combined with traditional supply chain management methods, a set of reliability management methods for the open source software supply chain is proposed, which constitutes a management system of the open source software supply chain. With the construction of a Linux operating system distribution as an example, the experiment demonstrates how the open source software supply chain supports the reliability of the operating system. Results show that the open source software supply chain will help to clarify and evaluate the reliability risk of large complex system software.

Keywords operating system; software reliability; open source software; supply chain; knowledge graph

Citation Liang GY, Wu YJ, Wu JZ, Zhao C. Open source software supply chain for reliability assurance of operating systems, *International Journal of Software and Informatics*, 2021, 11(2): 217–241. http://www.ijsi.org/1673-7288/00250.htm

This is the English version of the Chinese article "面向操作系统可靠性保障的开源软件供应链. 软件学报, 2020, 31(10): 3056–3073. doi: 10.13328/j.cnki.jos.006070".

Funding items: Bulletin of Chinese Academy of Sciences (XDC05040100); National Key Research and Development Program of China (2017YFB0801900); Key Research Program of Frontier Sciences, Chinese Academy of Sciences (ZDBS-LY-JSC038); National Natural Science Foundation of China (61772507)
Received 2020-02-16; Revised 2020-04-04; Accepted 2020-05-09; IJSI published online 2021-06-22

Traditional research on software reliability focuses on the ability of systems and components to perform the required functions at a specified time interval and under specified conditions in a piece of software[1]. In the reliability research of one piece of software, the software is usually divided into several independent modules. Then, the relationship between software modules and the failure rate of each module are analyzed with different models to obtain the final evaluation results of software reliability[2].

However, with the continuous proliferation of the open source collaborative model, the construction and development process of the software has undergone tremendous changes. Nowadays, agility and efficiency are highlighted more during software development. For basic functions, one usually gives priority to reuse existing open source software, and developers only need to make necessary extensions and improvements. This model can effectively reduce the cost of software development, accelerate software iteration, lower the development threshold, and expedite the response to demands. The annual report of Octoverse (https://octoverse.github.com/, which collects the activity data of personal developers in Github and describes the data from multiple dimensions) shows that by the end of 2019, on average, there had been more than 3.6 million open source repositories dependent on the top 50 open source projects, 180 package dependencies per open source project, and more than 350,000 developers participating in the top 1,000 open source projects, with a total contribution of more than 5 million times. In addition, more than 1.3 million developers contributed to open source software for the first time in 2019. The above data reveals that in the open source collaborative mode, software dependencies will become more common and complicated.

Operating system refers to the collection of system programs that manage computer hardware and software resources, including kernel and other system tools[3]. The operating system distributions derived from the Linux kernel (such as Ubuntu, CentOS and Android) are collectively called Linux distributions. They organically integrate many types of open source software with Linux kernel in the form of software packages to meet the different needs of end users[4]. Data from DistroWatch (https://distrowatch.com/, which collects and explores the latest information about open source operating system distributions) reveals that for the common Linux distributions, tens of thousands of software packages need to be maintained to support the functionality and ecology of one version (for example, Ubuntu 18.04 involves 29,207 software packages, and Debian Unstable involves 32,453 software packages). Even a compact system constructed through tailoring contains nearly a hundred software packages. Without the help of tools, professional knowledge and detailed preparations are required to install the software package according to the correct sequence and dependency. With regard to this problem, as a necessary component of the Linux distribution, a package management tool splits or merges open source software into different packages according to functions and maintains their dependency, so as to effectively reduce software management burden to the user. What's more, to help users to build customized Linux distributions, as the core component, the package management tool helps users to clarify dependencies between software modules that need to be added. In addition to build tools based on package management tools, the Linux community has launched the Linux From Scratch (LFS) project, which aims to guide users to build operating systems from scratch and provides package dependencies by relying on package management tools to build different systems. LFS derives the Automated Linux From Scratch (ALFS) project and provides users with automated build tools. In addition, more advanced build projects for customized Linux distributions (such as Yocto) have emerged to help Linux distribution builders shield against the differences in underlying hardware architecture.

As mentioned above, in the open source collaborative model, the distribution of Linux operating system contains complex software dependencies, which makes its reliability not only depend on the kernel itself, but also on other open source software that build the whole ecosystem. Although package management tools can maintain dependencies among software packages, the reliability of these relationships relies more on manual management and maintenance. Based on this information, the reliability evaluation of large-scale and complex foundmental software and its ecology is greatly limited in the open source collaborative model. Accordingly, the existing system build tools also only focus on the build function itself, and barely consider the reliability of the final operating system and its ecosystem. Reference [5] also focused on such software organization and called the complex network formed in this form the open source supply chain and put forward three challenges facing open source supply chain:

(1) The learning cost of individual developers is further increased.

(2) Group collaboration is more complex.

(3) The threat to ecological sustainability continues to grow.

In addition to these challenges, when building an operating system distribution, the supply chain reliability of open source software also faces specific challenges related to the software itself, namely

(1) Diversity in software sources, licenses and suppliers;

(2) Fluctuation and conductivity of software quality;

(3) Supply interruption caused by intellectual property litigation and export control;

(4) Sustained maintenance and development.

These challenges are dominating factors affecting the reliability of current open source operating systems and their supply relationships.

With regard to these challenges, this paper proposes a knowledge-based management method for software supply reliability in the open source collaborative model. Firstly, the ontology for the open source software ecosystem is designed, and the knowledge graph of open source software is constructed to achieve the extraction, storage and management of knowledge. Driven by the knowledge contained in the open source software graph and combined with the traditional management methods of supply chains, this paper proposes a set of reliability management methods for open source software supply chain, which constitutes a supply chain management system for the open source software to respond to the above challenges. Section 1 of this paper expounds the overall research background; Section 2 introduces the achievements in the related fields; Section 3 elaborates on the supply chain system of open source software proposed in this paper. In this section, the definition of open source software supply chain is introduced first. Then, the whole system architecture, data management, construction process, and reliability management methods are described. In addition, build tools for the operating system based on software supply chain are introduced, and the advantages over the existing build tools (LFS and Yocto) are indicated. Section 4 focuses on the prototype system implemented to verify the method proposed in this paper. It also takes the native operating system for RISC-V as an example to test the current system functions and evaluate the reliability of open source software supply chain for the maintenance of the operating system. Finally, the whole paper is summarized, and future improvements and research directions are discussed.

1 Related Work

1.1 Supply chain and its management

The concept of supply chain was first put forward in the field of enterprise management, which can be traced back to "Material Requirement Planning (MRP)" in the 1960s. Since the production capacity of enterprises was low at that time, the contradiction between supply and demand mainly concentrated on resources. MRP was proposed to solve the planning problem between the inventory of raw materials and the production volume of product parts, with the minimum investment and critical path as its basic starting point. With the increasingly fierce competition among enterprises, the scope of their own resource management is also developing towards a broader and more refined direction. What's more, the material-oriented management in MRP can no longer meet the demand. Therefore, enterprises further associate materials with resources including capital, manpower and equipment for more comprehensive planning and control, which makes MRP evolve into MRPII, namely "manufacturing resource planning". With the introduction of information technology in the 1990s, a new enterprise management plan, "Enterprise Resource Planning (ERP)", was put forward, becoming a standard for large-scale enterprise management. As the core of ERP, supply chain management is mainly used to help companies clarify business processes and effectively solve the common problems such as overlap of institutional personnel and low resource utilization in traditional enterprise management.

Nowadays, supply chain has many different definitions according to different target needs and application environments. Among them, the definition given by British supply chain management expert, Martin Christopher, in 1998 has a high degree of public recognition. He defined the supply chain as a network composed of multiple organizations. In this network, the organizations are interconnected with the relationship of upstream and downstream, and they contribute to the end customer products in the form of products or services in different production activities or processes[6].

Combined with the definition of supply chain, it can be abstractly regarded as an integration method of production materials. It will collect data from all parties in the chain as comprehensively as possible, so as to facilitate overall analysis and coordination. The maintenance of supply chain data and the optimization of supply plans can be collectively referred as supply chain management. The definition of supply chain management proposed by Ellram has been widely recognized, namely that supply chain management is the material and product flow that integrates control and planning between suppliers and consumers[7]. The task of managing the supply chain system involves many aspects, while the core task mainly involves two aspects, namely performance management and risk management. Simchi-Levi accurately describes the supply chain performance management: Supply chain management is to effectively integrate resources including suppliers, manufacturers, warehouses and shops through a series of methods, so that enterprises can accurately control the quantity of goods produced and distributed, while ensuring that necessary resources appear in the right place at the right time, so that the supply chain system can minimize the overall overhead as well as meet service needs[8]. The performance of a supply chain system can be quantitatively described by evaluating the relationship between total cost and final output.

