TrustForge: Flexible Access Control for Collaborative Crowd-Sourced Environment

Jian Chang, Peter Gebhard, Andreas Haeberlen, Zack Ives,
Insup Lee and Oleg Sokolsky
{jianchan, pgeb, ahae, zives, lee, sokolsky}@cis.upenn.edu
Department of Computer and Information Science
University of Pennsylvania, Philadelphia, PA, 19104

Krishna K. Venkatasubramanian
kven@wpi.edu
Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA, 01609

***Abstract*—Observing the success of the open source software movement, the Adaptive Vehicle Make (AVM) is a program run by the Defense Advanced Project Agency (DARPA) with the goal of applying crowd-sourced and component-based engineering to the design of military vehicles. In this paper, we present a credentialing system called *TrustForge*, which enables effective and flexible access control for the AVM crowd-sourced repository.**

Credentialing systems are essential in crowdsourcing to ensure quality, since it is potentially open to contributions made by anyone. The open source software community has developed elaborate *manual* approaches of managing its contributor community, which are often very labor-intensive and inefficient. Our aim with TrustForge is to improve the automation of the credentialing and access control process in the context of component-based systems, where users contribute components at various levels of abstraction. TrustForge takes a hybrid approach that combines trust policy and reputation to address this problem. In TrustForge, a policy language is used to specify the access control rules for users in the system to contribute components. In addition, reputation values computed for users based on the quality of their past component contributions are used to tune the static policies to enable flexibility and adaptiveness.

The *contributions* of this work are as follows: (1) the design of TrustForge – an effective and flexible access control mechanism that combines policy and reputation approaches; (2) the identification of heuristics for component quality measurement and a novel reputation computation algorithm for evaluating user trustworthiness; (3) a data model based on provenance graphs that allows efficient repository information storage and retrieve. We have implemented TrustForge system and integrate it with the VehicleFORGE repository system to support the operation of the AVM challenge program. The evaluation results based on real-world deployment and systematic simulation demonstrate that TrustForge can effectively discern the trustworthiness of users within the crowd-sourced system.

I. Introduction

Flexible manufacturing and component-based design has the potential to revolutionize the way we build next-generation cyber-electro-mechanical systems. The use of components can improve outcomes and reduce costs of designing and building complex systems. Further, such *component-based* system designs can be *crowd-sourced*, where scores of participants collaborate to achieve a common goal. This is the vision of the Adaptive Vehicle Make (AVM), a program run by Defense Advanced Project Agency (DARPA) with the goal of applying crowd-sourced, components-based engineering to the design of military vehicles. The success of crowd-sourced software platforms such as Linux, Firefox, and Apache that adopt a similar methodology have paved the way for migrating the basic approach to military systems.

This work is supported in part by the Defense Advanced Research Projects Agency AVM program under Contract No. HR0011-11-C-0096 and ONR MURI N00014-07-1-0907. The views expressed are those of the TrustForge team and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

In this paper, we present *TrustForge*, an effective and flexible access control mechanism for crowd-sourced systems. TrustForge serves as part of VehicleFORGE, a large repository that stores the various component contributions and user activities for the AVM program. The repository provides TrustForge with metadata on users activity and components information, which is used to evaluate the trustworthiness of users for making access control decisions. Due to the sensitive nature of the content stored in the VehicleFORGE repository, the requirement of making effective access control decisions is more demanding than in other open-source platforms. This is because: (1) the identity of the participants may not be directly known to the project management; (2) the strictness of access control needs to be flexible depending upon the criticality of the project; and (3) the large scale of the enterprise regarding the number of participants and projects and their components.

Traditional policy-based trust management approaches rely on credentials that are assigned to participants by trusted authorities. Such approaches rely on cryptography and give precise guarantees that only participants with the right credential can gain access to the repository. Today's crowd-sourced software platforms follow such policy-based trust management, where the credentialing is done manually[1]. On the other end of the spectrum are reputation-based approaches that are based on *feedbacks* (*i.e.,* prior interaction experience) between the participants in the collaborative environment. In such approaches, the trust decision is made based on the history of interactions instead of being predetermined by the authority, which is more much dynamic and adaptive than policy-based approaches. With TrustForge, we are taking a hybrid approach for access control by incorporating constraints on user reputation in the access control policy. Reputation becomes one element in the array of user attributes. Examples of others attributes may be the citizenship status, the length of relevant experience, *etc.*.

[1]It is often slow and time-consuming, which in turn becomes a considerable barrier of entry.

In order to maintain reputation values, TrustForge periodically recomputes user reputations based on the history of user activities. In order to protect the reputation scores, TF is designed to compute reputation based on the objective information, rather than on subjective opinions of other users. Since we are primarily concerned with the quality of the components that a user contributed to the repository, the reputation of the user is primarily determined based on the reputation of his/her submitted components. The most objective assessment of component quality is through testing such as running simulation or other analysis method. However, testing must be performed by trusted users, who are by necessity a minority of users. In a large repository, one can only expect a small fraction of components to be tested with sufficient confidence. Therefore, we supplement test results with its utility information – if component C is used by a large number of components with high reputation, it means that contributors of those components found C acceptable in its quality, which gives us additional confidence in its evaluation.

The contributions of this work are as follows: (1) the design of TrustForge – an effective and flexible access control mechanism that combines policy and reputation approaches; (2) the identification of heuristics for component quality measurement and a novel reputation computation algorithm for evaluating user trustworthiness; (3) a data model based on provenance graphs that allows efficient repository information storage and retrieve. We have implemented TrustForge system and integrated it with VehicleFORGE repository system to support the operation of the AVM challenge program. The evaluation results based on real-world deployment and systematic simulation demonstrate that TrustForge can effectively discern users trustworthiness within the crowd-sourced system. Even though this work was done in the context of the AVM program, we will keep our discussion more general. The TrustForge architecture can potentially be used for crowdsourcing applications, as long as the underlying repository provides the necessary inputs it requires to operate.

The paper is organized as follows: Section II presents the related work. Section III presents the problem statement and our principal design goals for TustForge. Section IV presents the TrustForge system in detail including the policy engine, the reputation function design, and the metadata model. Section V then presents performance analysis results for evaluation. In Section VI, we conclude the paper.

II. Related Work

To the best of our knowledge, our work is one of the first that focuses on understanding access control automation in an open-source and collaborative code repository by adopting a quantitative methodology. A survey on the trustworthiness of open-source software was conducted in [1], in which the authors identify the reasons and motivations for companies to adopt or reject open-sourced software. The definition of trust in their context is more narrow than in our paper: [1] focuses only on software quality and trustworthiness, whereas we also focus on the trustworthiness of software developers. A few research efforts have identified effective practices adopted in the open-source software community for building high-quality and trustworthiness software as presented in [2], [3], and [4]. The key difference between TrustForge and the previous systems is that TrustForge enables a quantitative measurement of the trustworthiness of the contributors and their components in a collaborative environment.

