TRUSTIE: A Software Development Platform for Crowdsourcing

Huaimin Wang, Gang Yin, Xiang Li and Xiao Li

Abstract Software development is either creation activities that rely on developers creativity and talents, or manufacturing activities that follow the engineering processes. Engineering processes need to include creation activities to address tasks such as requirement elicitation and bug finding. On the other hand, by exploiting the crowd wisdom, open-source development has been demonstrated to be a suitable environment for software creation. However, it also has several limitations, such as guaranteeing the progress and quality of production process. This paper introduces a software development platform and ecosystem that combines the strengths of the two models. First, we propose the Trustworthy Software Model as a basis to support such a hybrid development ecosystem. The core of this model contains a novel lifecycle model, an evidence model and an evolution model. Second, based on the model, we propose the Trustworthy Software Development and Evolution Service Model. It integrates *crowd collaboration*, *resource sharing*, *runtime monitoring*, and *trustworthiness analysis* into an unified framework. Based on this integrated model, we designed and implemented TRUSTIE, which distinguishes itself from other software crowdsourcing platforms by providing the software collaborative development service and the resource sharing service with the general support of trustworthiness-analysis tools. TRUSTIE enables crowd-oriented collaboration among internal development teams and the external crowds by combining the software creation and software manufacturing in one ecosystem.

H. Wang · G. Yin (✉) · X. Li · X. Li
National Laboratory for Parallel and Distributed Processing, School of Computer, National University of Defense Technology, Changsha 410073, China
e-mail: jack_nudt@163.com

H. Wang
e-mail: whm_w@163.com

W. Li et al. (eds.), *Crowdsourcing*, Progress in IS,
DOI 10.1007/978-3-662-47011-4_10

1 Introduction

Software development is an intellectual activity [1]. During the early phases of the software development, most of works are creative activities where people work together to analyze requirements and design software. Once the initial specification or design is available, automated algorithms are available to perform analysis for producing quality code, such as completeness and consistency checkers, automated code generators, and test case generators. The later processes, often rigorously, may be considered as a software manufacturing process. In spite of significant progress in software technology, many steps of software development processes are still manual. For example, requirement elicitation and bug removal [2] are mostly creative tasks in which automation plays a very limited role. Encouraging and facilitating creative activities are very important.

1.1 Lessons from Open-Source Software Development

Recently, Open-Source Software (OSS) has significantly changed our understanding of software development. Since the 1980s, OSS has continued to grow in both quality and quantity, and has become a source of software for numerous organizations. OSS development is different from traditional software development in several aspects: teams are decentralized; resources are rapidly shared; new versions are frequently released; and online communities of developers have always been formed. These characteristics enable people to create software in a distributed and collaborative manner. For example, OSS websites like *Github*, *Google Code* and *Sourceforge* make it possible for anyone to create and manage OSS projects at any time. Besides, OSS projects are open to all the developers. For example, users from all over the world, regardless of their prior training or experience, can engage in design discussion, contribute their code, and engage in testing through bug reporting. Thus, software development is greatly facilitated through this openness and massive crowd participation. Software development will profit greatly from an effective ecosystem empowered by *crowd wisdom*.

1.2 Crowd Wisdom

In this paper, "crowd" means "an undefined large group of people" [3]. For example, in Wikipedia, there are more than 19 million registered user accounts,[1] who have edited more than 30 million pages.[2] Their accuracy was found to be similar to the Encyclopedia Britannica [4]. Linus Torvalds, creator of the Linux Open Source

[1] http://en.wikipedia.org/wiki/Wikipedia:Wikipedians.

[2] https://en.wikipedia.org/wiki/Special:Statistics.

Operating System, said that "the most exciting developments for Linux will happen in user space, not kernel space" [5], in which "the user space" is the environment where a large number of people contribute their code. The Mozilla OSS project, which produces the famous Firefox browser, has gathered a crowd of over 1,800 people as acknowledged contributors.[3]

Software creation activities are now becoming an active arena for crowd wisdom. The success of this transformation is evidenced by the above-mentioned and other successful OSS projects. The insight nature of this success can be explained by the "wisdom-of-crowds" effect in cognition, coordination or cooperation problems [6]. The aggregated performance of a crowd will often outperform any single elite or small team of elites. Software creation tasks, such as eliciting requirement, negotiating the design of modules, finding and fixing bugs, are indeed cognition, coordination or cooperation problems. In traditional software engineering, these innovative tasks are assigned to dedicated teams, and are performed under a central control. However, as reported in [7], innovation and knowledge are essentially distributed and can hardly be aggregated by using centralized models. In OSS projects, software creation is outsourced to an open crowd, where massive, diversified, and non-professional contributions converge to the diffusion of innovation, resulting in the rise of wisdom-of-crowds revolution in software development.

1.3 Ecosystem Incorporates Engineering and Crowd Wisdom

By exploiting crowd wisdom, OSS development can alleviate the problems encountered in software creation which are hard to tackle by engineering methods. However, crowd wisdom method is not intended for all scenarios. The majority of commercial or industrial software systems are still developed through traditional engineering methods, though with the addition of agile elements recently. This is due to the reason that the engineering methods and tools have central control over requirements management, progress scheduling and quality assurance. Crowd wisdom is essentially elusive and unpredictable. Without central control, OSS development can hardly guarantee anything ahead of time, which is intolerable for most commercial products.

For the above reason, we do not advocate that the crowd wisdom method should replace the engineering method. Instead, we propose that these two paradigms should be combined, so that traditional software production can benefit from crowd wisdom. The end of this reasoning coincides with the business strategies of big companies, such as IBM, who embraces open source to benefit its software business [8]. However, we take a different perspective as development platform designers in understanding this end.

Our approach is to establish a software ecosystem that incorporates engineering methods and crowd wisdom. In the ecosystem, software creation activities are

[3] http://www.mozilla.org/credits/.

well-supported by exploiting crowd wisdom; meanwhile software manufacture is well-supported through implements of engineering methods. While crowd wisdom methods stress more in respecting the creativity of each individual [9], an important goal of engineering is the quality or trustworthiness of the software system. A key challenge in bridging these two types of development methods is to ensure the quality or trustworthiness of a software system up to an industrial standard and meanwhile respecting the creativity of each member of the crowd.

In this paper, based on how software systems are actually evolved in crowd-based development practices, we adopt a *crowd-based approach*, by proposing a Trustworthy Software Model (TSM) for quality assurance of the new ecosystem with a platform. We propose a new Software Development and Evolution Service Model (SDESM) that offers *crowd collaboration, resource sharing*, *runtime monitoring* and *trustworthiness analysis* as four basic services. Based on the TSM and SDESM, we implemented TRUSTIE (**Trust**worthy software tools and **I**ntegration **E**nvironment), a software development platform.

This paper is organized as follows: Sect. 2 describes the TSM; Sect. 3 proposes the SDESM. Section 4 introduces TRUSTIE including its architecture and application practices; Sect. 5 covers related work; The last section concludes this paper.

2 Trustworthy Software Model (TSM)

In the crowd-based software development paradigm, the meaning of software trustworthiness is fundamentally different from that of the traditional software. It is the foundation for designing a software development ecosystem built upon crowd wisdom.

2.1 Life Cycle Model

In the traditional lifecycle model, software consists of program and documentation, and it has two phases: development phase and application phase [10], as shown in Fig. 1a. After completing the development, a software system enters the application phase through a distribution or releasing process. Any user feedbacks are returned to the development team for software update for the next release.