The main purpose of supply chain risk management is to cope with the development trend of industry. These trends include more outsourcing, lower supply base, more accurate control time and shorter life cycles of products, greatly increasing the risk of an enterprise supply chain system[9, 10]. To make the risk of a supply chain system controllable, supply chain risk management will customize a series of strategies to identify, evaluate, process and monitor risks[11–13]. As the primary task, risk identification is the basis for carrying out other tasks[14]. At the moment, many risk-identification-related strategies have been proposed in various studies, and some of them have been verified in practical applications. Risk assessment is usually based on data or expert experience[15]. Risks in a supply chain system usually do not appear independently. Therefore, it is necessary to comprehensively consider the relationship among multiple risks and assess their risk levels according to the strategic model, namely the priority of processing[16]. The specific method for risk treatment depends on the environment of the supply chain, which can be roughly divided into acceptance, avoidance, transfer, sharing and mitigation[17]. Since the risk of the supply chain is not static, it needs to be monitored at all times to discover the development trend in the risk and use it as the basis for adjusting the processing strategy. Accordingly, the monitoring of risk is not only highly dependent on the results of the first three tasks, but also requires a certain real-time performance[18].

The goal of this paper is to maintain a reliable open source software supply chain and provide technical support for building a reliable open source operating system independently. Combined with the supply chain management methods described above and the description of the reliability risk characteristics of the open source software supply chain in Section 1, this paper will highlight how to realize the reliability risk management of open source software supply chain. In future work, the performance of supply chain management will be improved under the premise of controllable risks.

1.2 Knowledge graph

Knowledge graphs do not have a clear definition. They can be mostly mixed with the concept of "ontology" in information science[19]. As early as 2012, Google first used the term "knowledge graph" in the technical introduction to its information retrieval system[20], and it has been extensively used in related studies since then. It has a widely recognized definition as a collection of descriptions of entities. These entities have internal associations, which can be objects, events, specific situations or abstract concepts in the real world. To be specific, (1) the descriptions must have a unified structure to ensure that both human and computers can process them in an efficient and clear way at the same time; (2) the descriptions form a network and complement each other, so that each entity is part of the description of its neighboring entities[21].

In general, a knowledge graph is composed of the following parts:

- entity: a specific case or object in the knowledge network;
- category: a collection of entities sharing a certain characteristic or concept;
- attribute: the characteristics describing the entity;
- relationship: the connection mode between categories or entities;
- event: generated when a property or relationship changes.

In addition, it also has the following characteristics:

- supporting structured query language access;
- maintaining mesh data in the form of graphs;
- supporting data understanding and reasoning and the maintained data with formal semantics;
- possessing logical formalization, supporting the generation of new information, forced consistency and automatic analysis;
- having a dynamic nature, with the maintained data supporting automatic or manual continuous integration and update.

This paper chooses an entity graph as the starting point to maintain open source software supply chain, constantly expands the content, and finally builds a knowledge graph dedicated to the field of open source software, laying the foundation for subsequent work.

1.3 Common build methods and tools

Flexibility and customization are major features of open source software and open source operating systems. However, with the Linux system as an example, the complexity of an open source operating system results in more difficult customization, and accordingly, many open source projects have aimed at solving this problem. Due to space limitations, two representative projects, LFS and Yocto, will be highlighted in this section.

1.3.1 *LFS*

The purpose of the LFS project is to guide users to build a customized Linux operating system step by step from the source code. It is expected to help users understand the internal working mechanism of the system in this process. In nature, LFS can be regarded as a framework where users can add and delete functions according to their own needs and finally build a system with a higher degree of compressibility and customizability. In addition, the powerful flexibility of LFS can also help users add customized security components to enhance the security of the final system. In light of the higher versatility of the entire project, LFS has several other sub-projects.

- Beyond Linux from scratch (BLFS): Based on LFS, it guides users to configure and build a Linux system with richer functional software packages, such as a graphical interface (the system generated by LFS does not include a graphical interface).
- ALFS: As mentioned above, this project ais to provide an extensible system builder and a package installer for automatic construction of the system.
- Cross Linux from scratch (CLFS): Based on LFS, it guides users to build Linux operating systems with different frameworks through cross-compilation technology.

As such, the significance of education and guidance of LFS and its subprojects is far greater than that of engineering practice, and the project itself clearly states that the operating system built by users according to the guidance is not the most concise one. To build a system image meeting engineering requirements, users need to not only fully master LFS and flexibly apply the framework provided by LFS, but also acquire enough system knowledge to ensure the correct configuration and construction. Although there are engineering-oriented projects such as ALFS, the development of ALFS is not as expected due to the low community vitality. With regard to software package dependencies (supply), LFS only lists a limited number of software packages as examples. However, more software packages and their relationships need to be explored and managed by users themselves, and the management and verification of supply relationship reliability are hardly observed. If the built system is directly put into production, a great potential hazard will be posed to reliability.

1.3.2 *Yocto*

The original intention of this open source project is to help users build a customized embedded operating system based on Linux. Yocto is originated from the OpenEmbedded project, while the latter focuses on providing the construction system. At the point, the construction system in the Yocto project continues to use the construction system provided by the OpenEmbedded project, which is jointly maintained by both communities. Flexibility is most valued in the Yocto project. The project itself is composed of three important components, namely tool library, build template distribution (named Poky) and OpenEmbedded build system mentioned above. The rich tool library can help users realize functions, including automatic construction and testing, multichip support, compilation process verification and information verification of embedded system components, and also support the sharing of tools among users. The build template distribution is a complete build sample teaching users to use Yocto. In addition, by customizing and modifying the content of the template, users can also build a customized system distribution.

Before using Yocto, users need to first define configuration details including the target architecture, construction strategy and patches. After that, the build system will obtain the target source code according to the configuration, complete the local compilation and build according to the strategy, and finally package it into the specified package format (such as deb, rpm or ipk). When all the software packages have been built and passed all the inspection items, the build system will package all the software packages into a customized software source. At last, the system image of the target root file is created on the basis of the software source.

The above process demonstrates that the early user-defined configuration is crucial to the entire construction process. Yocto abstracts the concepts of recipe and layer and encapsulates the construction configuration of different functions to simplify feature customization and function multiplexing; it still has a steep learning curve notwithstanding. In addition, it is worth noting that the dependencies (namely the supply relationship referred in this paper) among the software packages described by the build configuration in Yocto need to be customized by a user or directly inherited from an existing configuration shared by other users. However, effective tools are waiting to be developed to monitor and inspect the reliability of supply relationships.

2 Open Source Software Supply Chain

2.1 Definition

As mentioned above, the open source model is prevailing, and IT industry giants have invested massive manpower and financial resources in open source, which also has won them many benefits such as market recognition and technical dominance. For example, Microsoft used to be famous for closed source software including Windows and Office. In the past two years, it has built over 3,200 open source projects on an open source social networking site, GitHub, attracting more than 4,300 developers to participate in them and producing up to 20 mature open source software tools. In these circumstances, the mode of software development has also changed from independent development by a single person/organization to multi-person/multi-organization cooperation. With the support of modular software design, the development model of building one's own customized software based on the open source software modules released by others has gradually become the mainstream, and the open source operating system is a typical product of this model. Therefore, according to the special production model of open source software and the definition of supply chain in Section 2.1, it can be concluded that the core of open source software supply chain is regarding the dependencies among open source software modules as a supply relationship built on the operating system. For a more intuitive description, the following concept mapping is constructed:

- commodity: the software delivered to end users, which refers to the large-scale complex operating systems and related software packages in this paper;
- manufacturer: an individual or organization that designs and implements open source software modules;
- supplier: the company or individual providing open source software services;
- warehouse: centrally hosting of open source software.

As indicated by the concept mapping, compared with the traditional software dependency model, open source software supply chain can describe the composition of the software more comprehensively. Besides the modules that constitute the software itself, module developers, maintainers and access sources are also introduced. The original intention of building an open source software supply chain is to solve the problem of open source software reliability in the current model. From the new perspective of supply chain, the composition of large-scale and complex software can be inspected, so that the potential reliability risk of software can be found to deal with the challenges in Ref. [5].

Definition 1 (**Open source software supply chain guaranteeing operating system reliability**). An extended finite automaton $A = (Q, O, \Sigma, \rho, q_0, F)$, where

- $Q = \{S_1, S_2\}$ is a finite set of states, with S_1 representing a reliable state while S_2 an unreliable state.