Another domain of active research that is closely related to our work is on the quantitative measurement of software quality. This topic has attracted a lot of attention from the software engineering community since the very beginning of software and software development [5]. A lot of efforts has been spent on the precise definition of software quality, and on developing concrete quality metrics [6]. The current industry standard on software quality is ISO/IEC 9126 [7]. A comparison study has been conduct in [8], which compares various quality models including ISO 9126, ISO 15504 and describes a design of a more systematic model for assessing software efficiency and effectiveness. In [9], the authors propose a quantitative measurement technique of software architecture quality by combining the COSMIC full function points and ISO 9126 quality standards. A case-study is presented in [10], which use real-world data to illustrate the practice of ISO 9000 quality guideline in the software development process. In [11], the authors presented one of the earliest works that establishes a framework in the analysis of the characteristics of software quality. The authors also claim that paying close attention to characteristics of software quality can lead to significant savings in software life-cycle costs. Using techniques such as granular computing, neural networks, in [12] the authors propose an approach towards a quantitative software quality assessment with respect to extensibility, reusability, clarity and efficiency. In [13], the author present an overview of quantitative analysis techniques for software quality and their applicability during the software development life cycle. The paper also highlights how these techniques can be used for managing and controlling the quality of software. Applying existing quality metrics, a measurement study was conducted over 100+ open-source projects and the results are reported in [14]. Although none of these work directly address the problem of quantitative measurement of contributor trustworthiness, all these efforts can be adopted into TrustForge to build more fine-grained and application-specific metrics for evaluating the quality and trustworthiness of the contributions.

III. Problem Statement and Design Goals

The primary questions being addressed in this paper is to design an access control system for crowd-sourced, component-based repository system that is both *effective* and *flexible*. We would like to define clear and unambiguous trust policies to achieve effective management of complex trust relationships present in the crowd-sourced system, and grant only necessary privileges to end users according to system security requirements. Meanwhile, we would like to incorporate the dynamic and flexibility provided by reputation management system to improve the liveness of the access control mechanism by considering user contributions. Therefore, our answer to the question above is to combine the advantages of policy-based and reputation-based approaches. When designing such an access control system, we would like to achieve three main design goals:

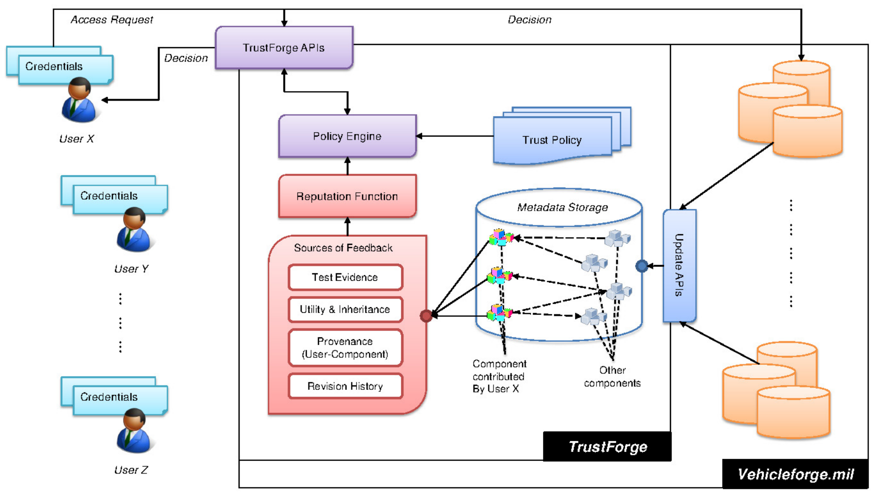

Fig. 1. TrustForge System Architecture

(1) Application-Specific Access Control Policy: The access control policy language should be expressive enough to capture the complicated trust semantics presented in the development scenarios of component-based systems. At the same time, we also want the policy language to be concise enough for fast policy reasoning during runtime.

(2) Discerning and Robust Reputation Algorithm: The reputation algorithm should be effective in discerning high-quality components from low-quality ones and effective in differentiating trustworthy users from malicious ones. Further, the reputation algorithm should be robust against attacks and strategies of gaming the system. Last but not the least, it should generate effective reputation values with only limited information and quickly converge to stable value ranges.

(3) Efficient Data Engine: To support the above two goals, it is necessary to organize and encode the information of the system, user, and component states in a very efficient way. An efficient data model and a high-performance storage engine is critical for the whole system to operate in real-time fashion.

IV. The TrustForge System

Given the design goals, in this section, we provide detailed discussion on the design of the TrustForge system. The system architecture of TrustForge is illustrated in Figure 1. In summary, when a user make an access request to any components stored in VehicleFORGE, he or she needs to present credentials. Please note that in VehicleFORGE, we have the notion of *project*, which is a set of relatively independent and self-contained components. Access control policies and the corresponding credentialing are configured by project administrators on the per-project basis. One user may participate in many different projects, and therefore has various credentials. Upon the receiving of user credentials and the access request, the policy engine of TrustForge check the credential against the corresponding project policies, in combination with user reputation information, to make the final access control decision. The reputation of a user is global across all the projects, which is computed based on various forms of feedback. All these metadata about users and components are stored and queried from a graph-based database. VehicleFORGE actively pushes metadata to populate this database.

A. Policy Specification and Evaluation

We use the KeyNote trust management language for specifying these access policies in TrustForge. KeyNote is a declarative language describing relationships among principals and evidence that permits principals to perform certain actions [15]. These relationships are specified as policies. If cryptographically signed, these policies can be viewed as a credential. KeyNote credentials and policies are known as *assertions*. When a trust inquiry is made, users present a set of credentials along with the desired action they wish to perform. The KeyNote compliance checker evaluates this input and returns a Compliance Value (CV) in a linearly ordered set, between an application specified minimum compliance value and maximum compliance value. In the simplest case, the set of compliance values can be {`DENY, ALLOW`}. The CV can then be used to make the access control decision.

B. Reputation Function

In order to enable the policies specified in KeyNote to be flexible, we use the notion of user reputation computed based on their contributions to the projects in the repository. Therefore reputation mechanism in TrustForge identifies high-quality components and high-quality contributors. Our design takes into account the presence of malicious users who may try to bias the reputation mechanism by either introducing erroneous feedback or taking advantages of the loop hole of the reputation algorithm design. In this section, we discuss our design of an effective reputation mechanism, which is also robust to potential attacks.