In crowd-based development, huge amounts of software data can be generated at different phases of software life cycle. These data are accumulated, and shared in various developer communities, which not only reflect software functionality, but also can be used to analyze and measure various software properties. Take OSS as an example. The software data are spread mainly in collaborative development communities (such as *SourceForge* and *GitHub*) and knowledge sharing communities (such as *StackOverflow*). The former consists of software repositories that can be used to analyze the quality of codes and software development processes, while the latter

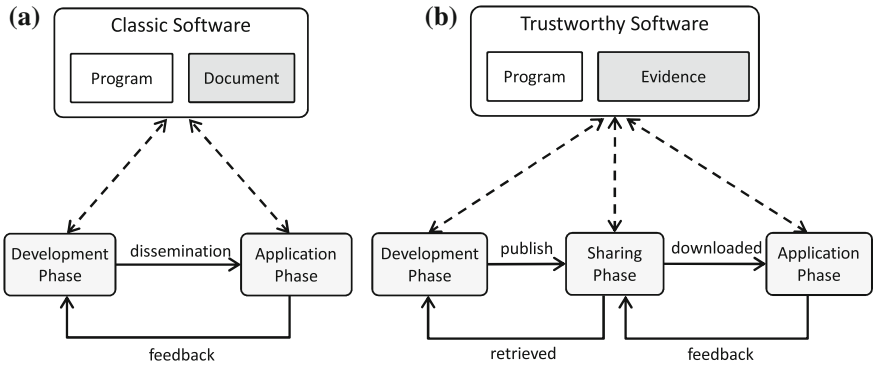

Fig. 1 Lifecycle models of **a** classic and **b** Trustworthy Software

can be used as the textual feedbacks from the crowds (such as comments, Q&As). The corresponding life cycle model is shown in Fig. 1b.

In this new model, the contents of *software* are code and evidences, and specifically, evidences as specific data that provide useful facts about various software properties. A software life cycle is now extended into three interwoven phases: development, sharing and application. The sharing phase is important for the collection and exploitation of crowd wisdom. When early versions of software (e.g., alpha versions) enter the sharing phase through releases, they can be downloaded, tested, analyzed, and assessed by the public. All user-generated data are then fed back to the development team, so the developers can identify defects and possible extension points.

Based on this model, a software entity has different forms in the three phases: software project, software resource and software instance. A *software project* contains a set of software artifacts and development data generated by developers with a roadmap. A *software resource* contains a set of software programs and evidences published by its provider. A *software instance* contains a set of running programs, status and application data of an online software system. The separation of the three forms of software entities will make it clear how to support software development in different software lifecycles.

2.2 Trustworthy Evidence Model

Software trustworthiness evidences are structured or unstructured data that can directly or indirectly reflect trustworthiness attributes. Here, attributes include not only *objective* quality attributes like correctness, reliability, performance, safety, security [11], but also *subjective* attributes such as user evaluations. In some cases, meta-attributes calculated by comparing different quality attributes can also be used as evidence. In different phases of a software life cycle, different trustworthiness

evidences can be generated. The trustworthiness evidence model mainly contains three types of software evidences:

Development Evidences are evidences produced in the development phase, including measurements and descriptions that reflect various attributes of a software artifact, development process and teams. Examples are source code quality, bug fix time, and measurements on the other development activities.

Sharing Evidences are evidence produced in the sharing phase, including community-based measurements like number of downloads and followers, measurements of project activities, and ranking results. Community evidences are data generated from online sharing platforms. Thus, they are indirect attributes of software systems. These reflect the social attributes of a software system.

Application Evidences are evidences produced in application activities, including measurements and assessments given by users that reflect either the quality (availability, reliability, and security) or the functional and performance features (usability and maintainability) of a software system.

Different kinds of software entities contain specified sets of software evidences. A software project generally contains all development evidences of a software system. Software resources usually include all the sharing evidences of a software system, relevant parts of the development evidences and application evidences. A software instance contains all application evidences of an online software system.

2.3 Software Evolution Model

Software evolution is a continuous process of modifications to meet application requirements. It is an essential way towards producing quality software systems in crowd-based development. During the software evolution process, developers modify and upgrade the software system based on existing trustworthiness evidences. Evolution activities will also produce new trustworthiness evidences. Based on the new software lifecycle model mentioned in previous subsection, evolution activities fall into the following three categories: version evolution, resource evolution, and runtime evolution. Each corresponds to the development, sharing and application phase respectively. Given a specific software system, a version evolution can generate multiple instances of resource evolution that in turn generates multiple instances of runtime evolution as shown in Fig. 2a.

Version Evolution is the continuous source code evolution of a software system during the development phase. This includes the design, coding, testing and release activities carried out in the face of changing requirements, and different development processes might be involved. Version evolution activities produce development evidences and it has a dependency on trustworthiness evidences produced by resource evolution and runtime evolution activities.

Resource Evolution is the change of software trustworthiness attributes directly or indirectly caused by continuous updates of software trustworthiness evidences.

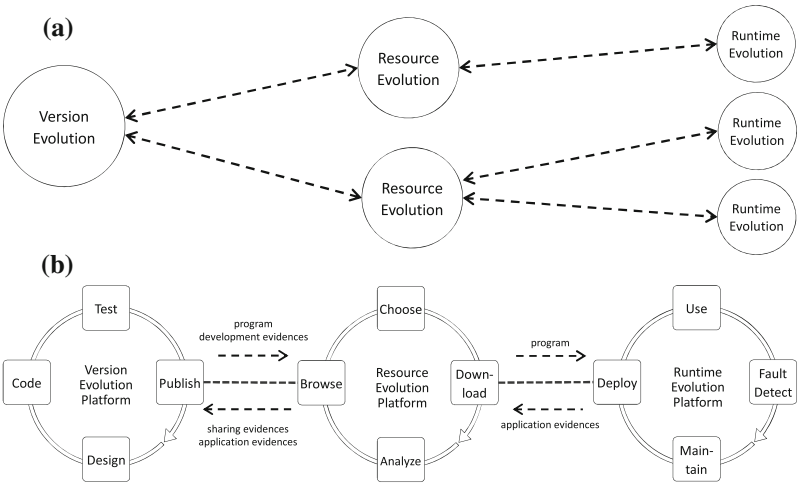

Fig. 2 Three evolution models: **a** relations between three kinds of evolution. **b** The internal actions in software evolution loops

The basic steps of resource evolution include the update of programs, evidences and the recalculation of trustworthiness attributes.

Runtime Evolution is the change of software runtime activities including software update, application deployment, system maintenance, error handling and so on. Systems of different types or different scales evolve differently in their application phase. Runtime evolution provides trustworthiness evidences generated in software systems' application phase to resource evolution activities.

There are complicated mutual impacts and restrictions between different evolution activities. Malfunction of any type of evolution activity can cause serious negative or harmful effect to the development of the software system. For example, the more runtime evolution instances, the more trustworthiness evidences they can provide to resource evolution instances and the quicker version evolution activities such as bug fixing, hence the faster the maintenance and improvement process. Version evolution is the original driving force of other evolution activities; malfunctioned version evolution activities like poor project management might lead to software runtime failure and sometimes force teams to start new version evolution activities. Besides, resource evolution should focus on the aggregation, sharing and analysis of software evidences. Suitable resource evolution mechanisms should be designed to attract participations of software vendors and users.

Under specific circumstances, software vendors might directly offer their software system as online services, which to some extend brings the above three types of evolution activities together. This integration is meaningful to the improvement of the overall efficiency and quality of crowd-based software evolution activities.

3 Trustworthy Software Development and Evolution Service Model

The TSDESM (Trustworthy Software Development and Evolution Service Model) is an Internet-based software evolution model. It supports software evolution activities in the development, sharing and application phases, and the formation, gathering, sharing and utilization of trustworthiness evidences. It provides an evolution-based approach for crowd-based software development paradigm. As shown in Fig. 3, this service architecture centers on *trustworthiness analysis*, offers *crowd collaboration, resource sharing* and *runtime monitoring* as basic services. Specifically, the crowd collaboration service supports crowd creation and manufacture, and provides mechanisms for the integration or transformation between the two. The resource sharing service provides mechanisms, like entity sharing and evidence sharing, to achieve rapid prototype distribution and application feedback. The runtime monitoring service provides services of data gathering, aggregation and analysis service during the runtime. The trustworthiness analysis service is responsible for measuring and analyzing the data generated in the crowd collaboration service, runtime monitoring service and the resource sharing service. Meanwhile, it provides comprehensive analysis mechanisms for various crowd-based development tasks.

