Chapter 9
Crowd-Based Methodology of Software Development in the Internet Era

Huaimin Wang, Gang Yin, Tao Wang and Yue Yu

Abstract In today's Internet era, software has infiltrated all aspects of people's lives, the trend of software-defined everything is essentially unstoppable. The classical methodologies in software engineering are expected to produce software at a low cost and with strong functionality by guiding the development process using industrialization methods and principles. However, as the complexity of software application scenarios and operating environments continues to increase, especially in the Internet era, prominent bottlenecks remain in improving the efficiency and quality of software development. Compared to the engineering methods, open source can attract tens of thousands of contributors to participate in the software creation process. This methodology is more deferential to each developer's individuality and aims to create a liberal, diverse, and democratic environment, thus stimulating the enthusiasm and creative inspiration of contributors on a large scale and ultimately generating greater collective wisdom. But the challenges in the diversification of individual interest concerns, the unevenness of contribution capabilities, and the unpredictable results of group collaboration make it unable to fully fulfill the tasks of clear and organized software manufacturing. In this chapter, we propose a crowd-based methodology that integrates the software creation process into the software manufacturing process, link a small-scale but well-organized core team with self-organized but large-scale crowd contributors, and transform a software opus to products in a timely fashion. Based on the crowd-based methodology , we design and implement the TRUSTIE environ-

H. Wang (✉) · G. Yin · T. Wang · Y. Yu
National Laboratory for Parallel and Distributed Processing, School of Computer,
National University of Defense Technology, Changsha, China
e-mail: hmwang@nudt.edu.cn

G. Yin
e-mail: yingang@nudt.edu.cn

T. Wang
e-mail: taowang2005@nudt.edu.cn

Y. Yu
e-mail: yuyue@nudt.edu.cn

B. Fitzgerald et al. (eds.), *Towards Engineering Free/Libre Open Source Software (FLOSS) Ecosystems for Impact and Sustainability*,
https://doi.org/10.1007/978-981-13-7099-1_9

ment to support the construction of software ecosystem. We illustrate the framework and key technologies and present typical application practices in both proprietary companies and online communities.

9.1 Introduction

—"I have a good business idea, but need a programmer to implement it!"

The hottest buzzword in information technology today is "software-defined", spanning from software-defined networking (SDN), software-defined storage (SDS), and software-defined data center (SDDC), which are part of a broader trend that people might call software-defined everything. Looking around the world, the total market capitalization of the top five Internet companies, i.e., Apple, Amazon, Microsoft, Google, and Facebook, has exceeded 3700 billion U.S. dollars[1] in 2018. *Mechanical Turk*, *AlphaGo*, and other breakthrough products have been invented by those companies, directly driving the innovation development of global technology. Software technology is widely combined with the urgent needs of traditional industries to create extremely innovative business services to facilitate our daily life. For example, when software meets the transportation, *Uber* is created; when it meets the catering industry, *Yelp* is created; when it can be embedded in tangible products, many smart hardware products are created, e.g., *Google Glass* and *iWatch*. It may be hard for people today to imagine how uncomfortable will be with no software support in their lives. Similar to the books that carry the text civilization using written language in the past, various kinds of software have become a new expression of information civilization in the Internet era.

Definitely, we do not expect that a consummate solution and methodology for software development can be found overnight. With respect to the development of computer hardware technology, until today, no software development methodology can promote software evolution at the same speed (i.e., the development of hardware capabilities is in line with the growth of Moore's Law [1, 2], while software technology cannot be guaranteed.

Case of ARM Ecosystem. The ARM ecosystem deeply integrates software with hardware. In contrast to Intel's ecosystem which has accumulated its own sophisticated software platforms, the main problem of the construction of ARM ecosystem is how to transform and optimize various kinds of open source and commercial software to be better compatible with the ARM hardware in a high efficiency and quality way, assisting the relating companies in the ARM community in delivering their products to the market rapidly. However, the transformation and optimization processes cover almost all software stacks, including the operating system, database, web applications, and the infrastructures for cloud computing and big data, which are beyond the capability of traditional methodologies in software engineering.

[1]https://www.statista.com/statistics/277483/market-value-of-the-largest-internet-companies-worldwide/.