1) Reputation Representation: In TrustForge, the reputation of components and users is denoted as a three-tuple (t, c, f) analogous to the representation in CertainLogic [16]. CertainLogic provides a novel model for the evaluation of the trustworthiness of complex systems under uncertainty, which is also compliant with the standard probabilistic approach. The first two dimensions (t, c) represent the measured reputation

$$t_B = \begin{cases} \frac{\sum_{i=1}^{n} t_{A_i}}{n} & \text{if } c_{A_1} = c_{A_2} = \cdots = c_{A_n} = 1 \ , \\ 0.5 & \text{if } c_{A_1} = c_{A_2} = \cdots = c_{A_n} = 0 \ , \\ \frac{\sum_{i=1}^{n} (c_{A_i} t_{A_i} \prod_{j=1,\ j \neq i}^{n} (1 - c_{A_j}))}{\sum_{i=1}^{n} (c_{A_i} \prod_{j=1,\ j \neq i}^{n} (1 - c_{A_j}))} & \text{if } \{c_{A_i}, c_{A_j}\} \neq 1 \ . \end{cases}$$

$$c_B = \begin{cases} 1 & \text{if } c_{A_1} = c_{A_2} = \cdots = c_{A_n} = 1 \ , \\ \frac{\sum_{i=1}^{n} (c_{A_i} \prod_{j=1,\ j \neq i}^{n} (1 - c_{A_j}))}{\sum_{i=1}^{n} (\prod_{j=1,\ j \neq i}^{n} (1 - c_{A_j}))} & \text{if } \{c_{A_i}, c_{A_j}\} \neq 1 \ . \end{cases}$$

$$f_B = \frac{\sum_{i=1}^{n} f_{A_i}}{n}$$

Fig. 2. CertainLogic Fusion Operator

(t is the measured value, c is the confidence value), which encodes the reputation value computed based on direct feedback. Meanwhile, the third dimension f is called default reputation, which encodes the reputation value computed based on indirect evidence or inference. Two operators are defined for the reputation vector following the semantic of CertainLogic: (1) *Fusion operator* – it merges several reputation vectors into one. The detailed definition of the fusion operator is shown in Figure 2. And (2) *Expectation operator* – it computes a scalar reputation value based on the vector representation:

$$Expectation(A) = t_A * c_A + (1 - c_A) * f_A$$

2) Design of the Reputation Function: With the understanding of the reputation representation, we now describe the reputation algorithm used by TrustForge. We started by identifying all the feedback information source available, which can be used to evaluate the trustworthiness of user and the quality of component. For each feedback information source, we further identity a suitable algorithm, which can be used to infer the trustworthiness of components and users based on the corresponding feedback type. These algorithms serve as the basic building blocks. By properly combining them, we design a hierarchical algorithm that generates final reputation for both users and components as shown in Figure 3.

a) Feedback and Information Sources: In total, we identified four independent feedback sources for reputation computation, namely: (1) *Test Evidence:* the qualitative or quantitative test results on how well components satisfy their corresponding requirements. (2) *Component Utility:* the information on how the components are used as building blocks to compose more complex components. In our setting, the component hierarchy is well-defined (*e.g.,* determined by the physical structure of the vehicle design), which is specified by in the high-level project specifications. (3) *User-Component Provenance:* the information on components and their corresponding contributors, which is recorded as part of the repository log. (4) *Revision History:* the information on how component and user reputation evolves over times, which is also properly tracked by the code repository. All the feedback information is stored in the metadata model as discussed in Section IV-C.

b) Component Reputation Calculation: Figure 3 shows the skeleton and information flow of our reputation algorithm design, where nodes with the + and E symbol correspond to the fusion and expectation operator, respectively. To compute the measured reputation of component, we have designed two basic building blocks:

1) *Utility Graph:* This building block uses the graph of component utility relationships based on the assumption that if a component is highly used by other components in the system, then it is of high-quality. The t value is computed based on PageRank algorithm [17] over the graph of inheritance and utility links between components. The c value is assigned based on a TrustRank algorithm [18], which can prevent malicious users from unfairly increasing their reputation by adding meaningless links in the graph.
2) *Test-based:* This building block is based on test evidence. The idea is that by using qualitative or quantitative tests, one can measure how well a component satisfies its requirements. The t value is computed based on statistical distribution estimation of the test evidence to get the probability of requirement satisfaction by a component. The c value is computed based on the test of significance of the corresponding statistical estimator.

The value computed by these two building blocks are merged using the fusion operator to get the measured component reputation. For computing the default reputation of component, we have designed three building blocks:

1) *Utility Graph:* This building block uses the same component utility graph, but focuses on the out-going links instead of in-coming ones as in the computation of measured component reputation. The idea here is that if a component reuses design from other high-quality component, then this component is more likely to be of high quality, and *vice versa*. To compute this building block, we fetch the measured reputation of all the components from which one component uses.
2) *Revision History:* This building block uses the revision history information of components. The assumption is that if a component is high-quality in the past revisions, it will still be high-quality in the future, and *vice versa*. To compute this dimension, we fetch its past measured reputation values.
3) *Provenance (User-Component):* This building block uses the relationship of components and their corresponding contributors. The idea is that high-quality components are contributed by trustworthy users, and *vice versa*. To compute this dimension of reputation for a component, TrustForge fetches the measured reputation of all its contributors.

TrustForge merges the reputation value computed by these three building blocks using the fusion operator. However, we cannot directly use the resulting (fused) (t,c,f) tuple for the default reputation as in our model it is represented as a single value. Therefore we computed the expected value from the fused (t,c,f) tuple, as shown with the operator E in Figure 3.