The basics of this architecture are the massive amount of trustworthiness evidences generated in different software evolution processes. The four services not only output different types of trustworthiness evidences, but also establish and optimize some of their own functionalities by utilizing trustworthiness evidences, as depicted in Fig. 3. Specifically, collaboration development activities are supported by the crowd collaboration service. The development process data they produce are the main source of development evidences, including version repositories, code commit logs, and

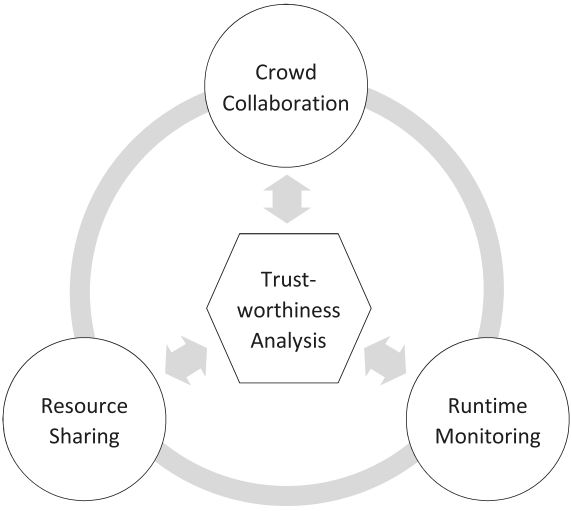

Fig. 3 Trustworthy software development and evolution environment architecture

issue tracking repositories. The resource sharing service continuously aggregates and accumulates trustworthiness evidences from the crowd, including test reports, use case description and comments. The runtime monitoring service outputs behavior data of software instances, such as running log, which is important for evaluating and improving the runtime trustworthiness. The trustworthiness analysis service is responsible for measuring, analyzing and evaluating various evidences, providing developers and users with important statistical results and analysis tools.

Besides, the four services are not isolated. They provide services to each other through open interfaces. For example, through the interfaces provided by the resource sharing service, the crowd collaboration service can recommend useful software components and services to programmers to facilitate agile development. The resource sharing service can also call the interfaces of the trustworthiness analysis service to get the trustworthiness evidences of a certain software resource. The crowd collaboration service uses these interfaces to evaluate on-going development activities in the code quality and development efficiency. And the runtime monitoring service can publish logs of critical system faults to resource sharing service, which in turn publishes these log data to crowd for possible solution.

3.1 Crowd Collaboration Service

Software is the virtualization of real world objects and the incarnation of knowledge and wisdom. All software development activities are essentially a kind of knowledge-intensive collaboration activities [12, 13], be it software manufacture or software creation. However, these two types of collaborations are different in many aspects. Collaborations in software manufacture activities are conducted by a closed team of developers, while collaborations in software creation often involve the external crowds. This difference entails different mechanisms and tools for development, interaction and rewarding systems.

The goal of the crowd collaboration service is to support the integration or transformation of software creation and software manufacture. As an indispensable procedure in software creation activity, it means to make adaptation of selected works of creation and integrate them into products of manufacture. Online communication and sharing tools like BBS, blog, wiki, micro-blog, which can support collaborations of a large crowd, are important for software creation activities. These tools can help disseminate creative ideas and works in a short period of time, attracting more potential users and contributors. Meanwhile, tools used in software manufacture are mainly aimed at improving the level of development automation and better process management. These include tools of desktop development, version management, process management, bug management and so on.

The crowd collaboration service should provide the development tools of both types and the mechanisms for their integration. These mechanisms can resolve their inter-dependencies and conflicts. Besides, these mechanisms should be flexible and adaptable, especially for projects which adopt engineering management on core

members and crowd management on peripheral contributors. Here we propose three reference models.

Community Model is the mechanism that supports communication and sharing among a group of individuals with similar interests. Community tools include BBS, blog and wiki and so on. The community model integrates community tools with software manufacture tools, builds development communities around development processes or software artifacts. For example, developers and users can create a community for sharing and communicate development activities based on a certain software issue. The community mechanism supports and encourages individual developers to be creative and in turn inspires crowd wisdom. See Fig. 4a.

Social Network Model is the mechanism for maintaining social relationships among users. Basic elements of a social network are relations like "follow" , "friend" and "group". Users broadcast their dynamics to other relevant users through the social network. Often, there are complicated social relations among software developers, such as code committer network [14] and bug report network [15]. Social network tools are mainly responsible for maintaining social relationships in software development activities, achieving mutual awareness and seamless interaction. As depicted in Fig. 4a.

Process Organization Model is the mechanism for organizing and reusing efficient collaboration processes and tools for software development activities with a relatively stable work flow. Software production line is a new network-based software development environment which is integrative, extensible and collaborative [16]. Following a given development procedure, this new environment can organize and customize software development elements related to developers, tools and artifacts.

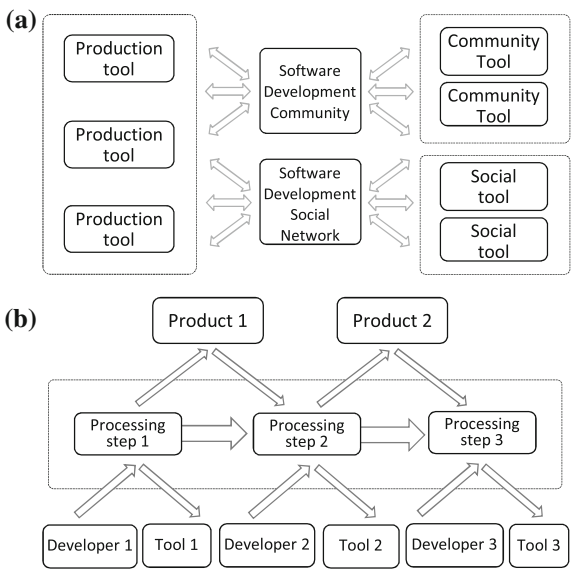

Fig. 4 Network-based crowd collaboration service: **a** the two integration models for software creation and production. **b** The tool integration model for development progress organization

In this way, it can easily customize a dedicated software development environment for a particular team of developers. As depicted in Fig. 4b.

3.2 Resource Sharing Service

During the evolution of a software project, developers and users create various artifacts, tools and data. These resources are valuable to reuse and reference in later development. Resource Sharing is an essential utility for both software manufacture and software creation. For software manufacture activities, sharing of artifacts, processes and information within a certain closed-team project should be supported and encouraged in the platform. On the other hand, to attract a larger crowd to participate in software creation activities, the platform should also encourage and remind core developers to share tutorials and example codes. Oftentimes, resources and knowledge shared in crowd development are not well-structured documentations and goes beyond the boundary of any project or team. This implies that the platform should have free sharing utility for the crowd to upload, edit, mark, comment and vote for or against various kinds of resources.

The recourse sharing service is the major platform for software resource evolution. It can provide software program and evidence sharing utility to different groups of developers. The challenge here is to continually collect and store massive amounts of software resources. Two mechanisms are introduced in this service.

Program Sharing Mechanism supports publishing, accessing and updating of software components, software services and other types of software programs. For software services, the software entity data also include the interface descriptions and URL addresses of each service instance. For open source software, software entity data often include source code, compilation or installation scripts and the address of the source code repository. The software program sharing mechanism can accelerate software system's distribution and dissemination. In other words, it accelerates a software instance's transformation from version evolution to runtime evolution, speeding up the exposure of software bugs and other issues.