To address the software challenges in the Internet age, we propose crowd-based software development methodology and its supporting environment called TRUSTIE. Our main idea is based on the linking viewpoint, i.e., effectively linking different types of development activities and different types of development collaborators to improve the innovation efficiency of the software ecosystem and reduce the cost, thereby optimizing the business patterns of all stakeholders. The remainder of this chapter is organized as follows: Sect. 9.2 introduces the methodology of TRUSTIE and the key concepts underlying the crowd-based software development methodology. Section 9.3 presents the framework and typical algorithms in TRUSTIE, as well as the related support platform and tools. Section 9.4 describes the application of TRUSTIE.

9.2 The TRUSTIE Methodology

In this section, we illustrate the classical methodology of software engineering, open source, and the main idea of crowd-based software development methodology.

9.2.1 Software Engineering and Software Manufacturing

The main question in software engineering is how to continuously improve the development efficiency and software quality. Since the 1960s, software practitioners have noted that strong challenges have arisen from the backward mode of software production. To address the "software crisis" [3], the concept of "software engineering" [4] was proposed. Practitioners aimed to implement systematic mechanisms to manage software developers and development activities in the form of a "project". This strategy is expected to produce software at a low cost and with strong functionality and high quality by guiding the software development process using industrialization methods and principles. In this concept, there is no essential difference between software development and industrial activities, e.g., automobile manufacturing, garment production, and building construction. Both of these broad fields are expected to achieve increased efficiency through strict and precise task decomposition and personnel organization.

Under the guidance of this classic concept of software engineering, the academic and industrial communities have conducted continuous exploration and research on various aspects, e.g., development methodology, project management, and software architecture. These communities have proposed a series of classic methodologies, e.g., the waterfall model [5], the constructive cost model (COCOMO) [6], and component-based software engineering (CBSE) [7]. The most representative and comprehensive methodology is exemplified in the software product lines [8], which can be summarized in the following three steps: (1) extracting the public structure and characteristics of specific fields or similar products through domain engineering induction; (2) organizing developers to write code and assemble modules by formulating detailed and strict production plans based on standardized reusable software assets and the software development life cycle; and (3) building a batch of software products that meet the needs of specific markets.

In brief, we summarize the software development processes organized by the industrialization solutions as **Software Manufacturing**, in which the outcome is the software product or production-ready software. These approaches are very effective in organizing software development activities with relatively clear and stable targets. Over the years, such approaches have directly supported the smooth advancement of a series of technology-intensive projects, e.g., projects in the aerospace and aviation fields.

However, as the complexity of software application scenarios and operating environments continues to increase, the challenge of "no silver bullet" [9] has become increasingly prominent. Especially in the Internet era, software stakeholders have changed from small-scale specific groups to large-scale, dynamic, and open Internet users, leading to the recognition that the software development process is no longer composed of activities with clear and stable targets. For example, in an open environment, the requirements of large-scale groups cannot be frozen, i.e., the demands of users are dynamic. Additionally, in a complex scenario, software testing cannot fully cover the restricted search space, i.e., the test target is not clear. The software workers who are struggling on the software production line have also not escaped the challenges encountered by Chaplin's character in the film *Modern Times*. Even though the overall labor importation and workload of the software workers are maximized, prominent bottlenecks remain in improving the efficiency and quality.

9.2.2 Open Source and Software Creation

During the period over which other engineering methods have struggled for prominence, OSS has achieved remarkable success after decades of vigorous development. From the early operating systems of BSD and Linux to today's smartphone operating systems (Android), application container engines (Docker), and deep learning frameworks (TensorFlow), many high-quality OSS has gained more market share than similar commercial software [10].

Excellent open-source projects can attract tens of thousands of developers to participate in their development, which represents strong productivity in industrial production. For example, the Linux kernel has more than 400,000 contributors [11], while the total number of employees of Microsoft's multinational technology company is only approximately 110,000. In the world of open source, the philosophy of democracy attracts different types of public contributors to continuously contribute to OSS projects that interest them. The efficient reputation propagation effect motivates top universities and scientific research institutions to release the latest scientific research results to society in a timely manner. Further, unlimited potential innovation has motivated more and more software companies [12, 13] to achieve high-speed growth through the model of open-source development accompanied by a service payment.

