

# Managing Trust in Collaborative Software Development

Haoyang Che<sup>1</sup> Dongdong Zhao<sup>2</sup>

<sup>1</sup>*Institute of Software, the Chinese Academy of Sciences, 100080, Beijing, China*

<sup>2</sup>*School of Software, Dalian University of Technology, 116621, Dalian, China*

<sup>1</sup>*chehy@hotmail.com*

## Abstract

*The advent of Collaborative Software Development (CSD) provides opportunities for software developers in geographically dispersed locations to communicate, and further build and share common knowledge repositories. However, such distributed development environment has posed too much higher amount of uncertainty and security problems when compared with traditional ones. To collaboratively work in a more secure and healthy CSD environment, trust must be built, negotiated, evolved, and managed among participating developers and external knowledge resources prior to the flowing of knowledge. This position paper introduces trust-related issues when it comes to modern CSD.*

## 1. Introduction

Recent years have seen a wide emergence of new paradigms of CSD, such as open-source and pair-programming software development. These styles of CSD are no longer confined to individual developers but should be reframed as inherently a knowledge intensive and distributed cognitive activity [1]. They can bring a multiplicity of benefits to participating developers, such as time to market, reusability, robustness, extensibility, testability, and/or adaptability. However, CSD that does not regulate the joining and departing of developers and “freely accessible” external knowledge sources can be subject to risks. In particular, malicious participants may resort to a variety of attacks, including sending spurious information, knowledge like fake code snippets and fake documents or manuals, acting as “the men in the middle”, and even posing as some others. The point is how can we trust other developers and knowledge sources in CSD. In addition to methodologies and user-centered CSD tools, extensive research must be conducted on how to construct a trustworthy CSD environment.

Why is trust important for CSD? In CSD settings, trust is required in order for developers to collaborate effectively. Without trust, developers will not normally share transient information and permanent knowledge, thus limiting their productive capacity. On the contrary, if

higher degrees of trust can be established, developers can work more efficiently, and adapt more quickly to ever-changing circumstances.

Existing CSD methodologies and tools suffer from the lack of supporting trust management during the coordinative and cooperative processes of knowledge construction among software developers, who have to resort to other tools and communication media for successful knowledge collaboration [1]. Trust-related issues in CSD fall into the social, organizational, and technical aspects, of which the former two dimensions involving cultural change and management commitment problems are beyond our focus.

## 2. Some Trust Issues in CSD

It is much easier to establish trust in a face-to-face environment than in CSD, in which the complexity of managing trust largely increases. Fundamentally, CSD focuses on whether or not knowledge senders and network resources can be trusted to deliver certain properties, such as privacy, integrity, and quality of service. To fulfill this, there are two main issues that need to be considered: trust model and knowledge sources.

Firstly, in order to implement efficient trust management and provide a uniform access method for every developer, a viable trust model must be devised and incorporated. The basic problem herein is that information and knowledge is dispersed throughout the CSD network so that every peer can only build an approximation of “the global situation in the network”. Therefore, peers may make assessments about directly-connected peers according to the history of attacks and unexpected behaviors and make recommendations to other indirect peers.

Our preliminary considerations have revealed the following technical challenges in building an efficient trust model:

- The basic functionality of the trust model is to judge to what degree a peer will cheat. Here the peer refers to the end developer/knowledge source and the intermediary. Therefore, we must determine the

factors and metrics for computing trust measure, and on top of that build a set of assessment methods.

- Software developers may not know the locations of their desired peers and knowledge. Thus, in the trust model trustworthy and knowledgeable peers should be easily identified with the aid of a sound QoS routing and recommendation algorithm.
- Trust data management is another challenge. Can the local algorithm be efficiently implemented in a given data management infrastructure? Could we store the data into Trusted Third Parties, or just store locally? Are multiple copies feasible in order to deal with single point of failure? Can we use buffer strategies for speedup in knowledge acquisition?

Secondly, we must identify potential knowledge sources including software source codes, reusable software components, and all kinds of documents, etc. When the trust model is applied, is it scalable to support more types of potential knowledge sources? How we can trust an unknown knowledge source? Consider the following scenario. A certified developer sends you a COM+ component to help your work. Although we can trust the developer in our trust model, should we fully trust the component which he may acquire from an unknown source outside our model? Perhaps eventually attacking the problem will require content-based analysis method seamlessly incorporated in the trust model. However, as an alternative, cryptology will help a lot to authenticate unknown knowledge sources at the very start.

### **3. Conclusions**

Trust in CSD is a relatively new line of research, and there are many open questions and interesting issues to be addressed. Since the growth in CSD will continue and the security threats will multiply, trust-related research seems imperative under the situation.

### **References**

[1] Ye, Y., Yasuhiro Yamamoto, and Kouichi Kishida, Dynamic Community: A New Conceptual Framework for Supporting Knowledge Collaboration in Software Development, Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04), Busan, Korea, 472-481, 2004.