Evidence Sharing Mechanism supports publish, access and update of software trustworthiness evidences generated in the development, sharing and application phases. For software services, their evidences also include service instances' real-time availability status and operation status. For open source software systems, the evidences include bug repositories, mailing lists, open source licenses, sponsors' information and the activeness of development. The software evidence sharing mechanisms can speed up software evidences' dissemination and update; shorten the response time on user feedbacks.

Currently, software resources are widely dispersed over various online software resource sites. Take open source software resource sties as an example. They include software community sites (e.g., *Linux*, *Apache* and *Eclipse*), project hosting sites (e.g., *SourceForge* and Github), software directories (e.g., *Softpedia.net* and *Ohloh*) and programming Q&A websites (e.g., *StackOverflow* and *AskUbuntu*). These sites

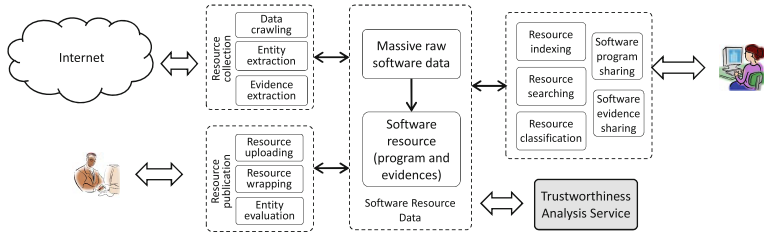

Fig. 5 Core mechanisms of the software resource sharing service

contain huge amounts of publicly accessible software programs and various types of trustworthiness evidences. These data are of different formats and their organizations are different from one site to another, so it is challenging to discover quality software resources (Fig. 5).

There are several techniques that can be used to collect massive quality software resources. These include:

Resource Publication allows users to register and submit various software entities and their trustworthy evidence to the resource sharing platform. Different types of software resource data are organized and wrapped by a unified structure, e.g., the RAS mechanism [17].

Resource Collection crawls and preprocesses software resources from various online software resource platforms. It is an important automatic technique for establishing the large-scale software resources sharing service.

Resource Organization and Mining supports effective classification and retrieval of massive software resources, and mines the data for software evaluation and analysis based on trustworthiness evidences.

3.3 Runtime Monitoring Service

The runtime monitoring service is an infrastructure for achieving trustworthy runtime evolution in the application phase. The idea is to monitor the behavior and status of software instances, and provide raw or filtered runtime log data for trustworthy analysis service. By providing such information, the runtime monitoring service can support fault localization and dynamic modification, and eventually support trustworthy running of software systems.

The runtime monitoring service can be implemented in three modules as shown in Fig. 6.

Monitoring Development Tool transfers normal software into monitoring-enabled software. A general transformation approach is to insert monitoring probes into the targeted software system. Specifically, the monitoring-enabled transformation process contains several sequential phases, including monitoring requirement description, monitoring probes generation, monitoring probes insertion and

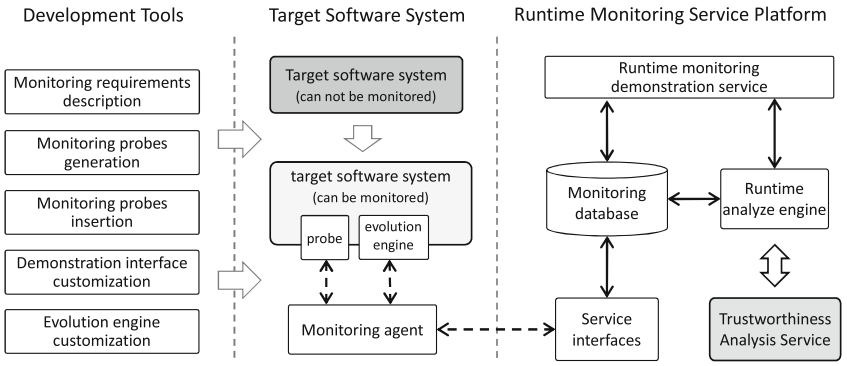

Fig. 6 The model of software runtime monitoring service

software interface customization. In addition, the monitoring development tool can also support dynamic adjustment by deploying evolution engine into a software system. However, the implementation of evolution engine needs API support of targeted software system.

Targeted Software System contains three general components. The monitoring probes send system runtime status to a monitoring agent which deliveries these information into the runtime monitoring service platform. For software systems that change frequently, the targeted software system can deploy an evolution engine to implement dynamic adjustment. In details, the evolution engine can either execute evolution commands sent from the monitoring agent, or automatically adjust system based on local monitoring data.

Runtime Monitoring Service Platform is a system-level remote service that supports monitoring of several targeted software systems. It contains APIs of monitoring service, runtime analysis engine and runtime monitoring demonstration service and so on. The monitoring database is responsible for storing raw monitoring data and recognized faulty event data. For very large complex software system, the monitoring database also needs mechanism to process stream data and the capacity of massive data storage. The runtime analysis engine conducts data mining on monitoring data through calling the APIs of trustworthy analysis service, to evaluate system running status, diagnosis of system faults, and update monitoring database. The monitoring agent can send monitoring data, acquire fault information or issue adjustment commands through accessing the APIs of monitoring service.

The runtime monitoring service can be regarded as a new service provided to software system running in the network. By doing so, this not only simplifies application logic of targeted software system, but also unleashes the advantage of data mining to improve the effectiveness of monitoring.

3.4 Trustworthiness Analysis Service

Trustworthiness analysis measures and evaluates various development behaviors, software artifacts and components by mining massive software evidence data. Resources and knowledge generated by software creation activities are neither well-managed nor well-organized, thus posing challenges for effective data mining. For example, in crowd wisdom methods, most requirements are hidden in comments and discussions informally created by the crowd. Thus, data from crowd have to be preprocessed and analyzed to be properly reused and referenced in software manufacture activities. Trustworthiness analysis service is the key to disclose and refine them, making the transformation from creation to manufacture possible. Besides, for both engineering methods and crowd wisdom methods, analysis is also crucial for understanding development status and evaluating development problems. Finally, results of analysis often act as useful references for optimizing future development. For example, the analysis and monitoring of user feedback has become the norm to evaluate existing software features and extract new requirements [18].

In the proposed software model, trustworthiness analysis service is the fundamental mechanism that makes trustworthy evolution during the development and sharing phase possible. It measures and assesses development behaviors, software artifacts and software products by analyzing and mining massive software evidence data, thus leading software evolution activities towards the desired direction. The trustworthiness analysis service contains three core mechanisms, i.e., development data analysis (for development evidences), runtime data analysis (for trustworthiness evidences) and resource trustworthiness rating (for runtime evidences).

Development Data Analysis consists of mechanisms which analyze software development data (including process data and work-in-process) to measure and assess software development activities and ultimately help improve project development efficiency and software product quality. The core model of these mechanisms is given in the left triangle of Fig. 7. The task of development evidence extraction is to extract relevant development data from the software project environment. This process in turn imposes new requirements on the reorganization or adjustment of software project process data. The development data analysis mechanism is to analyze

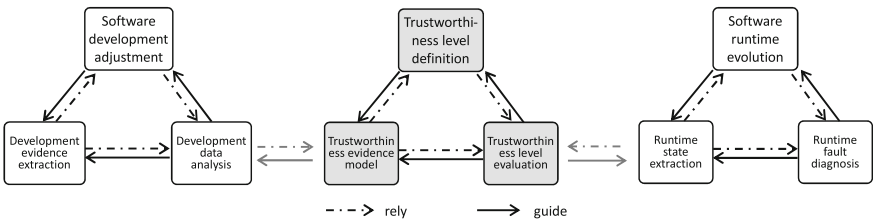

Fig. 7 The core model of trustworthy software comprehensive analysis (from *left* to *right*): **a** development data analysis model of software projects. **b** Trustworthiness evaluation model of software resources. **c** Runtime data analysis model of software instances

and measure the extracted data. This helps assess development status and identify software problems, providing important statistics for development optimization.

