Constructing Supply Chains in Open Source Software

Yuxing Ma
The University of Tennessee, Knoxville
yma28@vols.utk.edu

ABSTRACT

The supply chain is an extremely successful way to cope with the risk posed by distributed decision making in product sourcing and distribution. While open source software has similarly distributed decision making and involves code and information flows similar to those in ordinary supply chains, the actual networks necessary to quantify and communicate risks in software supply chains have not been constructed on large scale. This work proposes to close this gap by measuring dependency, code reuse, and knowledge flow networks in open source software. We have done preliminary work by developing suitable tools and methods that rely on public version control data to measure and comparing these networks for R language and emberjs packages. We propose ways to calculate the three networks for the entirety of public software, evaluate their accuracy, and to provide public infrastructure to build risk assessment and mitigation tools for various individual and organizational participants in open sources software. We hope that this infrastructure will contribute to more predictable experience with OSS and lead to its even wider adoption.

CCS CONCEPTS

• **Software and its engineering → Risk management**; *Reusability*; Maintaining software;

KEYWORDS

software supply chain, open source, knowledge flow, risk management

ACM Reference Format:
Yuxing Ma. 2018. Constructing Supply Chains in Open Source Software. In *ICSE '18 Companion: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3183440.3183454

1 INTRODUCTION

Supply chain concept of a system of organizations, people, activities, information, and resources involved in moving a product or service from supplier to customer has been tremendously successful for helping businesses manage risks that arise from the distributed nature of production. OSS (Open source software) production is also distributed with decision making that is not centralized and done by a variety of individuals and groups. With a variety of risks affecting software development, it is reasonable to assume that the supply chain concept will benefit open source community and the achieved experience and findings in traditional supply chain for mitigating various risks will apply in OSS.

In regular supply chains the networks include material, financial and information flows. Analogous concepts need to be operationalized in the software domain. Based on the characteristics of software domain, especially the open source community, we propose to consider three types of network data to operationalize different kinds of flows: dependency network, code reuse network and knowledge flow network. In the first phase of research we focus on constructing supply chain based on these three networks.

For the sake of clarifying the benefit of supply chain for OSS, we define the term *visibility* in OSS supply chain as follows: *visibility refers to how far you can see upward beyond direct upstream, i.e. how many layers of dependency you can see from a software in supply chain.*

Constructing supply chain for OSS would increase developers' visibility over softwares which helps evaluate risks for softwares, e.g. a software with multiple layers of dependencies may have a larger risk compared to a software with less layers of dependency, because vulnerability discovered in one layer of dependency may expose a risk on the software at the bottom.

As a part of the first phase of research, we have already constructed dependency networks for R CRAN.

2 RELATED WORK

Though the study of supply chain is quite popular in traditional industries, the idea of software supply chain is pretty new.

In 1995, Jacqueline Holdsworth in his book [2] referred 'supply chain management' as the overall process for designing and engineering your success by specific value criteria set. He argued that the term 'software development life cycle' is too theoretical and lacks integration with business. It seems that his concept of software supply chain is a little narrow by restricting product range and no approach is provided on how to measure supply chain.

In 2003, Jack Greenfield [1] claimed that software supply chain emerged to feed the demand for components created by software factories. It can be inferred that in his paper the products in software supply chain are components that will be assembled in software factory to produce new products, which is partial because in software supply chain the product can be knowledge or expertise as well. He also argued from the prospective of product line that the emergence of software supply chain is to create standards to enhance the match and alignment between components and facilitate assembling process in down streams, but what we care is rather software supply chain's impacts on software quality.

To our knowledge no concrete supply chain have been built in OSS nor reasonable approaches have been promoted to built it. So we decided to build the first supply chain for OSS with 3 networks revealing the complicated relationships among the softwares and the developers.

3 RESEARCH PROBLEM

Based on the categories of product in supply chain, we come up with three networks to measure supply chains in OSS:

- Dependency network if the product is package or library
- Code reuse network if the product is a commit, a file or code snippet
- Knowledge flow network if the product is expertise

Hence our research question in the first phase is:

RQ How to construct these three networks in OSS domain?