Relative to the standardized and strongly organized engineering methods, the open-source method is more deferential to each developer's individuality and aims to create a liberal, diverse, and democratic environment [14], thus stimulating the enthusiasm and creative inspiration of contributors on a large scale and ultimately generating greater collective wisdom [15]. We call the OSS development process **Software Creation**, which is very similar to the process of creating artwork. Therefore, we refer to the software artifacts created during the creative process as the software opus, which is akin to works of art. On the one hand, in the context of ambiguous open issues (e.g., the requirement elicitation for innovative software), some unexpected solutions or so-called "killer applications" can be generated from the software creation process because of the inspirations of individuals or the swarm intelligence of crowd. On the other hand, challenges in the diversification of individual interest concerns, the unevenness of contribution capabilities, and the unpredictable results of group collaboration make this process unable to fully fulfill the tasks of clear and organized software production.

9.2.3 Crowd-Based Methodology

In today's Internet era, software has infiltrated all aspects of people's lives, and the wave of *Software-Defined Everything* (SDE) is essentially unstoppable. The unremitting progress made in the areas of engineering methods and open-source methodologies has made people see the daylight of fundamentally breaking through the bottleneck faced by software development, as discussed in Brooks's "The Mythical Man-Month" in 1975. When we rethink all kinds of software development activities, the concepts of "software manufacturing" and "software creation" are thought-provoking. If we compare software development to automotive manufacturing, the engineering methodology represented by the software production activities is similar to the assembly line for Ford or Toyota automobiles. This process splits the phase of atomization of software development activities (e.g., in the waterfall model, the life cycle of software development is divided into six basic activities: planning, demand analysis, software design, programming, software testing, and operation and maintenance). Through advanced production and management processes (e.g., the organizational model of the company), software development has grown from a small, personal programming workshop (PROG workshop) to a large group that can support hundreds or even thousands of large-scale collaborations (software bloc).

However, operating software in a virtual space is not identical to other labor-intensive industrial products, e.g., automobiles. When the technical barriers to the underlying infrastructure (e.g., storage or CPU speed) are broken, the large-scale mass production and transmission costs of mature software products are almost zero (e.g., copying CDs or hard disks). Consequently, in the field of software development, the advantages of scalable replication, the most significant improvement in

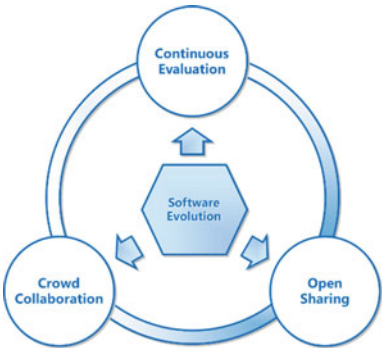


Fig. 9.1 The crowd-based methodology model

industrial production efficiency, have been greatly reduced. As the demand for new software features continues to grow, the bottlenecks faced by software developers are increasingly focused on software creation and the transition from creation to production.

At this point, the high degree of decomposition of the development process and the isolation in the collaboration of the development groups have not only failed to improve the efficiency of software creation but also limited the scope of software creation to a certain extent. In contrast, the open-source method, at the other end of the scale, can fully create the atmosphere and environment needed for software creation activities. For popular OSS, the development process seems to be similar to global car enthusiasts participating in the design of a certain concept car, which is an extremely effective strategy for innovation. However, open source has always been organized by loose mechanisms, which has led to a large number of metaphorical open-source concept cars unable to smoothly exit the software "Utopia" on time.

Building on the progress of previous innovators, we suggest that immediate remedies to software development barriers can be found in the above two types of processes, i.e., software manufacturing and software creation. Currently, developers are seeking an intermediate path between these two processes, e.g., the *Agile* development evolved gradually from the engineering method and the *DevOps* booming in the open-source world. From our perspective, the new software development methodology should integrate the software creation process into the software manufacturing process, and link a small-scale but well-organized core team with self-organized but large-scale crowd contributors.

When the goals are not clear, the "core" coordinates the "crowd" to achieve creative work efficiently, and after the goal is finalized, the "core" organizes the "crowd" to produce software products or transform a software opus to products in a timely fashion. We call this method the **crowd-based methodology** of software development. The essence of this crowd-based methodology consists of three essential and interconnected elements: crowd collaboration, open resource sharing, and continuous evaluation, as shown in Fig. 9.1.

9.3 Key Technologies of Crowd-Based Methodology

In this section, we present the key technologies of crowd-based methodology in TRUSTIE, including crowd collaboration, open resource sharing, and continuous evaluation.