- $O = O_1 \cup O_2 \cup \cdots \cup O_n$ is the set of software packages provided by each Linux distribution, with O_n representing a certain Linux distribution.
- $\Sigma = \{\omega_i | \omega_i \in O_j\}$ is a finite set of inputs, with ω representing an open source package that belongs to the distribution O_j.
- Reliability management method, $\rho : \Sigma \to \{0, 1\}$, judges whether the software package is reliable or not.
- State transition function is $\delta : Q \times \rho(\Sigma) \to Q$, and the state transition relationship is illustrated in Figure 1.
- $q_0 = S_1$ represents the initial state.
- $F = \{S_1\}$ indicates the set of accepted states;
- $\mathcal{L}(A) = \{\omega | \rho(\omega) = 1, \omega \in O_j\}$ refers to a reliable open source software supply chain for a Linux distribution O_j.

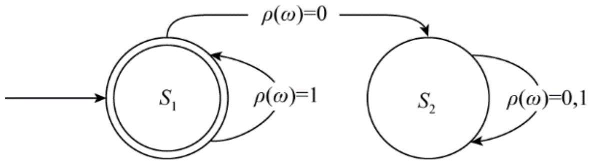


Figure 1 State diagram of supply chain system of open source software

According to Definition 1, the problem studied in this paper is converted into designing a reliability management method ρ to screen out a subset of software packages that meet the output state set F from the package set Σ so as to form a reliable open source software supply chain for Linux distributions. In light of this goal, the accumulated experience of traditional supply chain management and the rich expressivity ability of knowledge graphs are combined to build an open source software supply chain system with the open source software graph as its core. The overall architecture is shown in Figure 2, and the responsibilities of each component are as follows:

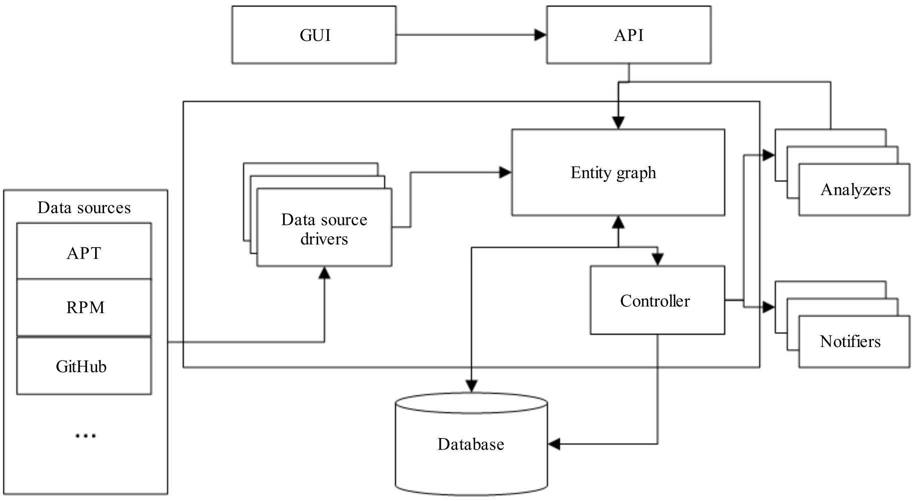


Figure 2 Architecture of supply chain system of open source software

- Graphical User Interface (GUI): It intuitively displays the supply chain system of open source software to help users to obtain rich information, including inter-dependency diagrams and reliability analysis charts of open source software.

- Application Programming Interface (API): It provides the supply chain system of open source software in the form of service interface, such as supply relationship acquisition and reliability analysis of open source software.
- Data source: the system takes the software package manager as the data source to extract the supply relationship among pieces of open source software. At the same time, it extracts the supplementary information of open source software through upstream warehouses including GitHub, such as evolution information of the software itself and author information, thereby enriching the data information in the supply chain system.
- Data source driver: To collect data from different data sources, it provides different drivers to verify the collected data and converts them into a unified format for system storage and management.
- Entity graph: As the core data structure of the system, it maintains all data information in the form of entity graphs. In addition to open source software, the entities also include personnel and companies (or organizations). The relationships among entities are also diverse, including supply, development, maintenance and subordination. The supply relationship of the entire open source world is modeled in an intuitive form. This paper will introduce this part in detail in Section 3.2.
- Controller: It is responsible for coordinating and distributing background tasks, including data analysis and status monitoring.
- Analyzer: It is implemented in the form of plug-in and responsible for performing specified analysis tasks, such as software reliability analysis.
- Notifier: It is implemented as a plug-in and responsible for sending notifications to a specified location when the system generates events (such as finding an abnormality or potential risk).

2.2 Data management

This section focuses on the data management implementation in a supply chain system of open source software. As mentioned above, the system organizes and maintains the information related to the supply chain in the form of entity graphs. Compared with other data organization forms, entity graphs reflect the relationships among entities more intuitively and contain more abundant semantic information, making it easier to understand data itself and derive new facts. With the increasing data volume and the continuous enrichment of data content, the ultimate goal is to build it into a knowledge graph about open source software supply chain.

2.2.1 *Ontology design*

Ontology design is the first step in the construction of a knowledge graph. In this step, the types of entities contained in the graph need to be abstractly defined to sort out the relationships between them, and finally they are organized in the form of graphs. In this paper, knowledge graphs are limited to the field of open source software to restrain the scale and complexity of the graphs. As Figure 3, five entity types are defined.

- Software: It is the core entity type of the open source software graph. According to the collaborative model of open source software, it can be divided into two entity subtypes: package and operating system. The relationship between software and the software entity type is “isA”. Package is the entity with smallest granularity in software, while operating system is composed of multiple packages. Therefore, there is a “hasComponent” relationship between them. The supply relationship among packages is described by “hasSupplier”, and the evolution relationship of packages is described by “hasPreversion”.
- License: This entity type describes a specific open source license. The software must “obey” one or more open source licenses. License restricts the way of using open source

software and has a potential impact on the availability of open source software. As a result, there is a potential reliability risk in the open source software supply chain.

- Location: This entity type describes a specific location and has a one-to-many relationship with an organization or person. Because any piece of software is maintained by at least one organization or individual, the software package itself has the attribute of location. Generally, organizations or individuals need to abide by local laws. However, when a country limits the level of openness of open source software to its organization or individual owner by legal means due to political factors, the existing supply relationship among pieces of open source software is highly likely to be damaged.
- Organization: This entity type describes a specific organization, which can be an open source organization or a company providing open source software. They have actual control over some open source software, with a relationship of "hasController". At the same time, they need to comply with local laws and regulations when "locateAt" a specific location;
- Human: This entity type describes a specific developer or maintainer who can contribute to open source software either in the name of an individual or the name of a specific organization ("hasEmployee"). Therefore, there is a "hasContributor" relationship between them and software. At the same time, a person may have actual control over a piece of open source software in his own name, so there may also be a "hasController" relationship between them and the software.

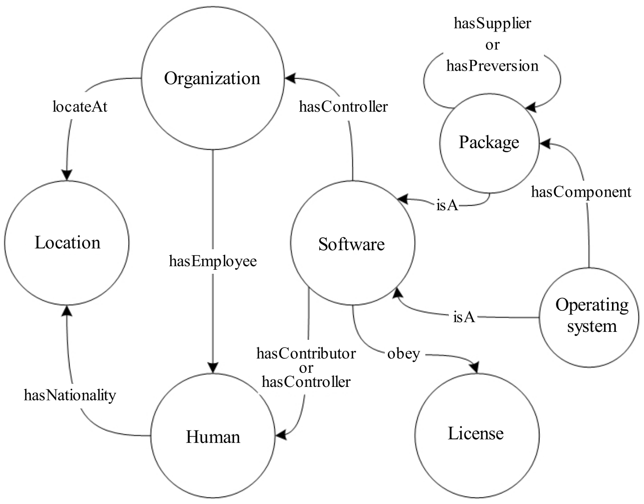


Figure 3 Ontology design of knowledge graph of open source software

For the sake of brevity, Figure 3 only shows the basic entity types and their relationships, omitting the attributes contained in each entity type. This part will be further described in Section 4.1.

2.2.2 *Build process*

The construction of open source software knowledge graph proposed in this paper has two main steps, as shown in Figure 4.

(1) Basic graph construction

In this step, with existing package managers including RPM (a common package management tool, a core component of mainstream operating system distributions such as Red Hat, Fedora, openSUSE and CentOS) and APT (a common package management tool, a core component of mainstream operating system distributions such as Debian and Ubuntu) as the data source and the software packages providing core functions in the target operating system as the "seeds", the basic supply relationship graph can be obtained by mining the supply chain of the "seed" package, and the basic attribute information can be filled for each package instance at the same time. It should be noted that whether a software entity already exists will be judged during this process. If a software entity exists, it should not be added repeatedly. This basic graph only contains the basic information of the software package, such as "name" and "size", while the information of attributes, such as "features" and "maintainers", cannot be acquired. Therefore, the "knowledge quantity" of the whole graph need to be improved and expanded through the second step.