Resource Trustworthiness Evaluation gives the estimated rating of the target software resource for the given trustworthiness attribute, by analyzing and evaluating the evidences generated in the three phases of software life cycle. The core model of these mechanisms is depicted in the central triangle Fig. 7. The rating of software trustworthiness is based on the trustworthiness rating requirement model, which can be established according to users' expectation. Trustworthiness evidence model is the set of pertinent evidences determined by the definition and assessment process of trustworthiness rating. It is the basis of software trustworthiness rating. Trustworthiness rating assessment is the method and mechanism which rates software entities' trustworthiness, which is often domain specific. An example of the trustworthiness rating from TRUSTIE can be seen in Sect. 4.3.

Runtime Data Analysis analyzes and mines the application evidences generated in software runtime to evaluate system running status and diagnose faulty. Bo doing so, we can dynamically adjust software system to achieve trustworthy running and evolution of software. The core model of these mechanisms is given in the right triangle of Fig. 7. The runtime state extraction component extracts and analyzes important evidences related to the analyzed targets, including application independent system-level evidences and application dependent logic-level evidences. The runtime fault diagnosis component mainly analyzes and localizes evidences, and diagnoses software runtime faults. The recognized faults can be taken as runtime evolution evidences, and be published into resource sharing platform as application evidences.

For the construction of any specific trustworthy software development environment, the software project data analysis model and resource trustworthiness rating model in Fig. 7 are widely applicable. Software development data can be intermediate software artifacts like source code files or executable software modules. They can also be procedure logs of a specific project task like development logs, issue lists and mailing lists. The software trustworthiness rating model can explicitly give a software system's trustworthiness rating and its definition. It may also give an unsupervised ranking requirement according to some trustworthiness attribute. The measurements and descriptions produced in the software project data analysis process are important evidences of the software development phase.

4 TRUSTIE: Software Production and Evaluation with Crowdsourcing

Based on the trustworthy software development service model, we designed and implemented TRUSTIE (**Trust**worthy software tools and **I**ntegration **E**nvironment), a platform for software production and evaluation with crowdsourcing (www.trustie.net).

The goal of TRUSTIE is to help internal development teams and external crowd developers to improve the quality and the productivity of software. To achieve this goal, TRRUSTIE uses software crowdsourcing to bridge the external crowd wisdom and internal engineering by using various contributions from the crowds, which are tasks that can be performed distributedly beyond the internal development team for the software projects. In TRUSTIE, any software development tasks can be outsourced in an implied manner. Even the evaluations of the submitted contributions are possible to be outsourced. The organizers of the outsourcing mechanism are mainly the internal teams of the software projects (Fig. 8).

TRUSTIE employs five kinds of technologies, achieving the key mechanisms of bridging the external crowd wisdom and internal engineering process, supporting central control, decentralized contract, and three application models. The platform has developed about 60 software tools covering software development activities including requirement engineering, design, packaging and maintenance. We have designed the system of software collaborative development analysis and the system of software product-line framework. Based on the former system, we developed four product-line systems for automatic software production. We also achieve the integration of the collaborative development core service with those systems. The technologies and platform of TRUSTIE have been used in China in various software industries and research institutions.

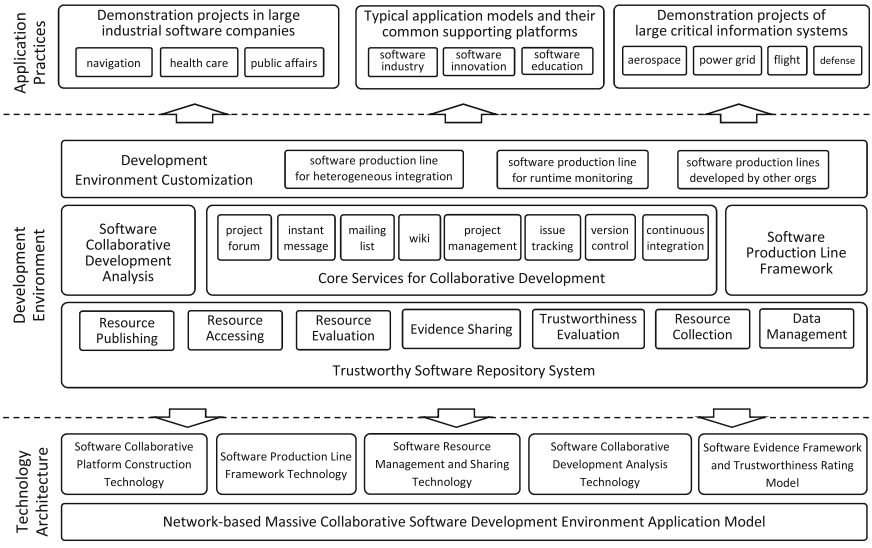

Fig. 8 The development environment, technology architecture and application practices of TRUSTIE

4.1 Software Crowdsourcing Model and Process in TRUSTIE

Based on the theoretical framework proposed in [19], the software crowdsourcing model in TUSTIE can be categorized as a non-competitive model. In TRUSTIE, either individuals or development teams can participate in a *collaborative* manner to create software. All the participants contribute code or resources throughout the entire crowdsourcing process for better connection of internal software teams and external Crowds. This is quite different from the competitive model (such as crowdsourcing platforms in *TopCoder* and *AppStori*), where people participate in a *competitive* manner to create software. Only selected or highly talented developers (or teams) can survive in the crowdsourcing process and become the only contributors or be rewarded with funding.

As a non-competitive model, the process of software crowdsourcing in TRUSTIE is to ease the collaboration burden and maximize the throughput of development outcome. This is supported by the collaborative development service and resource sharing service (explained later in Sects. 4.2 and 4.3) in TRUSTIE in an implied style. Generally, TRUSTIE includes three crowdsourcing processes: (1) Development outsourcing: TRUSTIE outsources the task pieces of software creation and production to the crowds by using its network-based crowd collaboration service. Currently, TRUSTIE is considering outsource the codes review tasks [20]. (2) Testing outsourcing: TRUSTIE enables the collection and integration of public bug reports and comments by software resource sharing service. (3) Maintenance outsourcing: TRUSTIE enables the collection of runtime status of monitored systems for quick runtime evolution. This may enable the outsourcing of system maintenance tasks by recruiting skilled system administrators. Currently, these processes are mainly driven by the interests and consumption requirements of the crowds.

The collaborative development service and resource sharing service are the keys to support the non-competitive crowdsourcing model in TRUSITE. In the next two subsections, we describe the two services in details.

4.2 Trustworthy Software Collaborative Development Service

The trustworthy software collaborative development service supports the crowdsourcing process with software creation tools and software manufacturing tools. These tools are exposed to both internal development teams and external crowd with configurable options. Their integration is achieved through the community model and social network model. Based on this, we build the collaborative development analysis system that can analyze and measure development behavior, and the software product-line framework system that supports the organization of collaboration process and the customization of the development environment. Specifically, it consists of the following services:

Social Collaborative Development Service: This service provides software creation tools like project forums, collaborative editing tools, mailing lists, and instant communication. It also provides manufacture tools like project management, configuration management, bug management and continuous integration tools. It combines both kinds of tools into a unified development environment [21]. Besides, this service supports the sharing of technical knowledge, and achieves developers' mutual awareness, and interaction through mechanisms applied in the community model and the social network model.

Collaborative Development Analysis Service: this service has provided a platform to fetch and store massive software development data. It has also integrated various tools for measuring development behaviors into the configuration management tool set. Behind this service is a comprehensive assessment technology which concerns the evidences of software products and evidences of development teams simultaneously [22]. Besides, it integrates a technology that analyzes a programmer's development capability and the traces of his or her technical improvement [23].

Software Product-Line Construction Service: specifically, this service organizes and customizes developers, software tools and software artifacts involved in a specific software development process based on some given development steps. In this way, the service helps establish suitable software production lines and a dedicated development environment for the developer team [24].