9.3.1 Crowd Collaboration

Crowd collaboration in an open-source ecosystem is based on the onion structure [16]. For a project team, such a structure consists of a small but strongly organized core team and a large scale but unorganized peripheral contributors. To keep the onion structure productive, various aspects are involved, including collaboration between developers, the connection of developers and development tasks, and the management of the development process. Figure 9.2 shows that multiple potential collaboration approaches exist among developers, such as @-mention and follow in GitHub. The developers can submit pull requests (PRs) or commits for collaboration between developers and development tasks. All of these approaches together promote collaboration in the deployment process.

Collaboration between developers. Many studies have proposed that social media tools can promote collaboration among developers, which is beneficial to software development. We used a mixed method, i.e., combining qualitative and quantitative analysis, to provide an in-depth understanding of how @-mention is used in GitHub issues and its role in assisting software development. Our statistical results indicate that @-mention attracts more participants and tends to be used to address more challenging issues. @-mention favors the solution of issues by enlarging the visibility of issues and facilitating developer collaboration. Our study also builds an @-network,

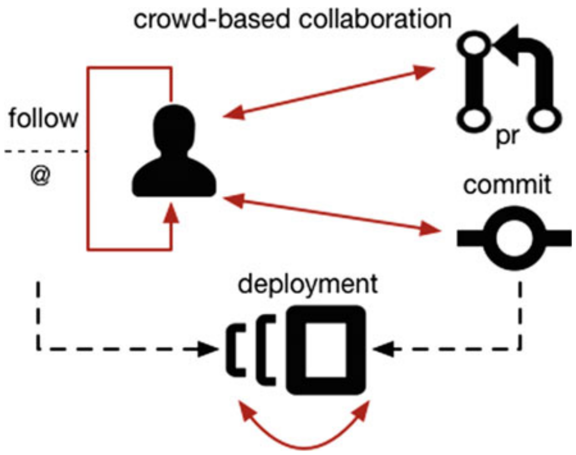


Fig. 9.2 Model of crowd collaboration in distributed software development

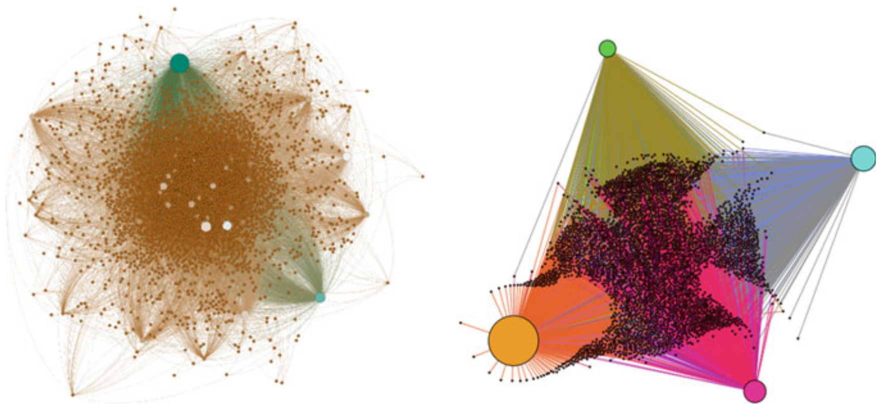


Fig. 9.3 @-network (left) and follow-network (right) in GitHub

as shown in Fig. 9.3 based on the @-mention database we extracted. Through the @-network, we investigate the evolution of the process over time and prove that we have the potential to mine the relationships and characteristics of developers by exploiting knowledge from the @-network.

The social coding paradigm has reshaped the distributed software development with surprising speed in recent years. GitHub, a remarkable social coding community, has attracted a huge number of developers in a short time. Various types of social network are formed based on social activities among developers. To determine why this new paradigm can achieve such great success in attracting external developers and how they are connected in such a massive community, we first compare the growth curves of projects and users in GitHub in three traditional OSS communities to explore the differences between their growth modes. We find explosive growth in the number of users in GitHub and introduce the diffusion of innovation theory to illustrate the intrinsic sociological basis of this phenomenon. Second, we construct follow-networks, as shown in Fig. 9.3, according to the follow behaviors among developers in GitHub. Finally, we present four typical social behavior patterns by mining follow-networks containing the independence pattern, group pattern, star pattern, and hub pattern. This study can provide several instructions for crowd collaboration to newcomers. Based on the typical behavior patterns, the community manager can design corresponding assistive tools for developers.