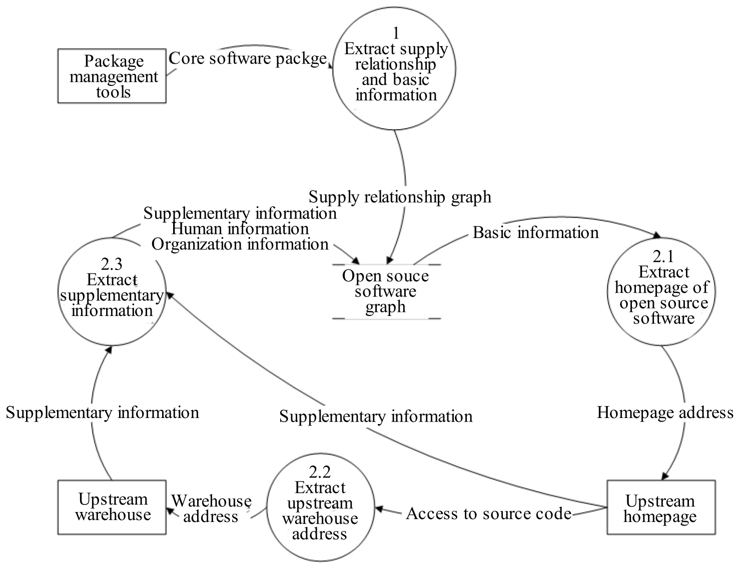

Package management tools
Core software packge
1 Extract supply relationship and basic information
Supply relationship graph
Supplementary information
Human information
Organization information
Basic information
2.3 Extract supplementary information
Open souce software graph
2.1 Extract homepage of open source software
Supplementary information
Supplementary information
Homepage address
Upstream warehouse
Warehouse address
2.2 Extract upstream warehouse address
Access to source code
Upstream homepage

Figure 4 Data flow diagram for constructing the knowledge graph of open source software

(2) Expand supplementary information

This step is based on the basic graph generated in Step 1, which can be expanded in three substeps.

(a) In the basic graph, each entity type of Package is traversed, and the homepage address of the upstream software is extracted.

(b) The homepage of each upstream software is visited and the acquisition method of source code is extracted, namely the warehouse address of open source code, such as the warehouse address in GitHub.

(c) The upstream code warehouse is visited to extract the historical version information of the software, including the version number, cumulative submission information (self-evolution history), developer information, version characteristics and known problems of each version. If some particular information is incomplete, it is necessary to extract supplementary information from the homepage in the previous step.

As stated in Section 2.2, an important feature of the knowledge graph is dynamic nature. For the knowledge graph of open source software, the process described in Figure 4 can be understood as a complete updating process. With the continuous supplement of new "seeds", the "knowledge quantity" of the graph will continue to expand, providing sufficient data to support the reliability management of the whole open source software supply chain. Meanwhile, it lays a solid foundation for future work, such as providing intelligent self-construction service of an operating system.

2.3 Reliability risk management

As mentioned above, this paper is not concerned with the reliability of a single piece of software in the general sense, but the ecological reliability risk of open source software introduced by the three challenges described in Ref. [5] in the current open source software model. Combined with the four main tasks of supply chain risk management, namely identification, evaluation, processing and monitoring (it is believed that in the supply chain risk management of open source software, the boundary between identification and evaluation is fuzzy, so they are merged into one task in this paper), relevant strategies and functions are designed and implemented in the system. This helps to solve the three challenges faced by open source software supply chain, while achieving the risk management of reliability.

2.3.1 *Identification and evaluation*

For the identification of the reliability risk in the supply chain, a measurement model of open source software reliability is built to enable the system to quantitatively describe software reliability. The specific measurement items and description are shown in Table 1. The measurement model is mainly divided into two parts, i.e. software package and the supply chain it depends on. This model is also a direct representation of our recognition of modern software structure, namely that reliability not only depends on some indicators of the software itself but its complex dependencies. In the model, except for conventional measures such as community activity and open source license, the location attributes of open source software including the nationality distribution of contributors, maintainers (or owners) and other dependencies in the supply chain are also highlighted to more intuitively describe the potential reliability risks introduced by political factors. At the same time, the substitutability of software packages is also considered. Apparently, irreplaceable software packages will become a weak link in the entire supply chain.

In the proposed system, the ontology design in Section 2.2.2 reveals that the basic items in the measurement model such as the ownership of software packages and the open source license followed have already been in the open source graph, while other advanced data such as substitutability and distributions need to be obtained based on the processing of basic data. Finally, based on the data provided by the measurement model, the software reliability can be quantitatively described in the form of scores. As far as we know, there is no relevant public benchmark dataset. Therefore, considering the objectivity and impartiality of our system for reliability evaluation of open source software, in the current system, multiple methods are employed to get the final reliability score. Furthermore, the identification and assessment of reliability risks of open source software supply are realized on the basis of this score.

Based on the open source graph and measurement model, the data flow described in Figure 5 is designed and implemented to produce the software reliability score. In the current system, the reliability score is limited to 0–5 points, and the middle score is represented by decimal with two decimal places reserved, such as 3.47. The calculation process of the reliability score is as follows:

$$P = f(e) \tag{1}$$

$$P_{\text{init}} = \text{similar}(\text{attr}, \text{cluster}(G)) \tag{2}$$

$$P = \frac{n^{i} a_i p_i}{n} \tag{3}$$

$$P_{\text{new}} = \frac{P_{\text{old}} + P_{\text{updated}}}{2} \tag{4}$$

Table 1 Reliability metrics of open source software

Metrics			Description
Software package analysis	Activity	Update frequency	Update frequency of software
		Response speed to problems	Response speed to feedbacks or suggestions
		Number of contributors	Number of participating contributors
	Locations of contributors		Nationality distribution of participating contributors
	Substitutability		Substitutability by software packages with similar functions
	Types of open source license		Open source license followed
	Ownership	Nature	Belonging to an organization or an individual
		Nationality	Nationality of an organization or an individual
Supply chain analysis	Reliability distribution		Reliability distribution of software packages in supply chain
	Location distribution of software packages		Nationality distribution of software packages in supply chain
	Distribution of open source licenses		Distribution of open source licenses in supply chain

(1) Update Path 1 is illustrated as Eq. (1). where e represents extracting relevant event information from the Internet, including information directly related to the software (for example, it is clearly stated that the software is restricted or no longer maintained, or in fact it has already been in no maintenance) and political factors of the nationality of software ownership (such as export controls, clear restrictions on the usage by organizations or individuals in certain countries, or clear restrictions on certain purposes). f indicates the specific calculation method of scores. Since the calculation process can be reused with monitoring tasks, it is described in Section 3.3.3 in detail. In addition, Eq. (4) demonstrates that when the supply relationship of software includes the software package affected by the event, its reliability score will be influenced accordingly.

(2) The basis for reliability scores obtained only through Path 1 will be limited. The packages without initial scores will be clustered according to the data provided by the measurement model, as shown in Eq. (2), which makes the software packages with similar "reliability characteristics" be classified into one group. Then, the software without score can be given initial score based on similarity.

(3) In the case that the data that can be obtained through Path 1 is extremely scarce, the reference value of the initial score given through Path 2 is limited. Therefore, in Path 3, users are allowed to score software reliability according to the measurements provided by the system, and finally the scores from all users are synthesized to get the reliability score of software. As Eq. (3), considering the different professional levels of users, the weighted average is adopted when all the scores are synthesized. The professional level of users is positively correlated with the weight of their scores. In this way, the reliability evaluation that meets the standard of most people can be obtained from the public cognition of ecological reliability, and the reference value of the whole dataset can be enhanced.

(4) According to the measurement model, the reliability of the software itself depends on the software reliability in its supply chain. Therefore, when the reliability score of the software is updated in other paths, it is necessary to update all pieces of software in the supply chain of the software. Starting from the software whose score is updated, the open source graph is traversed reversely along the supply relationship and the reliability score of the software passed by is updated. Considering that the reliability risk does not decline or rise with a longer supply path,

the new reliability score by arithmetic average is calculated. As Eq. (4), this can not only ensure the simplicity of calculation but express the spread of its influence intuitively. Furthermore, it can reflect the consequences of multi-risk superposition. It is stipulated that only the first three updating paths will trigger the update process to ensure that each update process converges.