As of June 2013, TRUSTIE supported 600 software projects and 700 competition projects. Besides, by using a set of evidence standards, it has extracted the development evidences of quality OSS projects from online OSS communities.

4.3 Trustworthy Software Resource Sharing Service

In TRUSTIE, the trustworthy software resource sharing service supports the crowdsourcing process by allowing users, either internal development teams or external crowd, to publish, retrieve, and evaluate software resources. It can transfer the traditional *closed static* software component storage model into the *open dynamic* software resource sharing model [25]. Combined with the software rating and evaluation model [26], this mechanism integrates trustworthiness evidence framework into the software resource information architecture [27]. It seamlessly integrates central, static resource storage with open, dynamic resource aggregation technology.

Resource Collecting and Organization: this service collects massive software resources by using the resource publishing and resource retrieval technology. Based on the RAS technology standard, it achieves evidences' packaging and utilization throughout the whole software life cycle. In order to improve the effectiveness of resource management, it classifies massive software resources from multiple sites using text mining and tag mining technology [28] and feature analysis technology [29]. Both extracted meta data and mined knowledge data is stored and indexed in cloud-based persistent storage including relational and non-relational databases. Up to June 2012, TRUSTIE has already published 170 self-developed software

resources and more than 61,000 software resources collected from other online software libraries and open source communities.

Resource Trustworthiness Evaluation: this service provides automatic assessment mechanisms and manual assessment mechanisms, e.g., the assessment mechanism for service trustworthy evolution [30, 31]. Though software trustworthiness rating depends on specific application domain, we believe there can be a trustworthiness rating reference model which captures universal software attributes. This reference model can be customized for any specific application domain.

The trustworthiness rating model employed by TRUSTIE's trustworthy software resource repository system is a trustworthiness rating reference method whose rating dimensions are from user expectation, application validation and the methodology of evaluation [26], see Table 1. For example, in some critical areas (like aerospace), even

Table 1 Software trustworthiness rating model

Level	Name	Content
0	Unknown	The lowest trustworthiness level. It means no software trustworthiness evidence is found. It cannot determine whether the target software system satisfies users' expectations of the trustworthiness attributes of the same type of software systems
1	Available	The software entity can be accessed, and can function as described by the software provider. It implies that the target software system satisfies users' basic expectations over the functional attributes of the same type of software systems
2	Verified	On the basis of the Available level, software provider publishes a declaration of the set of software trustworthiness attributes according to some documented standards. This declaration can be verified through domain-specific software assessment mechanisms. It indicates that the target software system satisfies users' general expectations over the trustworthiness attributes of the same type of software systems, and these user-expected trustworthiness attributes are verified
3	Applied	On the basis of the Verified level, software systems have been applied in related domains and have verifiable cases of successful application. It implies that the software system satisfies users' general expectations over the trustworthiness attributes of the same type of software systems, and these user-expected trustworthiness attributes have been verified by real application
4	Assessed	On the basis of the Applied level, trustworthiness of an Assessed software system should pass the assessment conducted by authoritative software trustworthiness rating agencies according to specific documented trustworthiness rating standards. This indicates the software system satisfies relatively higher user expectations over the trustworthiness attributes of the same type of software systems, and these user-expected trustworthiness attributes are confirmed by authorities
5	Proved	The highest trustworthiness level. It means on the basis of the Assessed level, the user-submitted software trustworthiness attributes can be strictly proved

the first applicable version of the target software system is required to reach a high trustworthiness level (e.g., level 4). The prerequisite of reaching level 3 and above is the evidence of successful case application of the software system. This requires the trustworthiness rating standards of these areas to be customized according to domain-specific descriptions. For example, aerospace software users can specify the definition of "successful cases of application" as "successful experimentation under simulated environment".

To ensure the reliability of trustworthy evaluation, the data assessed by TRUSTIE platform are aggregated automatically from the software tools in TRUSTIE, such as issue tracking tools, version control tools, resource sharing tools and communication tools, without any intervention.

5 Related Work

Throughout the development of software technology, software development technology and its supporting environment have always played important roles in driving software technological innovation and industrial development. The rise of crowd-based open source development has brought new opportunities. The vendors of software development environments have shifted their attention from providing local development support to facilitating globally distributed crowd development.

In software development methodology and development environment architecture, researchers have studied the changes in software development technology. Yang and colleagues [13] realized the profound impact of the Internet on software systems and software development activities. They systematically proposed the Internetware model and a set of architecture-centered software technologies and development methods, which pave the way for establishing the conceptual model, the evolution process and the supporting environment of trustworthy software systems. Based on Sourceforge and other similar open source project hosting platforms, Booch and colleague [32] have described the definitions and basic characteristics of software collaborative development environments. Using OSS development and community-based service systems as prototypes, Kazman and colleague [33] have proposed the Metropolis model which facilitates crowdsourcing software development. The Metropolis model adopts a hierarchical software development architecture, where participants are divided into the core, the peripheral and the mass. It addresses principles like openness, mash-ups, unknown requirements, continuous evolution, unstable resources and emergent behaviors. Through numerous case studies, Tapscott and colleague [34] have indicated that software development and more and more other business models have begun to adopt the ideas and mechanisms of mass collaboration, including openness, peering, sharing, and acting globally. Herbsleb [2] has proposed the concept of Global Software Engineering. He discussed what new challenges global software development imply in aspects including software architecture design, requirement elicitation, development environments and tools. Possible future research directions have also been identified and illustrated.

For the crowdsourcing model, Howe [3] has illustrated its origin, present status and future with several real world examples. He listed the open source movement, the development of collaboration tools and the rise of vibrant community as the keys to the rise of crowdsourcing, which is meaningful in establishing the crowd-based software development ecosystem.

For collaborative development tools and technologies, recent years have witnessed numerous researches on the analysis of distributed, social development technologies and practices. Mockus et al. [35] have analyzed the software development data of the Apache Server open source project, which has a major impact in software engineering research and has become the pioneer work in the field of Mining Software Repositories. Crowston and Howison [36] have examined 120 OSS projects hosted on Sourceforge.com. They have identified different patterns of developer interactions and their relation with the team size. Sarma and colleague [37]have proposed a browser named Tesseract for visualizing 'socio-technical' dependencies in development activities, which aims at increasing mutual awareness among developers of distributed teams. Dabbish et al. [38] analyzed Github.com, a project hosting and social development website, and how transparency plays an important role in developer collaboration in Github. Posnett et al. [39] analyzed the focus and ownership relations between software developers and artifacts in distributed development. They propose a unified view of measuring focus and ownership by drawing the predator-prey food web from ecology to model the developer-artifact contribution network. They found through empirical studies that the number of defects is related to the distribution of developer focus. Bird et al. [40] analyzed the development process of Windows Vista. Specifically, they compared the post-release failures of components that are developed by collocated teams with those developed in a distributed manner. The difference is found to be insignificant. More recent researches on collaborative development analysis tend to focus on empirical study and aim at making constructive suggestions and possible improvements on existing collaboration tools.

In OSS, resource sharing, mining, and trustworthiness evaluation become important issues. SourceForge, Github and other project hosting sites have accumulated a huge number of projects and data. In 2009, Mockus and colleague [41] have conducted a preliminary census of OSS repositories. Their results indicate that more than 120,000 and 130,000 projects were then hosted on SourceForge.com and Github.com, respectively. In 2013, these two metrics are reported to be 470,000 and 4,000,000 respectively [42]. With the wide application of OSS, researchers began to focus on the measurement of OSS trustworthiness. In its essence, software trustworthiness is the natural extension of the notion of software quality in the Internet era [13]. How to scientifically assess software quality has always been one of the most challenging issues of software engineering research [43]. After 40 years of development, Software Metric has become an important software engineering research direction concerning the problems of software quality. Quality assessment technologies have become specialized and standardized. Numerous impactful software quality models were then proposed [44, 45]. Based on these models, researchers and practitioners have further designed quality models that take community factors into consideration. These include the Navica [46], OpenBRR [47] and SQO-OSS [48] models.