Connecting developers and tasks. The PR is the primary model for collaborative code contribution and aggregation between the core team and peripheral crowds in GitHub. To maintain the quality of software projects, PR review is an essential part of distributed software development. Assigning new PRs to appropriate reviewers makes the review process more effective, which can reduce the time between the submission of a PR and its actual review. However, the reviewer assignment is then organized manually in GitHub. To reduce this cost, we propose a reviewer recommender to predict highly relevant reviewers of incoming PRs. By combining information retrieval with social network analysis, our approach takes full advantage of the textual semantic of PRs and the social relations of developers. We implement an online system to show how the reviewer recommender helps project managers find potential reviewers from crowds. Our approach can reach a precision of 74% for top-1 recommendation and a recall of 71% for top-10 recommendations.

The continuous participation and contribution of the crowd are key factors for the success of open-source projects. However, given the massive number of competitors, it is difficult for a project to attract enough contributors by just passively waiting for enthusiasts to join in. Instead, the project should actively seek gifted developers. Most current studies have mainly focused on recommending experts inside a repository for some specific development tasks. To solve this problem, we propose the novel approach *ConRec* to recommend potential contributors across the entire open-source community for given projects. This approach leverages the developers' historical activities in projects to analyze their technical interests and technical connections with others. Thereafter, it combines a collaborative filtering algorithm with a text-matching algorithm to recommend proper developers. We conducted extensive experiments related to 5995 open-source projects and 2,938,620 developers in GitHub. The results show that the proposed algorithm can recommend contributors to open-source projects with the best performance of 63% in accuracy and solve the cold start problem as well.

Development process management. As an important approach in DevOps, continuous deployment aims to automate the delivery and deployment of a software product following any changes to its code. If properly implemented, continuous deployment, together with other automation steps implemented in the development process, can bring numerous benefits, including higher control and flexibility over release schedules, lower risks, fewer defects, and the easier onboarding of new developers. We conducted a mixed-method study to shed light on developers' experiences and expectations with continuous deployment workflows. Starting from a survey, we explore the motivations, specific workflows, needs, and barriers with continuous deployment. We find two prominent workflows based on the automated build features on Docker Hub or continuous integration services, with different trade-offs.

9.3.2 Open Resource Sharing

OSS community ecosystems (OCEs) can be seen as a complex network of resources from around the open-source community, including related open source projects, open source products, open source organizations, open-source developers, and users. OCEs have accumulated massive resources, and new resources are constantly being generated. These resources come in a variety of forms, including software artifacts (e.g., code snippets, and libraries), development documents (e.g., bug reports, design specifications), and behavioral data (e.g., review discussions, social communications). Open-source developers try to share these resources with the whole ecosystem as much as possible to obtain feedback (including criticism) and increase the

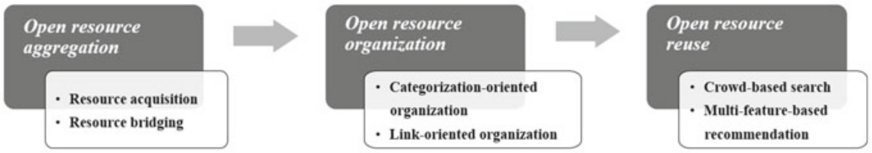

Open resource aggregation
• Resource acquisition
• Resource bridging
Open resource organization
• Categorization-oriented organization
• Link-oriented organization
Open resource reuse
• Crowd-based search
• Multi-feature-based recommendation