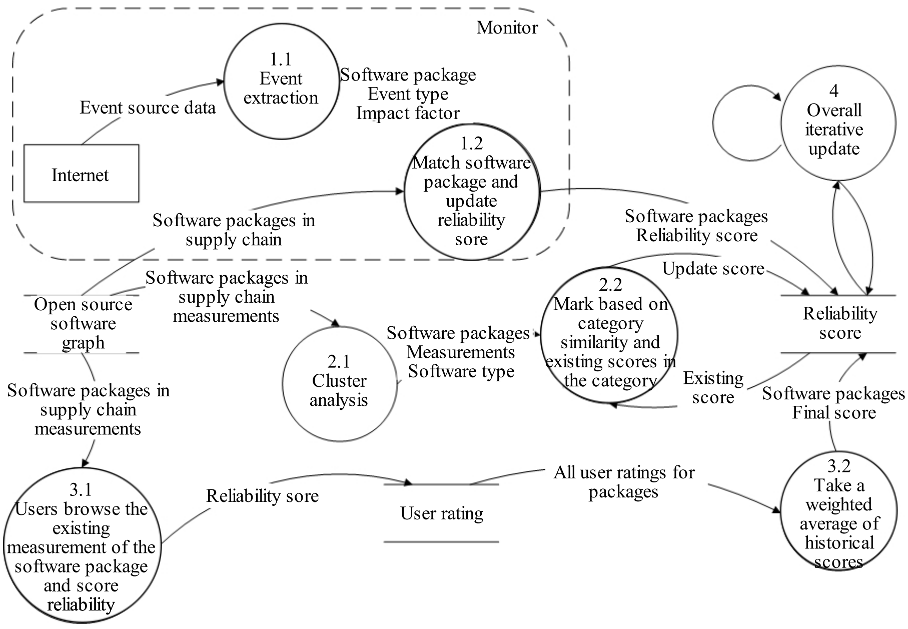

Monitor
1.1 Event extraction
Event source data
Internet
Software package Event type Impact factor
1.2 Match software package and update reliability sore
Software packages in supply chain
4 Overall iterative update
Software packages Reliability score
Update score
Open source software graph
Software packages in supply chain measurements
2.2 Mark based on category similarity and existing scores in the category
Reliability score
Software packages Measurements Software type
2.1 Cluster analysis
Existing score
Software packages Final score
Software packages in supply chain measurements
3.1 Users browse the existing measurement of the software package and score reliability
Reliability sore
User rating
All user ratings for packages
3.2 Take a weighted average of historical scores

Figure 5 Data flow diagram for generating evaluation of software reliability

2.3.2 *Processing*

As mentioned above, processing of supply chain risks can be roughly divided into acceptance, avoidance, transfer, sharing and mitigation. Considering the particularity of open source software supply chain, risk treatment solutions with regard to acceptance, avoidance and mitigation are proposed in terms of the level of reliability risks. The specific risk levels are illustrated in Table 2, i.e. low, medium and high. Packages with scores above 3.5 have lower risk and lower processing priority, so a temporarily-accepted processing method can be adopted. Software packages with scores between 1.5 and 3.5 are at the medium risk level, and necessary measures are required to mitigate the risk. Software with a score less than 1.5 has a high reliability risk and must be processed with the highest priority to completely avoid risks.

Table 2 Risk level of reliability

Reliability score	Risk
(3.5, 5]	Low
(1.5, 3.5]	Medium
[0, 1.5]	High

In the system proposed in this paper, direct measures will not be taken to deal with the captured reliability risk, but decision support for improving the reliability of software and its supply chain will be provided. Specifically, the system will recommend software packages whose risks need to be mitigated in the software supply chain from low to high reliability scores while displaying the reliability measurements of the software. In accordance with the scoring mechanism in Section 2.3.1 and the monitoring mechanism in Section 2.3.3, the system will re-evaluate the reliability of the software according to the optimized software packages, thereby reducing the reliability risk of the supply chain where the optimized packages are located.

2.3.3 *Monitoring*

A supply chain system is a dynamic system. The status of each node in the chain will change dynamically. Compared with general products, software has the characteristics of fast iteration, so the change period of open source software supply chain will be further shortened and there will be more uncertain factors. Therefore, the timeliness of reliability risk management is particularly important. If the events that affect the reliability risk cannot be captured in time, the system will make a wrong assessment of the risk. Combined with the system architecture described in Section 3.1, the reliability risk monitoring process is implemented on the basis of multiple components as shown in Figure 5 (the data flow is indicated by the dotted line in Figure 5). The whole process is divided into three stages. In the data acquisition stage, the corresponding drivers are implemented for different data sources (Table 3), which are respectively responsible for collecting relevant data and saving them in the form of time series data. In the data analysis stage, the analyzer program extracts events based on historical data and the latest captured data (detailed in the following). If a valid event is found, the event processor will be triggered to distribute it to the corresponding processing flow. In the current system, the reliability evaluation sub-process will be triggered, and the feedback of evaluation results will be sent to the monitoring time series data to optimize the extraction accuracy of the sub-process.

In the current system, events are extracted based on rules. In light of the proposed measurement model, six types of risks are summarized in Table 3. The table also shows the relevant attributes and data sources in the measurement model corresponding to each risk. As mentioned above, the corresponding data driver is implemented according to the different data sources of risks. Therefore, it is convenient to label data of each risk to identify the risk type. The event extraction sub-flow is illustrated in Figure 6. Firstly, the types of risk events are judged based on the label. If it belongs to one of the first two risk types in Table 3, the information extraction is realized based on keywords, mainly including software names and emotion words. The impact factors can be obtained according to the emotion word list prebuilt by the system. If it belongs to one of the last four risks, the software name is also extracted by keyword matching, and then the risk impact factors can be obtained by comparing historical information or calculated statistical data. The specific statistical indicators are displayed in Table 3. It should be noted that the current index is specified according to experience, which can be regarded as the initial value. In the future, the index will be adjusted according to the data collected by the system. The impact factor describes whether the monitored events have a positive or negative impact on the software reliability risk. If the event results can help reduce the risk, the impact is considered to be positive; otherwise, it is regard as negative. For simpler calculation, both positive and negative factors are taken to be 1 without regardless of the influence by different events. With data accumulation of data and system improvement, the impact of each event on software reliability risk will be objectively described by introducing weights in future work.

As shown in Figure 7, the final output of the event extraction process includes the name of the software package, the event type and the impact factor. After the above information is received in the event processing stage in Figure 6, the reliability score is updated according to the following formula:

$$v_i' = v_i + e \tag{5}$$

$$p = \begin{cases} \sum_{n}^{i=1} v_i \times \frac{P_{\max}}{N}, & N > 0 \\ 0, & N = 0 \end{cases} \tag{6}$$

Table 3 Risks to be monitored

Risk	Data source	Related attributes of measurement model	Description
Political risks	News site Social media	Location distribution of contributors Ownership/nature Ownership/nationality	Reliability risks brought by political factors: for example, the US government requires Google to restrict Huawei mobile phones from installing and using its service software because the tension between China and US.
Poor substitutability	Private website User supplement	Substitutability	No software with similar functions, or similar software with insufficient maturity.
License change	Homepage of open source software, social media and warehouse	Type of open source license	Usage restriction by changing the open source license: for example, major open source vendors restrict the usage by cloud vendors by replacing the open source license.
Low update frequency	Open source software warehouse	Community activity/update frequency Ownership/nature	If it is a personal project with no update for more than one week, the risk is greater.
Slow response to problems	Open source software community	Community activity/response speed to problems Ownership/nature	If the feedback is not responded for more than 3 days, the risk is greater for an individual project.
Small number of contributors	Open source software warehouse and community	Community activity/number of contributors Ownership/nature	The software has a potential risk of unavailability if the number of individual projects or contributors is less than 3.

Two variables for each software package, V and N, are maintained to calculate the reliability score of event impact. V is an array, recording the cumulative result of the impact factor after each risk event is captured. The initial value of each item is 1 (to ensure that the calculation result of Eq. (6) is the full score when there is no negative impact). Eq. (5) describes the update rule of cumulative results, where v_i and v_i' are the cumulative value of a certain risk event before and after modification respectively; e is the impact factor of the event captured this time. The other variable N indicates the number of captured events. Eq. (6) depicts the update rules of reliability score affected by events, where p is the final score; v_i has the same meaning as that in Eq. (5); n is the number of risk types; $P_{\max}$ is the value of full score (taken as 5). When N is 0, namely no relevant events have been captured, this approach has no contribution to the reliability score, so p is 0. When N is greater than 0, the total reliability score is obtained by accumulating the cumulative value of all risk impact factors. In order that the final calculated score is within the interval $[0, P_{\max}]$, a regularization term is added, namely $\frac{P_{\max}}{N}$. Apparently, when negative events are captured, the result calculated by Eq. (6) will be less than $P_{\max}$, and the final score is negatively correlated with the number of negative events captured. According to the update rules by the update Path 4 in Section 3.3.1, the events of different risk types captured by the monitoring task will eventually have a positive or negative impact on the software reliability score.