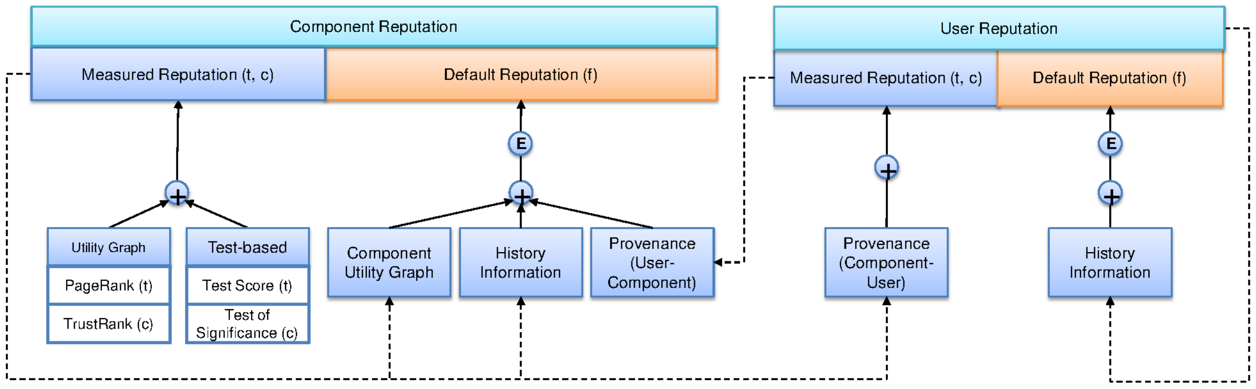

Fig. 3. TrustForge Reputation Algorithm Overview

c) User Reputation Calculation: The measured user reputation is computed using the provenance information between a user and all the components she or he has contributed to. The assumption here is that *high-quality* components are contributed by *trustworthy* users, and *vice versa*. To compute this dimension of reputation, we fetch the measured reputation of all components a user has contributed to, and merge these values using the fusion operator.

To compute the default reputation of user, we use the expectation of (t,c,f) tuple representing the revision history information. The assumption is that if a user contributed high-quality components in the past, he or she will keep doing so in the future, and *vice versa*. To compute this dimension of user reputation, we fetch historical reputation of a user (for new users, this step is skipped), merge these values using the fusion operator, and take the expectation of the aggregated value. And once the reputation values have been calculated, they are plugged into the KeyNote policy and the access control decisions are made for the individual users.

3) Attack Model and Defense Strategies: Thus far, we have described on the basic reputation algorithm design used by TrustForge. Here, we will look at its robustness. Malicious users can try to manipulate the reputation mechanism by introducing biased feedback information, and the algorithm needs to provide protection against these attempts. We first enumerate the attack models being considered in the TrustForge system. In summary, an attacker may want to unfairly boost his own reputation or decrease the reputation of benign users, in order to grant more access privileges. The attacker can only achieve the goal by injecting biased information into the TrustForge system to mislead the reputation algorithm. Concretely, we consider two types of attacks:

1) An attacker could introduce "dummy components" and "spamming links" into the utility and inheritance graph to unfairly increase the popularity of components contributed by the attacker. As a result, the PageRank value of the attacker's components will increase, and the PageRank value of others' components will decrease.
2) An attacker could submit false evidence to show that his components satisfy the corresponding requirements, or to show that others' components fail to satisfy the corresponding requirements. As the result, the "test" dimension of the component reputation will change in favor to attacker's components.

As the countermeasure to the first type of attack, we compute the TrustRank [18] as the confidence value as the countermeasure. As demonstrated in [18], the introducing of out-going spamming links won't increase the reputation of the attackers by using this defensive approach. To defend against the second type of attack, we specify trust policies to only allow a set of predefined trusted users to submit test results by given them curator privilege.

Furthermore, we assume that for the CertainLogic operators adopted in TrustForge, if the operant values are truthful and the operation is performed by truthful entity (in this case, TrustForge), then the result is also truthful. Section V provides analysis of TrustForge reputation function performance and demonstrates that it satisfies our design goals of being effective and robust.

C. TrustForge Metadata Model

In order to perform calculation of reputations, TrustForge needs to store and query certain metadata about the repository. This metadata is designed to capture three key aspects: the *provenance* of the objects in the repository, *i.e.,* who contributed them and who made modifications to them; the *usage* of the components, *e.g.,* as subcomponents of other components; and any *test results*.

TrustForge captures provenance information implicitly by observing checkins. Analogous to other version control systems, users can check in new components, modify or delete existing ones, and create or merge branches. Based on these checkins, TrustForge builds a history for each component in the repository that records all the changes that were made during the component's lifetime, as well as the corresponding time and the user who made them. Similarly, TrustForge also records the utility relationship between components.

Since both provenance and usage are inherently graph-structured, the decision was made to store the data as a graph, rather than in a relational database. The graph contains a vertex for each user, component, revision information, *etc.*; the edges describe relationships between vertices – for instance, that a particular revision was created by a specific checkin, or that a given component is part of another component. Test

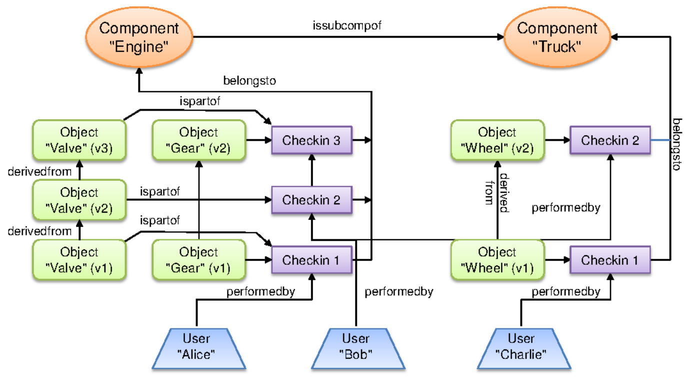

Fig. 4. The TrustForge data model. This simple repository contains two components, a truck and its engine, which were contributed by Alice, Bob, and Charlie.

results are not part of the graph; rather, they are represented as annotations to particular vertices.

1) Vertices and edges: TrustForge's data model contains three types of vertices – `user`, `component` and `checkin` – as well as five types of edges:

- A `derivedfrom` edge connects a component revision to other component revisions from which it was derived;
- A `ispartof` edge connects an component revision to the checkin that created it;
- A `belongsto` edge connects a checkin to the component to which it was applied;
- A `performedby` edge connects a checkin to the user that performed it; and
- An `issubcompof` edge connects a subcomponent to its parent component.

The provenance-related vertices and edges are created automatically after each checkin. When user U checks in modifications to a component C, TrustForge creates a new `checkin` vertex for the modified component and a `performedby` edge from C to U. The vertices are then annotated with some metadata, and the usage edges and the annotations for test results are created manually, in response to user actions.

Note that the graph is append-only – existing vertices and edges are never modified or removed. This is a security feature, since information is never lost, dishonest users have no way to remove telltale information. Of course, this means that TrustForge's storage requirements will grow over time, but they will not grow faster than those of the underlying repository, which never deletes past revisions either. If necessary, old revisions can be removed manually by the operators.