For example, the OpenBRR (Open Business Readiness Rating) model is a mature OSS quality assessment model which aims to rate software projects and the code of the entire OSS community in a standard and open fashion and eventually facilitates the evaluation and application of OSS. Its assessment categories include Functionality, Usability, Quality, Security, Performance, Scalability, Architecture, Support, Documentation, Adoption, Community and Professionalism. Its assessment process involves ranking the importance of categories or metrics, processing the data, and translating the data into the Business Readiness Rating. For the moment, assessment and utilization of online OSS resources are still a hot topic for SE researchers.

In the industry, there is a major trend of the integration of software development environment with online collaboration tools. CollabNet[4] is one of the software vendors who intentionally integrate OSS development methods into software development environments. It has published the TeamForge platform which integrates configuration management, continuous integration, issue tracking, project management, lab management and collaboration tools into a Web app life cycle management platform. In this way, it supports distributed collaborative development and high quality software delivery. The Visual Studio Integrated Development Environment[5] is an enterprise IDE for desktop development environment. Recently it has added TFS (Team Foundation Server) that supports team collaboration mechanisms like version control, iteration tracking and the task panel. IBM Rational Jazz[6] is an open and transparent collaborative development platform for commercial use. The team collaboration, requirements composition and quality management tools of Rational Jazz can support the development of trustworthy software products. Besides, IBM has once encouraged the use of the IIOSB (IBM's Internal Open Source Bazaar) [8] in its commercial software development environment, which we believe is an important attempt to integrate software creation and manufacture.

There are research efforts on approaches to harnessing crowd wisdom for software development. Some emphasizes the importance of open, decentralized management. Bird and colleague [49] found that the development of Firefox project is distributed both geographically and organizationally. According to interviews with the creators of Linux, Perl and Apache [50], letting the crowd takes over is an indispensable step for the success of OSS projects. Project owners are thus encouraged to set up mechanisms and generate utilities for a larger crowd to participate easily, rather than act against this openness. However, decentralization comes at a cost. Compared to traditional centralized, co-located projects, this globally distributed OSS development model must face the challenge of incompatibilities and the risk of lack of awareness [2]. To harness crowd wisdom, software projects should be equipped with tools and practices that meet increasing coordination needs. The importance of accommodating diversities and conflicts has also been addressed in OSS practices and researches. A typical OSS project uses issue tracking systems to manage bug reports and feature requests. These bug reports cover various aspects of the target software project, and

[4] http://www.collab.net.

[5] http://www.visualstudio.com.

[6] http://www-01.ibm.com/software/rational/jazz/.

some of them are conflicting with each other. However, there are no dictators who make arbitrary decisions to cast aside any of these bugs. Instead, on which advice to take is totally for the whole community to decide. Those not taken are also kept in the project memory, and may have the opportunity to get re-opened [51]. Similar mechanisms can also be seen in the way developers manage their code contribution. For many successful Git-based OSS projects like the Android project and projects on Github, contributors do not have to always follow the central code depot. They can independently code on their own branch of the project, and merge the code back as a patch whenever they want [52]. To accommodate diversity, project managers are recommended to set up mechanisms to foster a culture that encourages different opinions. Besides, communication tools are needed to resolve conflicts and build consensus. For example, the *Stack Overflow* uses a voting mechanism to identify high quality posts.

Recently, crowdsourcing software development or software crowdsourcing was coined to identify an emerging area of software engineering [53]. It is described to be an open call for participation in any task of software development, including documentation, design, coding and testing. These tasks are normally conducted by either members of a software enterprise or people contracted by the enterprise. But in software crowdsourcing, all the tasks can be assigned to anyone in the general public. Many software engineering researchers have studied the concept models, processes and common architecture of software crowdsourcing. Wu et al. [54] has studied two famous software crowdsourcing platforms, *TopCoder* and *AppStori*. By mining the *TopCoder* data, authors found that the number of participants and hours spent on competition are surprisingly smaller than expected. Clear problem definition, transparency, diversity have been pointed out as the key lessons learned from current software crowdsourcing. For both software crowdsourcing platforms, the Min-Max nature among participants has been found to be a key design element. In another paper [19], the authors proposed a novel evaluation framework for software crowdsourcing projects. In the framework, the Min-Max relationship is used as a major aspect in evaluating the competitions of crowdsourcing projects. In a previous *Dagstuhl* Seminar [55], researchers from different domains have spent collective effort exploiting and validating the new idea of *Cloud-based Software Crowdsourcing*, where the software crowdsourcing processes and models are achieved with computer cloud support. Possible common architecture for *Cloud-based Software Crowdsourcing* has been identified. Important issues related to concept models, processes and design patterns have also been addressed. As discussed in the study of Tsai et al. [56], software crowdsourcing has enabled the synergy architecture between a cloud of machines and a cloud of humans. In such architecture, crowdsourcing models including game theory, optimization theory and so on would be well supported by cloud-based tools.

6 Conclusion

This paper proposes an ecosystem framework to deeply integrate the traditional engineering methodology and the crowd-based development process. Based on this framework, we develop the TRUSTIE, a non-competitive software crowdsourcing platform. It supports crowd collaboration, resource sharing, runtime monitoring, and trustworthiness analysis for trustworthy software evolution. TRUSTIE has been used successfully in a number of software companies in China since 2008. Our future work includes improving the evidence management and analysis capability of TRUSTIE through infrastructure upgrade, improving the collaborative development service and resource sharing service, and exploring the possibility of integrating competitive crowdsourcing models, such as creative works competition.

Acknowledgments This research is supported by the National High Technology Research and Development Program of China (Grant No. 2012AA011200), and National Natural Science Foundation of China (Grant No. 61432020 and 61472430). Our gratitude goes to all members of the TRUSTIE project, for their hard work and contribution, and also to the experts from the information technology domain of the National 863 Plan, for their continuous support and guidance.

References

1. DeMarco, T., Lister, T.: Peopleware-Productive Projects and Teams. Dorset House Publishing Co., New York (1987)
2. Herbsleb, J.D.: Global software engineering: the future of socio-technical coordination. In: 2007 Future of Software Engineering, pp. 188–198. IEEE Computer Society (2007)
3. Howe, J.: Crowdsourcing: Why the Power of the Crowd is Driving the Future of Business. Random House, New York (2008)
4. Giles, J.: Internet encyclopaedias go head to head. Nature **438**(7070), 900–901 (2005)
5. Torvalds, L.: The linux edge. Commun. ACM **42**(4), 38–39 (1999)
6. Surowiecki, J.: The Wisdom of crowds: why the many are smarter than the few and how collective wisdom shapes business. Economies, Societies and Nations (2004)
7. Lakhani, K.R., Panetta, J.A.: The principles of distributed innovation. Innovations **2**(3), 97–112 (2007)
8. Capek, P.G., Frank, S.P., Gerdt, S., Shields, D.: A history of IBM's open-source involvement and strategy. IBM SYST. J. **44**(2), 249–257 (2005)
9. Raymond, E.: The cathedral and the bazaar. Knowl. Technol. Policy **12**(3), 23–49 (1999)
10. Xu, J.: System Programming Language. China Science Press, Beijing (1987)
11. Hasselbring, W., Reussner, R.: Toward trustworthy software systems. Computer **39**(4), 91–92 (2006)
12. Robillard, P.N.: The role of knowledge in software development. Commun. ACM **42**(1), 87–92 (1999)
13. Yang, F., Lü, J., Mei, H.: Technical framework for internetware: an architecture centric approach. Sci. China Ser. F: Inf. Sci. **51**(6), 610–622 (2008)
14. Huang, S.K.: Mining version histories to verify the learning process of legitimate peripheral participants. ACM SIGSOFT Softw. Eng. Notes **30**(4), 1–5 (2005)
15. Zanetti, M.S., Scholtes, I., Tessone, C.J., Schweitzer, F.: Categorizing bugs with social networks: a case study on four open source software communities. In: Proceedings of the 2013 International Conference on Software Engineering, pp. 1032–1041. IEEE Press (2013)