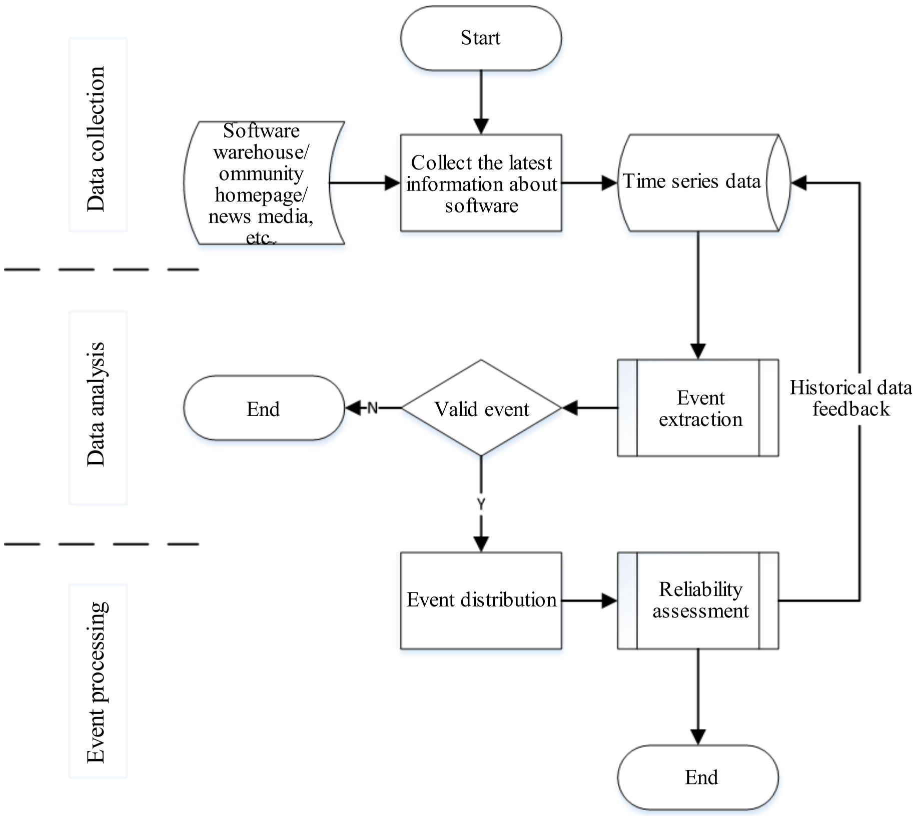


Figure 6 Flow diagram for monitoring risks of reliability

2.4 Building of operating system based on reliable supply chain

Although the open source projects represented by LFS and Yocto for operating system have met the needs of users to build a customized operating system to a certain extent, there are still many deficiencies to be improved, especially in the software ecological reliability of software. With regard to the reliability problem, a construction method based on the implementation of the open source software supply chain is proposed[22]. This method uses the open source software supply chain proposed in this paper to replace the management method of software dependency in existing build tools. Specifically, open source software supply chain has the following contributions.

- Lower learning threshold: The management system of open source software supply chain maintains the supply relationship among pieces of software, including the supply relationship among different pieces of software and that between the upstream source code and the specific packaging format. Through the retrieval function provided by the system, users can easily obtain the supply relationship, and the builder no longer needs to spend much energy to memorize the information, ensuring quick, accurate and efficient work.
- Reduce maintenance costs: The management system of open source software supply chain can display supply chain information in a graphical way, helping users manage the supply relationship more intuitively and improve their work efficiency.
- Enhance ecological reliability: The management system of open source software supply

chain enable users to identify, evaluate, process and monitor the reliability risk through the internal management mechanism, effectively reducing the reliability risk of the software ecology of the target build system and improve its reliability.

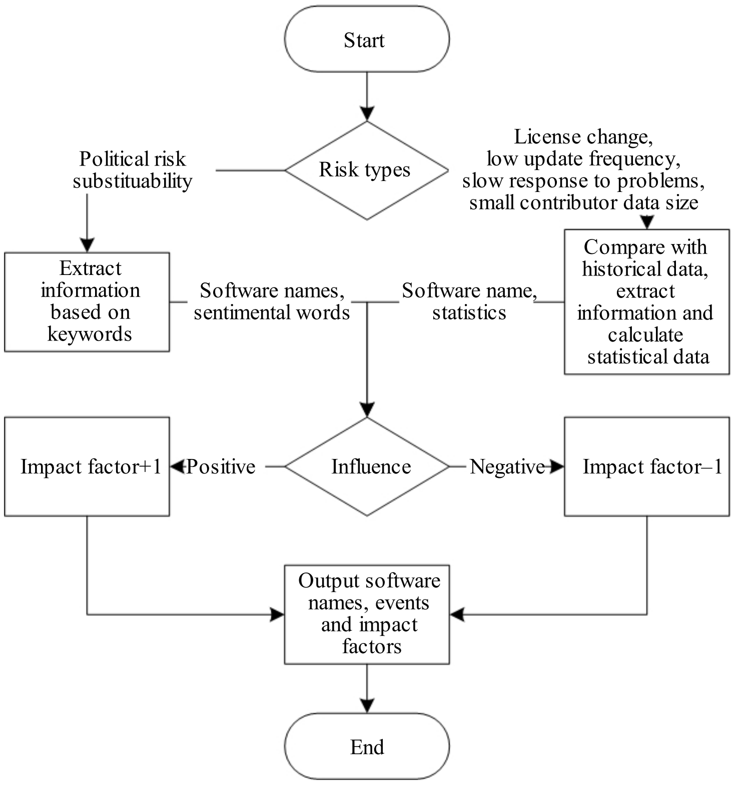

Start
Risk types
Political risk substituability
License change, low update frequency, slow response to problems, small contributor data size
Extract information based on keywords
Software names, sentimental words
Software name, statistics
Compare with historical data, extract information and calculate statistical data
Impact factor+1
Positive
Influence
Negative
Impact factor–1
Output software names, events and impact factors
End

Figure 7 Flowchart for extracting events

3 Experimental Verification and Results

3.1 System implementation

According to the system architecture shown in Figure 1, the system is split into functional components and implemented separately. In this section, some important implementation details will be described. Each component is deployed on a Kubernetes cluster in the form of a container to facilitate management and maintenance. Four servers with the same configuration are used to build the cluster. The configuration of a single server is shown in Table 4.

Table 4 Hardware configuration

Parameter	Configuration
CPU	E5 2690 V4(2 $\times$ 28 thread) 2.6 GHz
Memory	128 GB
Disk	38.3 TB

The entity graph is an important part of the system implementation. According to the ontology design illustrated in Figure 3, the entity graph is mapped as the data structure as shown in Figure 8. Specifically, the object-oriented design philosophy is strictly followed by treating each entity as a class and defining the related attributes and types for each class. In

Figure 8, arrows of different forms are adopted to describe the relationship among classes. Entity and Relation are the basic types; the empty arrow indicates the inheritance relationship, namely Software, License, Organization, Location and Human are inherited from Entity, while Package and operating system are inherited from Software; the dotted single arrow represents the dependency, describing the dependency of class attributes on other classes and their attributes; the solid line single arrow indicates the association relationship, namely that Relation points to the key attribute associating with Entity through from and to attributes, connecting two different Entity examples to show a certain Relation between two Entities.