2) Query language: Although the primary function of the graph is to provide the input for the reputation function in Section IV-B, we expect that it will have other uses – *e.g.,* to perform forensics when bad contributions are discovered, or to search for potential contributors for a new project based on the users' prior expertise. Furthermore, TrustForge's reputation function may evolve over time, *e.g.,* to support new types of tests or new kinds of contributions. Therefore, instead a hard-coded interface to the graph, TrustForge contains a general-purpose query language that can be used to formulate a wide variety of queries.

TrustForge's query language is an extension of ProQL [19], [20], a query language that was specifically designed for provenance graphs. ProQL queries use a *path expression* syntax: each query describes a set of paths in the TrustForge graph, and then performs some computation on those paths. Fruther, the original ProQL language did not support iterative computations similar to PageRank, which is an important building block in TrustForge's reputation function. Therefore, we extended ProQL with a `repeat...until` construct (which can express the iteration and its termination condition) as well as with support for propagation and adjustment of vertex annotations (which can carry the page ranks). With these extensions, we can express the entire PageRank computation in a short ProQL query, thus meeting our design goal of having a efficient data engine. A more detailed description of these extensions is available in [20].

V. TrustForge Evaluation

Now that we have described the TrustForge system design, in this section, we will evaluate its capabilities in terms of differentiating the trustworthy users and high-quality components from the bad ones. The TrustForge system has been implemented and integrated with the VehicleFORGE system, which has been operating for over 2 months to support the AVM FANG challenges [21]. In this section, we first report our experience of the real-world deployment. For more systematic evaluation, we further conducted a simulation-based evaluation and present the performance analysis results obtained through.

A. Real-world Deployment Analysis

The TrustForge system, being part of the VehicleFORGE system, has been supporting the operation of the AVM FANG challenges for over two months at the time of writing. The AVM FANG challenge requires participants to develop a subsystem of a military-relevant vehicle, focusing on its mobility and drivetrain. We report analysis of the preliminary data obtained . At the time of writing, the challenge has attract 1000+ registered users, with around 200 users actively contributing models and designs. So far 500+ subsystem designs have been submitted for test and evaluation, which utilized over 6000 components contributed within the repository. The reputation computation is triggered on a weekly basis. The most recent snapshot of user reputation (including only users who actively contribute to the repository) is shown in Figure 5. We can see that the reputation of users is evenly distributed from the lowest to the highest, which demonstrates the high differentiation capability of our reputation algorithm. Similar curve pattern holds for component reputation, which spreads from 0.0 to 0.99 with 0.5 as both mean and median value.

Moreover, we have observed two interesting metrics from more detailed data analysis. First, more than 90% of users have less than 10% of changes in their reputation values between two consecutive reputation updates. Further analysis shows that more significant reputation changes (*i.e.,* greater than 10%) are mainly due to newly registered users whose initial contributions change their reputation from the default value. Second, we compute the reputation variance of different

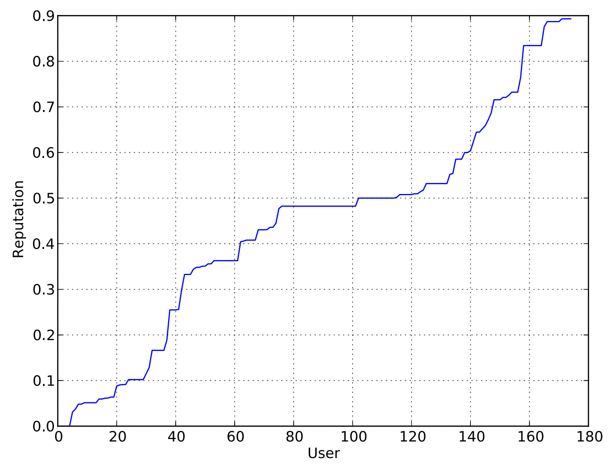

Fig. 5. User Reputation Snapshot

component contributed by the same user. The average variance is 0.02, which is a very small value. Both metrics indicate that the behavior pattern of user is stable over time, which provides strong justification to the usage of reputation as an effective trust quantification metric.

B. Simulation-based Evaluation

In order to systematically test the efficacy of the TrustForge policy engine and the reputation computation and fine tune its performance we built a VehicleFORGE simulator. The purpose of the simulator is to mimic the basic capabilities of the VehicleFORGE repository in terms of providing the TrustForge with information updates on the users and their component contribution, specifying policies and performing access control. This approach allows us to evaluate and tweak our system by covering the configuration space. Consequently, the principal capabilities of the simulator are: (1) the ability to provide updates regarding components and their interconnections; (2) the ability to add test results to components; and (3) the ability to simulate various user types with varying degrees of maliciousness.

1) Simulator Design: The simulator operates on a *trace*, which specifies properties of each user involved in the simulation, and records interactions between users and the repository: introduction and revision of components, and introduction of test resuts. A generated trace can be stored and used to evaluate different policies and reputation algorithm with respect to the same scenario. A trace is generated for a particular *component type hierarchy*, which is an acyclic graph specifying, for a given component, the necessary types of its subcomponents. A type hierarchy can be used to generate multiple traces with different kinds of users, as described below. The simulator processes entries in the trace one by one. Whenever a component of type ct is to be added by a user U, the simulator selects subcomponents of the right types from the repository, creates a *ground truth* quality value between 0 and 1 for the component based on parameters of U. The simulator then enters an instance of ct into the repository, linking it to the selected subcomponents. Trace generation algorithm ensures that subcomponents of ct will be present in the repository

User Type	Adding High-Quality Components	Choosing Basic Component
Good	100%	Best
Purely Malicious	0%	Worst or Own
Malicious Provider	0%	Random
Disguised Malicious	50%-100%	Random

Fig. 6. User Behavior Models in TrustForge Simulator

whenever an instance of ct is to be created. Whenever a test is to be added for a component C, the simulator randomly generates a test outcome based on the ground-truth quality of the component.

2) User Behavior Model: The simulator supports various types of user behavior patterns, shown in Figure 6. Each user has two properties: one specifies the expected quality of components added by the user; the other one describes how the user chooses subcomponents for the components it creates. A good (*i.e.,* trustworthy) user contributes high-quality components all the time, while choosing subcomponents with high reputation. A purely malicious user contributes bad components and chooses subcomponents with poor reputation to boost their perceived utility. Similarly, a malicious provider never contributes high-quality components and randomly chooses subcomponents for its composite components. Finally, a disguised malicious user only adds high-quality components between 50%-100% of the time to disguise its maliciousness. We use the normal distribution to determine the probability of adding high-quality components in a particular transaction for the disguised malicious users.