4 CHALLENGES AND POTENTIAL SOLUTIONS

4.1 Common Challenges across All Networks

- **Data discoverability**
- **Data size**
- **Data quality:**

(1) Different platforms may have distinct data formats.
(2) Multiple VCSs are supported within single platform e.g. Github supports SVN as well.
(3) Publicly available data archive maybe partial or not up to date e.g. library.io data archive may lack full historical metadata.

4.2 Unique Challenges for Individual Networks

- Dependency network – Categorization of Dependencies
- Code reuse network – Detecting Code Reuse
- Knowledge flow network – Setting Flow Properties[1]

(1) How to measure the amount of knowledge transfered between developers i.e. how to weigh the flow?
(2) How to determine who is the mentor and who is the follower i.e. how to set the direction of the flow?

Proposed solution: These flow property problems can be resolved by applying formula[3]:

$$S_2(d_{j_0}, d_{j_1}) = \sum_{i:\begin{cases} n_{ij_0}, n_{ij_1} > 0 \\ FC(f_i, d_{j_0}) > FC(f_i, d_{j_1}) \end{cases}} \frac{n_{ij_0} + n_{ij_1}}{\sum_j n_{ij}} \qquad (1)$$

Annotations of formula (1): S is the strength of expertise transfered, d stands for developer, i represents file i, n_{ij_0} represents the number of changes developer d_{j_0} made to file i, $FC(f_i, d_{j_0})$ means the time of developer d_{j_0}'s first change to file f_i.

The formula can be interpreted as follows: the strength of expertise flow from developer d_{j_1} to developer d_{j_0} is based on the sum of their contribution ratio to files in which developer d_{j_1}'s first change is earlier than developer d_{j_0}'s. Files changed mostly by others where the two developers had contributed little would not contribute much to the measure, but files where at least one developer made significant fraction of changes would contribute a lot.

[1]Setting flow properties: Flow properties are the values associated with a flow including weight, direction, etc.

5 CURRENT STATUS

5.1 Constructing the Dependency Network

We started constructing the dependency network by exploring R CRAN ecosystem.

By creating a link from individual package to each dependency in its 'imports' and 'depends', we construct a dependency network for R CRAN.

5.2 Constructing the Knowledge Flow Network

We began constructing the knowledge flow network by investigating the expertise flow in a popular web front side framework – emberjs. Web front side framework has been attractive for a number of years and many open source developers already participated in to contribute their expertise which makes emberjs an ideal software to illustrate how complicated a knowledge flow network could be.

5.3 Constructing the Code Reuse Network

We started the construction of the code reuse network by mining code reuse pattern of emberjs. We collected all the blobs(git object preserving the content of a file) for emberjs[2], searched for projects that share blob with emberjs and investigated blobs that span multiple projects. These projects were gathered and categorized into different groups indicating different code reuse patterns.

6 CONCLUSION

In this paper, we presented the idea of constructing supply chains in OSS community. In particular, we proposed to create three types of networks: dependency network, code reuse network and knowledge flow network. Various challenges in creating these networks were described and multiple solutions were provided. The dependency network for R CRAN was constructed and illustrated. In the future, we will apply these three networks to help solve thorny problems in software risk management such as discovering vulnerable softwares and optimizing risk-based software dependency structure. We will share these three networks by creating a website and providing APIs for query services so that anyone interested in would be able to utilize these data. In our planed time line, the completion of constructing work will be finished by August 2018 and the creation of the website will be completed by the end of 2018.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Audris Mockus for his support. The work is part of the project "Open Source Supply Chains and Avoidance of Risk" sponsored by the National Science Foundation under Grant No.: 8083

REFERENCES

[1] Jack Greenfield and Keith Short. 2003. Software factories: assembling applications with patterns, models, frameworks and tools. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. ACM, 16–27.
[2] Jacqueline Holdsworth. 1995. *Software Process Design*. McGraw-Hill, Inc.
[3] Audris Mockus. 2009. Succession: Measuring transfer of code and developer productivity. In *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 67–77.

[2]https://github.com/emberjs/ember.js