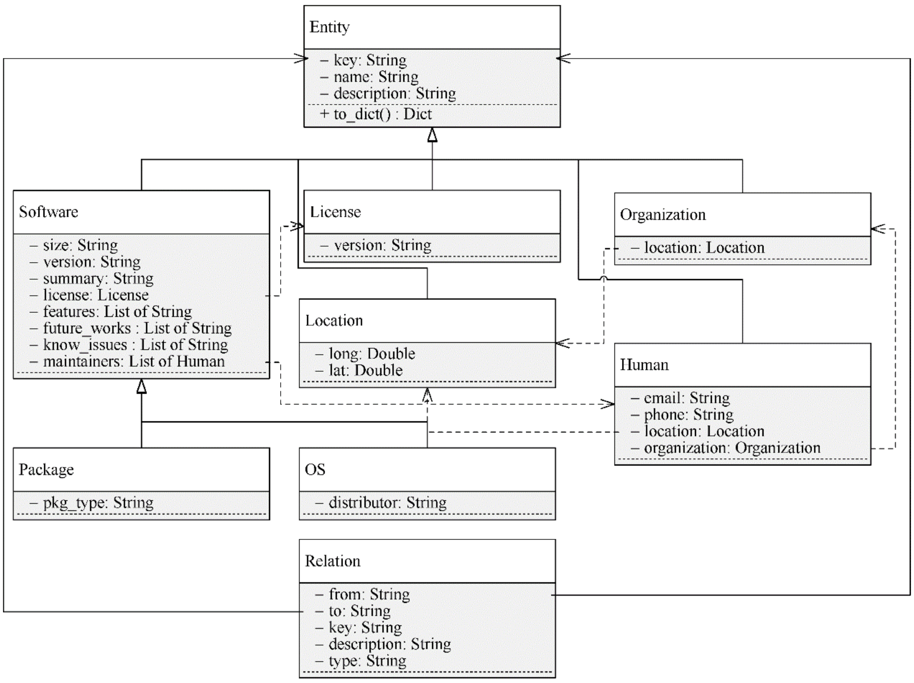


Figure 8 Data structure of knowledge graph of open source software

A graph database is used to save knowledge graph data for system implementation. According to the recommendation of G2 (when the user is looking for the optimal solution, G2 (https://www.g2.com/) provides objective suggestions through multi-dimensional data analysis to help users make decisions), the top two tools which are Neo4j (graph database, first version released in 2007, developed and implemented in Java) and ArangoDB (graph database, first version released in 2011, developed and implemented in C++ and Javascript) are selected among many candidate graph databases. Through further comparison, although Neo4j has a higher degree of adoption by engineers,, its community version (Neo4j is divided into commercial and community versions) is severely limited in horizontal expansion ability in terms of openness. In contrast, ArangoDB is completely open source, which is chosen. Based on the data access interface provided by ArangoDB, a set of interfaces are implemented for other components to retrieve and update entity graphs according to actual business requirements and the RESTful[23] standard.

Due to the huge volume of open source software, it can be expected that the knowledge graph of open source software will contain a large amount of data. The visualization of the open source software supply chain is realized by a graphical interface to reduce the use threshold, thus helping users to explore and obtain relevant knowledge and analyze intuitive data. During the visualization of a supply chain, the integrity of global information and the clarity of local details need to be considered at the same time. However, for the distribution of an open source operating system, the supply chain usually contains hundreds or even tens of thousands of nodes, leading to a sharp conflict with the limited screen size. Considering that every open source software supply chain can be regarded as a Directed Acyclic Graph (DAG)[24], when the supply chain is visualized, the characteristics of DAG are used to realize supply chain layering on the basis of the topological sorting algorithm[25], namely that each screen only needs to show the details of one layer of supply chain nodes, effectively overcoming the capacity limitation of visualizing the supply chain caused by the screen size. At the same time, the global thumbnail of the layered structure will be provided for each supply chain to retain the global information of the supply chain. When users examine the details of a certain level of supply chain, corresponding identification will be made in the global thumbnail to ensure that users obtain complete information.

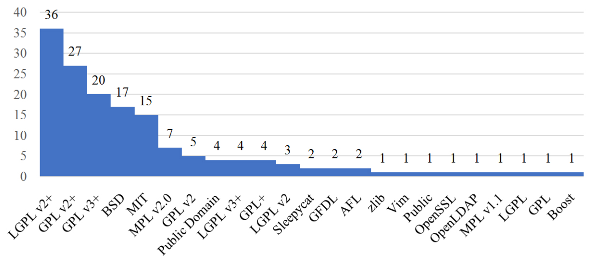
In addition, the controller, analyzer and event processor in the system are all implemented as plug-ins, which allows to extend the related functions of the system in the form of plug-in, thus ensuring the flexibility and expandability of the system.

3.2 Usability verification

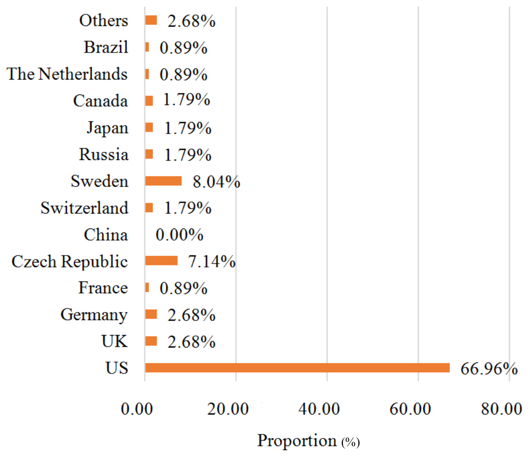
A streamlined Linux operating system built for RISC-V (an open Instruction Set Architecture (ISA) based on the principle of Reduced Instruction-Set Computer (RISC); initiated by the University of California, Berkeley; often interpreted as “open source hardware”) is taken as an example to build and maintain an open source software supply chain, so as to test the system availability. This streamlined operating system takes basic text editing as the use scenario. After tailoring, it is confirmed that only six core software packages, bash, coreutil, passwd, systemd-udev, util-linux and vim-minimal, need to be retained. For convenient retrieval, similar to the build process described in Section 1.2.3, a special software entity is added for the operating system, and these six software packages can be taken as their supply-dependent entities. After the information is input into the system, the software entity of the operating system can be added to the open source software graph. When the software supply chain of the system constructed in this example is extracted, it only needs to retrieve with the entity as the starting point. Finally, the open source software supply chain contains 112 software packages. The relevant statistical data is shown in Figure 9, including the distribution of open source licenses and the locations of software maintainers and contributors.

According to the distribution data of open source license, the adoption rates of LGPLv2+, GPLv2+, GPLv3+, BSD and MIT are relatively high in the supply chain, accounting for 73.2% of the total. Among them, LGPLv2+ has the highest adoption rate. The open source software under the license of this type is allowed to be used directly by commercial software, but it is not allowed to modify the source code. The adoption rate of GPLv2+ and GPLv3+ takes the second place. In general, open source software under GPL license allows to use and modify the source code, but the modified code is not allowed to be released and sold as part of closed source commercial software. The two licenses, BSD and MIT, are more loose, because they allow use, modification and commercial sale. On the whole, most of the licenses used by open source software encourage openness and sharing mainly. At the same time, the unconditional claim is further restricted. For the supply chain reliability of most domestic software products, if the cognition and attitude towards open source are not adjusted as soon as possible, to build a more reliable software ecosystem, these limitations will undoubtedly bring great challenges to the supply chain reliability of open source software in the future.

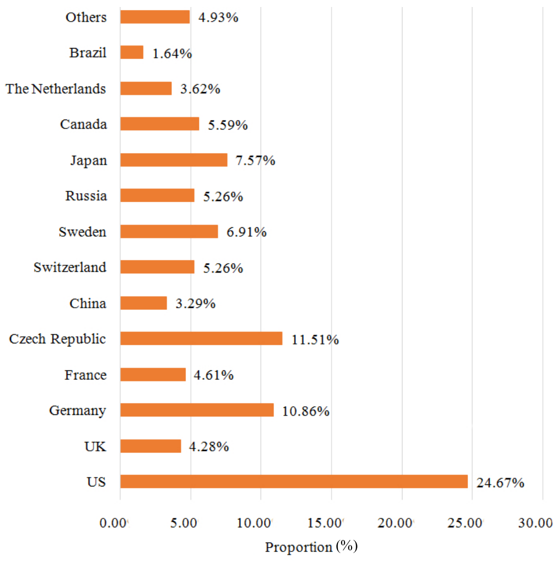
In addition, from the two charts related to location distribution in Figure 9, the proportion



(a) Distribution of open source license



(b) Locations of maintainers



(c) Locations of contributors

Figure 9 Statistics of open source supply chain

of project maintainers or contributors form China is relatively low among open source software packages needed to build basic software such as operating systems. The reason may be the limited understanding of open source of most personnel or related companies due to the late domestic open source culture on the one hand. Moreover, there are also practical problems such as relatively high threshold and high investment cost in basic software development, leading to low overall participation. On the other hand, China's open source contributors may lack discourse in the open source software, and many suggestions and contributions are difficult to be adopted, resulting in low contribution. Consequently, low dominance and participation will bring huge risks to the reliability of the open source software supply chain.

According to the generation process of reliability scores in Section 3.3.1, in this experiment, due to the lack of data related to Path 1, Path 3 is mainly relied on to ensure the objectivity and effectiveness of the data as far as possible. Additionally, four doctors engaged long in the research on basic software reliability and security, ten professionals with master's degree or more than five years of work experience and ten master candidates with more than two years of experience in using open source software were invited to participate in the experiment. All participants scored according to the statistical data provided by the system and their own experience. Considering the different professional abilities of personnel of three types, the weights of 0.5, 0.3 and 0.2 were assigned to their scores respectively. The system finally obtains the reliability score of the supply chain by converging with the mechanism of Path 2 and Path 4. The final score is shown in the blue bar ("before the event") in Figure 10, with 27.68% low risk, 70.54% medium risk and 1.79% high risk. According to the statistical data of supply chain, the scores from participants pay more attention to location distribution and open source license types. The software packages with lower scores belong to organizations or individuals from US and impose large limitations on the open source license type,, which shows that the participators are relatively concerned about political risks and the limitations of open source licenses. Further observation reveals that indicators such as activity and substitutability play a limited guiding role, because the supply chain of the target system differs little in the data of these indicators in the current experimental examples.