C. Simulation-based Evaluation Results

To exercise the simulator and the TrustForge infrastructure discussed in previous sections, we conduct experiments to evaluate the effectiveness of TrustForge by extensively covering the configuration space. To setup the experiments, we use the simulator to generate a component type hierarchy with 50 component-type nodes and 100 "component type to component type" links. For each experiment, we ran it over at least 1000+ revision iterations. And for every 10 revision updates, we re-compute component and user reputation based on the reputation algorithm discussed in Section IV-B. This configuration is fixed across the series of experiments we conducted.

On the user side, we set up a community of 20 users. The reason for choosing a relatively small number of users in the experiment is that we would like to accelerate the reputation evolving process. Since we set the reputation update interval to be 10 revisions, we want to make sure that the number of revisions per user is relatively large. Further, we vary the percentage of different user types among user community (detailed user behavior models discussed in Section V-B2). For disguised malicious users, we set their probability of contributing good components to be 50% in most of the experiments, unless otherwise noted. At most 50% of good users are given curator privilege to submit test results, and we also vary the size of this trusted tester set in some of the experiments. Tests are generated by such trusted testers for

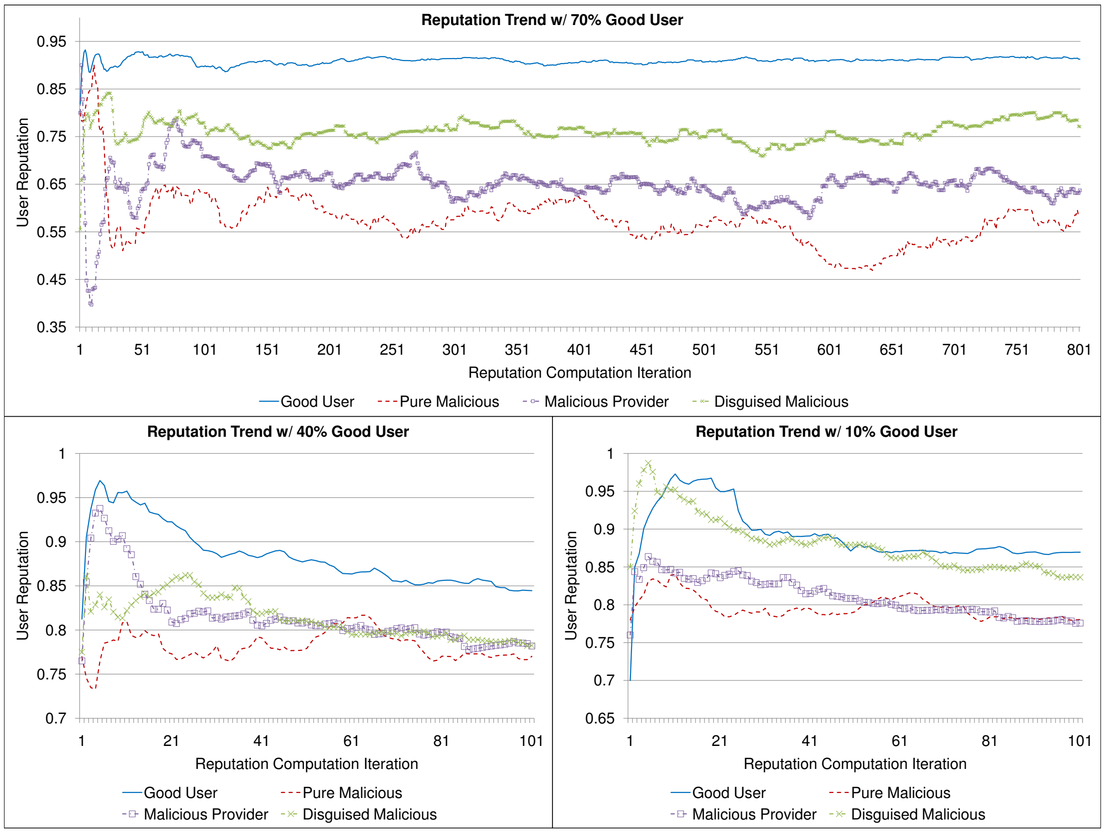

Fig. 7. Overall Reputation Trends

components that are randomly picked from the repository. We take this approach as its absence presents an easy attack vector for malicious users, who can introduce poor components within the system by giving them very high test scores. Test scores are therefore allowed from users who can be trusted to be accurate in their assessment of components. The measured trust value of test results is informed by the ground truth quality of the corresponding components with $\pm 10\%$ error. The confidence value of test results is uniformly set to be 95%.

1) General Reputation Trends: We first demonstrate the overall trend of user reputation computed by TrustForge. We vary the percentage of good users in the user community from 70%, 40% to 10%. Meanwhile, the corresponding percentage of total number of malicious users was increased 10%, 20%, and 30%. We run simulation experiments for each configurations over 8000 revision iterations (*i.e.,* 800 reputation computations). In Figure 7, we plot the average reputation trends for different user types over time. For clearer presentation purpose, we plot the full trend for the 70% good user configuration. For the other two configurations, we only plot the first 1000 revisions (*i.e.,* 100 reputation computations), which is good enough to capture the characteristics that we are interested in. Using the experiments, we observe that:

- *Convergence and Sensitivity:* The reputation curves quickly converge to a stable range for both good and malicious users after 20-30 reputation computation iterations. This shows that the sensitivity of the reputation function is good enough to capture user's behavior.
- *Effectiveness:* Further, we can observe a clear separation between different user behavior models. That is: (a) Good users often have much higher reputation than malicious ones; even when the good users are an absolute minority among the users. (b) Among the three malicious user types, the purely malicious users often get the worst reputation; both malicious providers and disguised malicious users get better reputation as they demonstrate more trustworthy behaviors than purely malicious ones. These results demonstrate that the reputation algorithm adopted in TrustForge is effective in discerning good and various types of malicious users.

2) Average Separation: With the overview of the reputation trends in mind, we further conducted experiments to investigate the reputation differences between good users and each of the three types of malicious users. In this regard, we created three user communities with good users and one type of malicious users – (good, purely malicious), (good, malicious provider), and (good, disguised malicious). For each user community, we vary the percentage of good users from 10%, 25%, 40%, 55%, 70%, 85%, 95%, and record the average reputation of good users and the corresponding type of malicious users from the 200th to 1000th revision iteration (*i.e.,* when the reputation value of user has converged to a rather stable range).