Fig. 9.4 Open resource-sharing pipeline

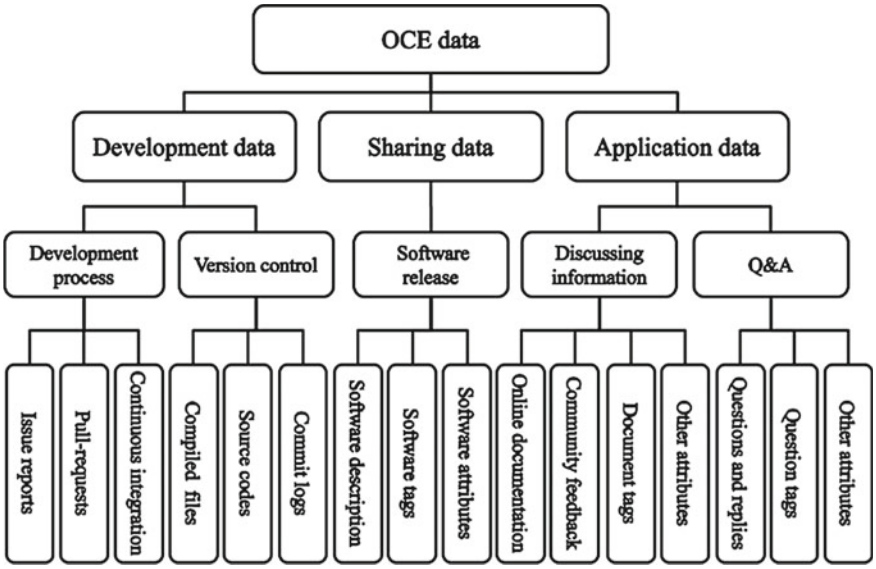


Fig. 9.5 Structure and types of collected OCE data

reuse rate of these resources. As shown in Fig. 9.4, to facilitate this process and fully explore the value of these resources, we propose an open resource-sharing pipeline consisting of three steps: open resource aggregation, open resource organization, and open resource reuse.

Open resource aggregation. As shown in Fig. 9.5, diverse resources such as software artifacts, historical process data, issue reports, and feature requests are produced and distributed dispersedly over various communities with the development of OCEs. To automatically and continuously aggregate such massive and diverse resources, we designed an aggregation system of high robustness, efficiency, and flexibility. The aggregation system consists of two processes: resource acquisition and resource bridging.

Resource acquisition. We use official APIs provided by open-source communities and web crawlers to obtain raw resources. Data crawling usually contains two interdependent processes, that is, crawling the raw web pages and extracting their attributes. However, direct extraction after crawling is not a suitable choice for rapidly changing and growing OCEs. To make the aggregation system acquire high quality and complete data, the system is designed with three stages: raw resource crawling, structured information extraction, and final data verification. These three stages are connected by the data flow and decoupled from the message queue and database. Their working state and interaction record are stored in case of exception or errors. Under this design, we can simply improve and restart any individual broken module without affecting and restarting other running modules.

Moreover, due to the various types of resources existing in OCEs, we make the aggregation system dynamically modifiable with external configuration files, which define the rules to generate initial page links, extraction templates, and verification specifications, among other possibilities. This approach provides a plugin-based feature to adapt to the resource diversity, and the only effort required to include a new type of resource is simply to write a new configure file; the source code remains unchanged. In summary, we have collected a broad variety of OCE resources of a wide range of types. The collected data currently cover nearly 20 well-known open-source sites, containing more than 14.2 million projects and 14.62 million posts.

Resource bridging. In spite of the diversity of OCE resources, open resources mainly exist in two types of communities: collaborative development communities (e.g., GitHub, Oschina) and knowledge-sharing communities (e.g., Stack Overflow, CSDN). The former resource type contains structured software artifacts, while the latter mostly contains textual posts. These two types of communities complement each other, and bridging them can expand the application value of OCE resources. While the bridging can be seen as a classification of posts to software projects, a conventional supervised ML-based classification algorithm cannot be applied due to the lack of training sets. Therefore, we use a text-matching method to solve this problem. We first extract the common attributes from different sources and define a uniform structure for each type of community. Two sets of resources are then integrated: posts and projects. Given a post, its title, tags, and content are matched with the name of a project, and each type of match is assigned a separate score. Finally, the match score between a post and a project is taken as the sum of these separate match scores with different weights.

Open resource organization. It is important to understand the collected resources before we can truly use them. An appropriate model and organization of open resources can lead to efficient and effective application.

Categorization-oriented organization. Categorization is considered to be an efficient way to manage information from large-scale data repositories. This approach clusters resources according to their topics and is quite useful for browsing and retrieving resources with similar functions. We propose a hierarchical repository of software features, which is an ideal technique to categorize software resources to support resource organization with flexible granularity. First, we extract a massive number of feature descriptions from online software profiles and mine their hidden semantic structure by a probabilistic topic model. Then, we present an improved agglomerative hierarchical clustering algorithm, seamlessly integrated with the topic model, to build the feature ontology.

Tagging is another popular and powerful mechanism for categorizing resources. To uncover the hidden semantics among tags, we attempt to induce an ontologylike taxonomy from tags. Specifically, we propose an agglomerative hierarchical clustering framework that relies only on how similar any two tags are. We enhance our framework by integrating it with a topic model to capture thematic correlations among tags. However, a severe problem for the current tagging systems in OCE is tag insufficiency. Consequently, we propose tag recommendation based on a semantic graph (TRG), a novel approach to discover and enrich tags of OSS. First, we propose a semantic graph to model the semantic correlations between tags and the words in software descriptions. Then, based on this graph, we design an effective algorithm to recommend tags for software.