For the testing of Path 1 and event monitoring of the system, the event "GNU organization restricting Chinese enterprises to use bash" is injected after the system score is stable. In this event, "GNU" and "bash" are the keywords used by the system for event extraction, and "restricting" is the keyword that affects the event. After the system captures the event, the system adjusts the reliability score of the whole supply chain, with the results in the orange column (after the event) in Figure 10. The results demonstrate that the system adjusts the reliability assessment of the "bash" package node in the supply chain due to the capture of negative events. At the same time, due to the update mechanism of the system, the reliability score of software packages which rely on "bash" has also been modified accordingly.

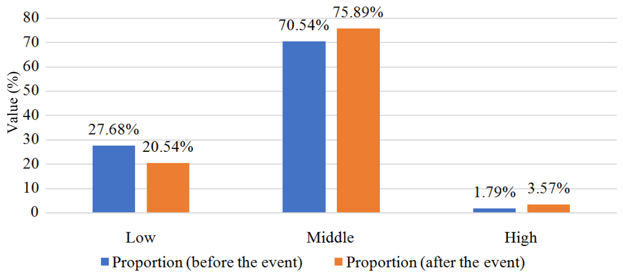


Figure 10 Proportion of reliability scores

4 Conclusions

The core functions and ecology of large and complex operating systems such as Linux distribution and Android are based on a large amount of open source software, which constitutes the supply chain of operating system construction. It turns out that open source software also has the risk of "off supply". If there are problems in the supply chain, the basis of constructing and operating the system is no longer reliable. Therefore, the premise of sustainable and high-quality development of an open source operating system is to ensure the reliability of the open source software supply chain. Based on the traditional supply chain method, this paper introduces a management system of open source software supply chain, constructs the knowledge graph of open source software, analyzes and evaluates the reliability of open source software packages and supply chain from multiple perspectives and makes more comprehensive risk assessment and elimination suggestions, so as to find out the key links for constructing large and complex operating system and expose the vulnerable links of ecology. The method and system proposed in this paper also serve for complex infrastructure software such as big data and cloud platform and lay the foundation of supply chain for the customization and intellectualization of the software. Overall, this paper makes the following contributions:

- Knowledge graph of open source software: Based on the construction method described in this paper, a knowledge graph for open source software is built. With the accumulation of knowledge, the graph will maintain the supply relationship among massive pieces of open source software and software-related entity information, providing a strong data support for related analysis and opening up a new path for the public to understand the world of open source software.
- A management method of software supply reliability: With regard to the open source collaborative model, a set of reliability management methods for open source software supply chain is proposed by constructing knowledge graphs of open source software and combining with traditional supply chain management methods with knowledge as its core.
- A method for constructing a reliability evaluation dataset: Whether in academia or industry, as far as we know, there is no large-scale benchmark dataset to evaluate the reliability of the open source software supply chain. Based on the measurement model and scoring method proposed in this paper, the system will accumulate and generate an effective dataset for reliability evaluation during operation.

Although a large number of research achievements have been witnessed in the research on software engineering management and supply chain management, software supply chain management is still in its infancy as an interdisciplinary field. Limited by the existing results, the system proposed in this paper still requires further improvement. In future work, it is hoped that exploration and research can be made in the following aspects:

- Improve the extraction method of events: The current system uses a keyword-based event extraction method, with limited extraction efficiency and accuracy. In future research, more advanced natural language processing models can be introduced to improve the performance.
- Upgrade the method of reliability evaluation: The current system can accumulate a large amount of effective annotation data while in use. In future research, the accuracy of the evaluation model and the intelligence of reliability evaluation can be further improved according to the contribution value of different dimensions in these data mining measurement models for reliability evaluation.
- Expand the knowledge graph of open source software: The knowledge graph constructed by the current system mainly highlights the reliability of the supply relationship of open

source software. In future research, it can be integrated with the existing knowledge graph for other open source software domains, making the knowledge graph of open source software more universal.

References

[1] Luo B, Ding EY, Liu Q. Software Engineering and Computing: Technical Basis for Software Development, Vol. 2. Beiing: China Machine Press, 2012. 308–309 (in Chinese).

[2] Goševa-Popstojanova K, Trivedi KS. Architecture-based approach to reliability assessment of software systems. Performance Evaluation, 2001, 45(2-3): 179–204.

[3] Tanenbaum AS, Woodhull AS. Operating Systems Design and Implementation. Pearson Education, 2011. 4–5.

[4] Vermeulen S. Linux operating systems: Distributions. http://swift.siphos.be/linux_sea/whatislinux.html#idm3548300567744

[5] Zhou HM, Zhang YX, Tan X. Software digital sociology. SCIENTIA SINICA Informationis, 2019, 49: 1399–1411 (in Chinese with English abstract). [doi: 10.1360/N112018-00319]

[6] Christopher M. Logistics and supply chain management. Journal of Marketing, 1998, 4–5.

[7] Ellram LM. Supply chain management, partnership, and the shipper. The Int'l Journal of Logistics Management, 1990, 1(2): 1–10.

[8] Simchi-Levi PKD, Simchi-Levi E. Designing and Managing the Supply Chain: Concepts, Strategies, and Case Studies. McGraw Hill Professional, 2003. 354.

[9] Colicchia C, Strozzi F. Supply chain risk management: A new methodology for a systematic literature review. Supply Chain Management: An Int'l Journal, 2012, 17(4): 403–418.

[10] Trkman P, De Oliveira MPV, Mccormack K. Value-oriented supply chain risk management: You get what you expect. Industrial Management & Data Systems, 2016, 116(5): 1061–1083.

[11] Neiger D, Rotaru K, Churilov L. Supply chain risk identification with value-focused process engineering. J. of Operations Management, 2009, 27(2): 154–168.

[12] Tummala R, Schoenherr T. Assessing and managing risks using the supply chain risk management process (SCRMP). Supply Chain Management: An Int'l Journal, 2011, 16(6): 474–483.

[13] Ho W, Zheng T, Yildiz H, Talluri S. Supply chain risk management: A literature review. Int'l Journal of Production Research, 2015, 53(16): 5031–5069.

[14] Kern D, Moser R, Hartmann E, Moder M. Supply risk management: Model development and empirical analysis. Int'l Journal of Physical Distribution & Logistics Management, 2012, 42(1): 60–82.

[15] Cohen MA, Kunreuther H. Operations risk management: Overview of Paul Kleindorfer's contributions. Production and Operations Management, 2007, 16(5): 525–541.

[16] Gaudenzi B, Borghesi A. Managing risks in the supply chain using the AHP method. Int'l Journal of Logistics Management, 2006, 17(1): 114–136.

[17] Fan Y, Stevenson M. A review of supply chain risk management: Definition, theory, and research agenda. Int'l Journal of Physical Distribution & Logistics Management, 2018, 48(3): 205–230.

[18] Zsidisin GA. A grounded definition of supply risk. Journal of Purchasing and Supply Management, 2003, 9(5): 217–224.

[19] Ehrlinger L, Wöß W. Towards a definition of knowledge graphs. SEMANTiCS (Posters, Demos, SuCCESS), 2016, 48.

[20] Amit S. Introducing the knowledge graph: Things, not strings. Google Official Blog, Retrieved September 6, 2014.

[21] What is a Knowledge graph. https://www.ontotext.com/knowledgehub/fundamentals/what-is-a-knowledge-graph/

[22] Zhou P, Liang GY, Yu JG, Wang JM, Wu JJ, Zhao C. Customized operating system for RISC-V fragmentation based on software supply chain. China RISC-V Forum, 2019 (in Chinese).

[23] Fielding RT, Taylor RN. Architectural styles and the design of network-based software architectures [Ph.D. Thesis]. University of California, 2000.

[24] Thulasiraman K, Swamy MNS. Graphs: Theory and Algorithms. Wiley, 1992. 118.

[25] Kahn AB. Topological sorting of large networks. Communications of the ACM, 1962, 5(11): 558–562.

Guanyu Liang, Master. His research interests include cloud computing, operating systems and intelligent operation and maintenance.

Jingzheng Wu, Ph.D., professional member of CCF. His research interests include system security, vulnerability mining and operating systems.

Yanjun Wu, Ph.D., doctoral supervisor, senior member of CCF. His research interests include operating systems and system security.

Chen Zhao, Ph.D., doctoral supervisor, senior member of CCF. His research interests include compilation technology, operating systems and network software.