As shown in Figure 8, the reputation margin between good users and malicious users are wide enough for most

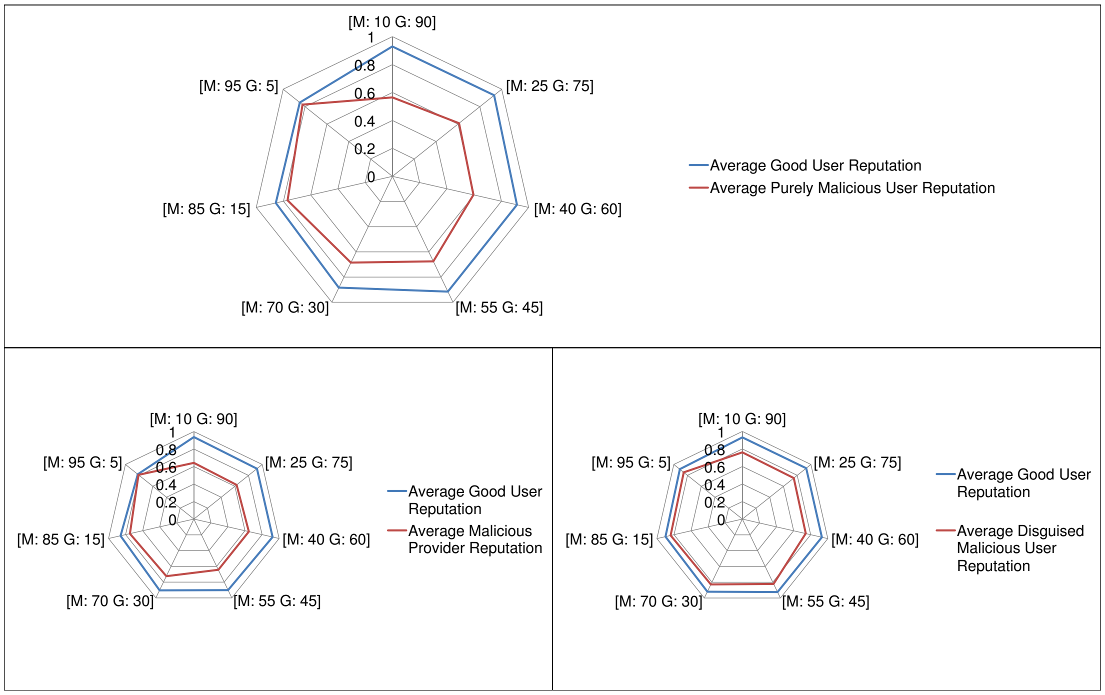

Fig. 8. Average Reputation Margin Between Good Users and Different types of Malicious Users. (Top) Margin between Good Users and Purely Malicious Users. (Bottom-left) Margin between Good Users and Malicious Provider Users. (Bottom-right) Margin between Good Users and Disguised Malicious Users.

of the scenario. Although the margin become narrow as the percentage of good user decreases within the community, we can still maintain meaningful margins even when the good users only 30% of the total number of users. Further, although the reputation of malicious users can be very close to that of the good users in some scenarios (especially when malicious users is more than 85%), the average reputation of good user is still higher than malicious ones. These results suggest that the reputation value computed by TrustForge has good average performance in discerning different type of users.

3) Minimal Separation: After a description of the average-case performance of the reputation mechanism, we switch our focus to the worst-case scenario. That is, we want to understand to what extent malicious users can game the system. As we see in previous experiments, the reputation margin between disguised malicious users and good users is the closest, since half of the time such malicious users demonstrate good behaviors. Therefore, we conduct further experiments by increasing the probability for disguised malicious users to exhibit good behavior from 50%, to 60%, 70% or even 90% (the percentage of good users in the community is set to be 70% in these experiments). Furthermore, we measure the margin between the disguised user who has the *highest* reputation value and the good user who has the *lowest* reputation value.

We illustrate the trends of this minimal reputation margin between good users and disguised malicious users in Figure 9. As we can see, as the disguised users exhibit more and more good behavior, essentially the observed behavior pattern of good users and disguised malicious users becomes very similar to each other. The reputation margin become smaller and it takes a longer time for the reputation algorithm

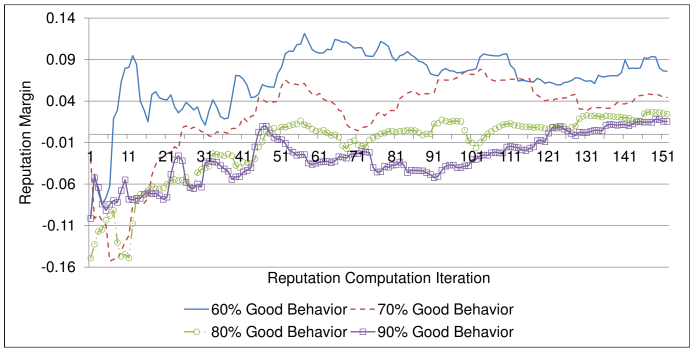

Fig. 9. Minimal Reputation Margin Between Good Users and Disguised Malicious Users with Various Probability of Exhibiting Good Behavior

to discern the difference. Overall, the reputation margin is positive for most cases, even when the disguised malicious users mainly exhibits good behavior. Currently, we are still actively working on improving this margin using improved trust calculation functions such as those presented in [22]. This along with investigating advance persistent threats where adversaries behave correctly for long durations of time and then suddenly switch to malicious behavior are important future research directions. In summary, the robustness of the TrustForge reputation mechanism is largely satisfactory in meeting our design goal under the attack models discussed in Section IV-B3.

4) Reputation under Limited Tests: We further conducted experiments to investigate the impact of the amount of available tests on the effectiveness of our reputation mechanism. In this experiments, we set up a user community with 70%

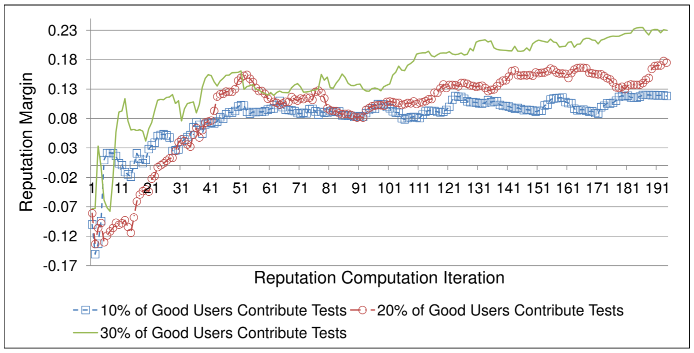

Fig. 10. Minimal Reputation Margin Between Good Users and Purely Malicious Users with Various Amount of Tests

of good users and 30% of purely malicious users. We then vary the size of the tester set, who are good users granted with curate privilege to submit test results, from 10%, 20%, to 30% of the total good users in the community (i.e., in contrast to 50% in our previous experiments). Furthermore, we measure the reputation margin between the purely malicious user who has the *highest* reputation value and the good user who has the *lowest* reputation value. By giving only limited test results to components, our reputation algorithm gets much less raw data that is used as the basis for the aggregate and computation of user reputation.