Link-oriented organization. Although the two types of open-source communities emphasize different aspects of OCEs, they are highly correlated and mutually complementary because they overlap with each other by containing shared participants and issues. To mine the potential value in the two types of communities, it is necessary to reveal the associations between them and link them for knowledge sharing. For example, to explore hidden links between Android Issue Tracker and Stack Overflow, we focus on two factors: text similarity and temporal correlation. Intuitively, two related threads in different communities are more likely to have similar descriptions and discussion texts and arise in the same short period of time, which can be seen as a type of temporal locality. Based on this intuitive result, we propose an approach that combines semantic similarity with the temporal locality to link correlated threads across communities.

Moreover, social coding facilitates the sharing of ideas within and between projects in an OCE. Bug fixing and triaging, in particular, are aided by linking issues in one project to potentially related issues within it or in other projects in the ecosystem. We present a mixed-method study of the relationship between the practice of issue linking and issue resolution in the Rails OCE. Using a qualitative study of issue linking, we identify a discrete set of linking outcomes together with their coarse-grained effects on issue resolution. We use these findings to guide our quantitative modeling study of patterns in developer linking within and across projects, from a large-scale dataset of issues in Rails and its satellite projects. We find that Rails OCE developers tend to contribute most of their work within the ecosystem but that the distribution of the work across projects varies. Furthermore, using models of issue resolution latency, when controlled for various attributes, we find no evidence that linking across projects retards issue resolution.

Open resource reuse. Open-source resources are generated by the crowd; in turn, they serve the crowd and link the ecosystem. The most common way to reuse shared resources is by searching or recommendation.

Crowd-based search. Global open-source resources have become an Internet-scale repository that provides abundant resources for software reuse. However, how to locate the desired resource efficiently and accurately from such a large amount is a challenging problem. To solve this problem, we propose a prototype search engine that leverages the crowd wisdom to optimize the search result ranking. The number of times a software project was discussed by the crowd in various communities reflects its influence, and we treat the crowd discussions as an important ranking factor. For a user query that is formulated to find reusable software resources, we consider the semantic similarities between the query, the indexed resources and the crowd discussion popularity of the resources, and we compute a combined ranking score. Finally, we return the resources that obtain the highest combined ranking score.

Multifeature-based recommendation. Due to the transparency and openness of OCEs, a large number of external contributors are attracted to open-source development. The massive numbers of developers are driven by an interest in participating in specific development tasks. They have different personality traits, educational backgrounds, and expertise levels. Therefore, a personalized recommendation service may be helpful to reduce developers' time and effort in reusing proper and interesting projects. Therefore, we also propose an active recommendation approach to recommend resources for developers based on multidimensional features. We model the potential correlations between developers and open-source projects from three different dimensions: the popularity of projects, technical dependency among projects, and social association among developers. We aggregate the three dimensions of features with a linear combination and a learning-to-rank approach. Subsequently, the aggregated score is used to rank and recommend the top-K candidates.

9.3.3 Continuous Evaluation

The trustworthy software has attracted public attention in the area of software quality. Among the classic automation methods and engineering methods, software quality assurance is mainly achieved through formal verification and software testing. These methods have high costs and are mainly used for objective quality analysis. However, these methods ignore the subjective evaluation of contributors in crowd-based development activities, which presents challenges in adapting these methods to the continuous evaluation of software with changing requirements.

In an open-source ecosystem, a large amount of process data is produced through software development, which presents a large scale, diverse types, rapid growth, and rich content of big data. There are rich subjective feedbacks such as user requirements and evaluations. The process data, which form a complete chain of evidence from the requirement specification to the software code, constitute a new and important source of evidence for the analysis of software trustworthiness. Facing the new changes of an open-source ecosystem, we conduct evaluation works for projects, development tasks, developers, and issues.

Evaluation of resources. The amount of software in the open-source ecosystem is increasing more and more rapidly. Such a huge amount of OSS makes the rapid evaluation of software a necessary skill for developers. However, conventional methods have high costs and sometimes conflict with developer experience. We present a method to evaluate projects based on crowd feedback. To achieve this goal, we first combine all software information from different communities and then bridge them with posts from StackOverflow, which provides feedback regarding the software. In the process of connecting software production communities, we filter the duplicative projects, build a list of software and integrate all of their information. Then, we bridge software with posts from StackOverflow, and we link feedback with software by keywords and other descriptions. Finally, we evaluate the popularity of software by the number of linked posts, view count, and up-vote scores of these posts.