16. Dou, W., Wei, G.W., Wei, J.C.: Collaborative software development environment and its construction method. J. Front. Comput. Sci. Technol. **5**(7), 624–632 (2011)
17. TrustieTeam: Trustie software resource management specification, trustie-srmc v2.0 (2011)
18. O'reilly, T.: What is web 2.0: design patterns and business models for the next generation of software. Commun. Strateg. (65) (2007)
19. Wu, W., Tsai, W.T., Li, W.: An evaluation framework for software crowdsourcing. Front. Comput. Sci. **7**(5), 694–709 (2013)
20. Yu, Y., Wang, H., Yin, G., Ling, C.: Reviewer recommender of pull-requests in GitHub. In: 2014 30th IEEE International Conference on International Conference on Software Maintenance and Evolution (ICSME 2014 TOOLS), pages to appear. IEEE (2014)
21. TrustieTeam: Trustie collaborative development environment reference specification, trustie-forge v2.0 (2011)
22. Lin, Y., Huai-Min, W., Gang, Y., Dian-Xi, S., Xiang, L.: Mining and analyzing behavioral characteristic of developers in open source software. Chin. J. Comput. **10**, 1909–1918 (2010)
23. Zhou, M., Mockus, A.: Developer fluency: achieving true mastery in software projects. In: Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 137–146. ACM (2010)
24. TrustieTeam: Trustie software production line integration specification, trustie-spl v3.1 (2011)
25. Zhao, J., Xie, B., Wang, Y., Xu, Y.: TSRR: A software resource repository for trustworthiness resource management and reuse. In: SEKE, pp. 752–756 (2010)
26. TrustieTeam: Trustie software trustworthiness classification specification, trustie-stc v2.0 (2011)
27. Cai, S., Zou, Y., Shao, L., Xie, B., Shao, W.: Framework supporting software assets evaluation on trustworthiness. J. Softw. China **21**(2), 359–372 (2010)
28. Wang, T., Wang, H., Yin, G., Ling, C.X., Li, X., Zou, P.: Mining software profile across multiple repositories for hierarchical categorization. In: 2013 29th IEEE International Conference on Software Maintenance (ICSM), pp. 240–249. IEEE (2013)
29. Yu, Y., Wang, H., Yin, G., Liu, B.: Mining and recommending software features across multiple web repositories. In: Proceedings of the 5th Asia-Pacific Symposium on Internetware, p. 9. ACM (2013)
30. Shao, L., Zhao, J., Xie, T., Zhang, L., Xie, B., Mei, H.: User-perceived service availability: a metric and an estimation approach. In: IEEE International Conference on Web Services, ICWS 2009, pp. 647–654. IEEE (2009)
31. Zeng, J., Sun, H.L., Liu, X.D., Deng, T., Huai, J.P.: Dynamic evolution mechanism for trustworthy software based on service composition. J. Softw. **21**(2), 261–276 (2010)
32. Booch, G., Brown, A.W.: Collaborative development environments. Adv. Comput. **59**, 1–27 (2003)
33. Kazman, R., Chen, H.M.: The metropolis model a new logic for development of crowdsourced systems. Commun. ACM **52**(7), 76–84 (2009)
34. Tapscott, D., Williams, A.D.: Wikinomics: How Mass Collaboration Changes Everything. Penguin, New York (2008)
35. Mockus, A., Fielding, R.T., Herbsleb, J.: A case study of open source software development: the apache server. In: Proceedings of the 22nd International Conference on Software Engineering, pp. 263–272. ACM (2000)
36. Crowston, K., Howison, J.: The social structure of free and open source software development. First Monday **10**(2) (2005)
37. Sarma, A., Maccherone, L., Wagstrom, P., Herbsleb, J.: Tesseract: interactive visual exploration of socio-technical relationships in software development. In: IEEE 31st International Conference on Software Engineering, ICSE 2009, pp. 23–33. IEEE (2009)
38. Dabbish, L., Stuart, C., Tsay, J., Herbsleb, J.: Social coding in GitHub: transparency and collaboration in an open software repository. In: Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work, pp. 1277–1286. ACM (2012)
39. Posnett, D., D'Souza, R., Devanbu, P., Filkov, V.: Dual ecological measures of focus in software development. In: 2013 35th International Conference on Software Engineering (ICSE), pp. 452–461. IEEE (2013)

40. Bird, C., Nagappan, N., Devanbu, P., Gall, H., Murphy, B.: Does distributed development affect software quality?: An empirical case study of windows vista. Commun. ACM **52**(8), 85–93 (2009)
41. Mockus, A.: Amassing and indexing a large sample of version control systems: towards the census of public source code history. In: 6th IEEE International Working Conference on Mining Software Repositories, MSR'09, pp. 11–20. IEEE (2009)
42. Begel, A., Bosch, J., Storey, M.A.: Social networking meets software development: perspectives from github, msdn, stack exchange, and topcoder. Softw. IEEE **30**(1), 52–66 (2013)
43. Liu, K., Shan, Z., Wang, J., He, J., Zhang, Z., Qin, Y.: Overview on major research plan of trustworthy software. Bull. Natl. Nat. Sci. Found. China **22**(3), 145–151 (2008)
44. Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., MacLeod, G.J., Merrit, M.J.: Characteristics of Software Quality, vol. 1. North-Holland Publishing Company, Amsterdam (1978)
45. McCall, J.A., Richards, P.K., Walters, G.F.: Factors in Software Quality. General Electric National Technical Information Service, Berlin (1977)
46. Golden, B.: Succeeding with Open Source. Addison-Wesley Professional, Boston (2005)
47. Wasserman, A., Pal, M., Chan, C.: The business readiness rating model: an evaluation framework for open source. In: Proceedings of the EFOSS Workshop, Como, Italy (2006)
48. Russo, B., Damiani, E., Hissam, S., Lundell, B., Succi, G.: Open Source Development, Communities and Quality. Springer, Berlin (2008)
49. Bird, C., Nagappan, N.: Who? Where? What? examining distributed development in two large open source projects. In: 2012 9th IEEE Working Conference on Mining Software Repositories (MSR), pp. 237–246. IEEE (2012)
50. Barr, J.: The paradox of free/open source project management (2005). http://archive09.linux.com/feature/42466. Accessed 6 May 2014
51. Aranda, J., Venolia, G.: The secret life of bugs: going past the errors and omissions in software repositories. In: Proceedings of the 31st International Conference on Software Engineering, pp. 298–308. IEEE Computer Society (2009)
52. Anonymous: Gerrit code review—a quick introduction, version 2.10-rc0-199-g60bca74 (2014). https://gerrit-review.googlesource.com/Documentation/intro-quick.html
53. Anonymous: Crowdsourcing software development, from Wikipedia (2014). http://en.wikipedia.org/wiki/Crowdsourcing_software_development
54. Wu, W., Tsai, W.T., Li, W.: Creative software crowdsourcing: from components and algorithm development to project concept formations. Int. J. Creat. Comput. **1**(1), 57–91 (2013)
55. Huhns, M.N., Li, W., Tsai, W.T.: Cloud-based software crowdsourcing (dagstuhl seminar 13362). Dagstuhl Rep. **3**(9) (2013)
56. Tsai, W.T., Wu, W., Huhns, M.N.: Cloud-based software crowdsourcing. IEEE Internet Comput. **18**(3), 78–83 (2014). http://doi.ieeecomputersociety.org/10.1109/MIC.2014.46