We illustrate the trends of this minimal reputation margin with limited test result in Figure 10. As we can see, after a few (about 5 - 20) initial iterations, the minimal separation between good user and malicious users is maintained. With less test results, the margin decreases, but is still wide enough for making correct access control decision. The results shown here suggests that our reputation mechanism can also work effectively with limited amount of test information. This result is also encouraging when one considers the application of TrustForge design to other open-source repository environments.

VI. Conclusions and Future Work

In this paper, we presented an access control system called *TrustForge* that enables effective and flexible credentialing for crowd-sourced component-based systems. In this regard, TrustForge automates the credentialing and access control process. It take a hybrid policy and reputation-based approach to address this problem. The policy language is used to specify the credentials for users in the system to contribute components. The reputation scores are then used to tune the credentials. Our implementation of TrustForge and its evaluation revealed its capabilities in terms of discerning users with various levels of trust. The next step in this regard is conduct long-term evaluation for our real-world deployment and fine-tune the reputation, data storage and policy engine further.

References

[1] V. del Bianco, L. Lavazza, S. Morasca, and D. Taibi, “A survey on open source software trustworthiness,” *IEEE Softw.*, vol. 28, no. 5, pp. 67–75, Sep. 2011. [Online]. Available: http://dx.doi.org/10.1109/MS.2011.93

[2] M. J. Gallivan, “Striking a balance between trust and control in a virtual organization: a content analysis of open source software case studies,” *Information Systems Journal*, vol. 11, no. 4, pp. 277–304, 2001. [Online]. Available: http://dx.doi.org/10.1046/j.1365-2575.2001.00108.x

[3] A. Mockus, R. T. Fielding, and J. D. Herbsleb, “Two case studies of open source software development: Apache and mozilla,” *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 3, pp. 309–346, Jul. 2002. [Online]. Available: http://doi.acm.org/10.1145/567793.567795

[4] L. Zhao and S. Elbaum, “A survey on quality related activities in open source,” *SIGSOFT Softw. Eng. Notes*, vol. 25, no. 3, pp. 54–57, May 2000. [Online]. Available: http://doi.acm.org/10.1145/505863.505878

[5] R. J. Rubey and R. D. Hartwick, “Quantitative measurement of program quality,” in *Proceedings of the 1968 23rd ACM national conference*, ser. ACM ’68. New York, NY, USA: ACM, 1968, pp. 671–677. [Online]. Available: http://doi.acm.org/10.1145/800186.810631

[6] K. Akingbehin, “A quantitative supplement to the definition of software quality,” in *Software Engineering Research, Management and Applications, 2005. Third ACIS International Conference on*, aug. 2005, pp. 348 – 352.

[7] “ISO/IEC 9126 software engineering - product quality.” [Online]. Available: http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=35733

[8] M. Ortega, M. Prez, and T. Rojas, “Construction of a systemic quality model for evaluating a software product,” *Software Quality Journal*, vol. 11, pp. 219–242, 2003, 10.1023/A:1025166710988. [Online]. Available: http://dx.doi.org/10.1023/A:1025166710988

[9] G. Zayaraz and P. Thambidurai, “Quantitative measurement of software architectural qualities through cosmic ffp,” in *India Conference, 2006 Annual IEEE*, sept. 2006, pp. 1 –4.

[10] A. Tiwari and A. Tandon, “Shaping software quality-the quantitative way,” in *Software Testing, Reliability and Quality Assurance, 1994. Conference Proceedings., First International Conference on*, dec 1994, pp. 84 –94.

[11] B. W. Boehm, J. R. Brown, and M. Lipow, “Quantitative evaluation of software quality,” in *Proceedings of the 2nd international conference on Software engineering*, ser. ICSE ’76. Los Alamitos, CA, USA: IEEE Computer Society Press, 1976, pp. 592–605. [Online]. Available: http://dl.acm.org/citation.cfm?id=800253.807736

[12] M. Reformat, W. Pedrycz, and N. Pizzi, “Software quality analysis with the use of computational intelligence,” in *Fuzzy Systems, 2002. FUZZ-IEEE’02. Proceedings of the 2002 IEEE International Conference on*, vol. 2, 2002, pp. 1156 –1161.

[13] M. A. Talib, A. Khelifi, A. Abran, and O. Ormandjieva, “Techniques for quantitative analysis of software quality throughout the sdlc: The swebok guide coverage,” in *Software Engineering Research, Management and Applications (SERA), 2010 Eighth ACIS International Conference on*, may 2010, pp. 321 –328.

[14] H. Barkmann, R. Lincke, and W. Lowe, “Quantitative evaluation of software quality metrics in open-source projects,” in *Advanced Information Networking and Applications Workshops, 2009. WAINA ’09. International Conference on*, may 2009, pp. 1067 –1072.

[15] M. Blaze, J. Feigenbaum, and J. Lacy, “Decentralized trust management,” in *Proceedings of the 1996 IEEE Conference of Security and Privacy*, Oakland, CA, 1996.

[16] S. Ries, S. M. Habib, M. Mühlhäuser, and V. Varadharajan, “Certainlogic: A logic for modeling trust and uncertainty (short paper),” in *In Proceedings of the 4th International Conference on Trust and Trustworthy Computing (TRUST 2011)*. Springer, Jun 2011.

[17] L. Page, S. Brin, R. Motwani, and T. Winograd, “The pagerank citation ranking: Bringing order to the web,” 1999.

[18] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen, “Combating web spam with trustrank,” in *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, ser. VLDB ’04. VLDB Endowment, 2004, pp. 576–587. [Online]. Available: http://dl.acm.org/citation.cfm?id=1316689.1316740

[19] G. Karvounarakis and Z. G. Ives, “Querying data provenance,” in *Proc. SIGMOD*, 2010.

[20] Z. G. Ives, A. Haeberlen, T. Feng, and W. Gatterbauer, “Querying provenance for ranking and recommending,” in *Proc. TaPP*, 2012.

[21] “Ready, set, design DARPA’s first FANG challenge begins today.” [Online]. Available: http://www.darpa.mil/NewsEvents/Releases/2013/01/14a.aspx

[22] M. Nojoumian and T. C. Lethbridge, “A new approach for the trust calculation in social networks,” in *3rd Int. Conf. on E-Business (ICE-B)*, 2006, pp. 257–264.