Evaluation of project. The integration and automation of the software development process have been key concerns in software engineering. We use large, historical data on process metrics and outcomes of GitHub projects to discern the effects of one specific innovation in process automation: continuous integration. We explore the impact of CI on software quality and the productivity of teams. We gather research metrics from three dimensions that are known to affect the rate of growth of projects' source base and the quality thereof: (a) the project attribute dimension (e.g., the project age, the project size, and whether the project uses CI), (b) the project popularity dimension (e.g., the number of forks and stars), and (c) the project development activity dimension (e.g., the numbers of opened issues and PRs and the numbers of merged and rejected PRs). By controlling for several known factors that affect the productivity and quality, we aim to discern the effects of CI. Then, we use multiple regression modeling to describe the relationship between a set of explanatory variables (predictors, e.g., usage of CI) and a response (outcome, e.g., number of bugs reported per unit time). Our findings clearly show the benefits of CI: more PRs get processed. Moreover, this increased productivity does not appear to be gained at the expense of quality.

Moreover, the open-source ecosystem presents extreme openness for developers to contribute, such as reporting issues. The extreme openness poses a severe challenge for the core team in project maintenance. Illustrated by the case of the issue tracker system (ITS), in large-scale projects, many undesirable and vague issue reports are submitted by external contributors (e.g., asking questions) because of their reluctance to spend adequate time to read and comprehend the contribution guidelines, which provide details on reporting an issue in a high-quality way and the type of issue that the project prefers to address. Thus, issue evaluation is a labor-intensive and time-consuming task for project managers. Furthermore, the core team members have to provide rapid responses and resolve the incoming issues in time to sustain the passion of external contributors. To help managers quickly evaluate whether the issue reports are a bug or not, we present a two-stage classifier framework to combine textual summary information and developer information that uses automatic classification techniques. The first stage extracts the probability of bug-prone and perplexity information of sentences for each issue from the free text, and in the second stage, some structured features about contributors who submit issue reports are provided, which can be expected to improve the performance of classification.

Evaluation of developers. Currently, more developers are adopting collaborative development models (e.g., pull-based model) in OCEs. The openness and convenience of such collaborative models reduce the contribution entries and promote developer enthusiasm. However, in a large OCE, the high volume of incoming contributions poses a severe challenge to project integrators who must review the contributions' quality. We first explore which factors affect the contribution evaluation latency in GitHub. We extract four indicators from the perspective of personal relations, namely, the submitter's success rate, whether the submitter is an integrator, the strength of social connection and the total number of GitHub developers following the submitter. Using regression modeling on sampled data, we find that these factors, including the submitter's track record, reputation, and social connection with project members, are highly significant. Contributions submitted by the core team members and contributors with more followers, more ties to project integrators, and higher previous PR success rates are associated with shorter evaluation latencies. In other words, open-source projects prefer a useful contribution from a well known and trusted contributor.

Furthermore, we aim to recommend appropriate reviewers to reduce the time between the submission of a contribution and its actual review. The two key concepts of our approach focus on the textual semantic of contributions and the social relations of contributors.

- The expertise of a reviewer can be learned from the reviewer's PR-commenting history. For a newly received PR, the developers who have commented on similar PRs frequently in the past are suitable candidates to review the new one.
- Common interests among developers can be measured by social relations between contributors and reviewers in historical PRs. Developers who share more common interests with the contributor are appropriate reviewers of that contributor's incoming PRs.

As a result, we first propose a novel approach to construct comment networks by mining historical comment traces. Based on the comment network and information retrieval technologies, we predict highly relevant reviewers for incoming PRs.

9.4 TRUSTIE Environment

Based on the crowd methodology and the key technologies, we designed and implemented TRUSTIE (**Trust**worthy software tools and **I**ntegration **E**nvironment) to support the modeling and construction of an open-source ecosystem. In this section, we give a brief instruction of the TRUSTIE architecture, and then present the typical support for ecosystem construction.

9.4.1 TRUSTIE Architecture

The core goal of TRUSTIE is to help form an open-source community ecosystem that connects diverse stakeholders to collaborate together in a community for continuous innovation and benefit. To this end, we built the TRUSTIE platform, which is composed of three levels: the data management infrastructure, the key technologies and mechanisms, and subsystems and services. The detailed architecture is shown in Fig. 9.6.

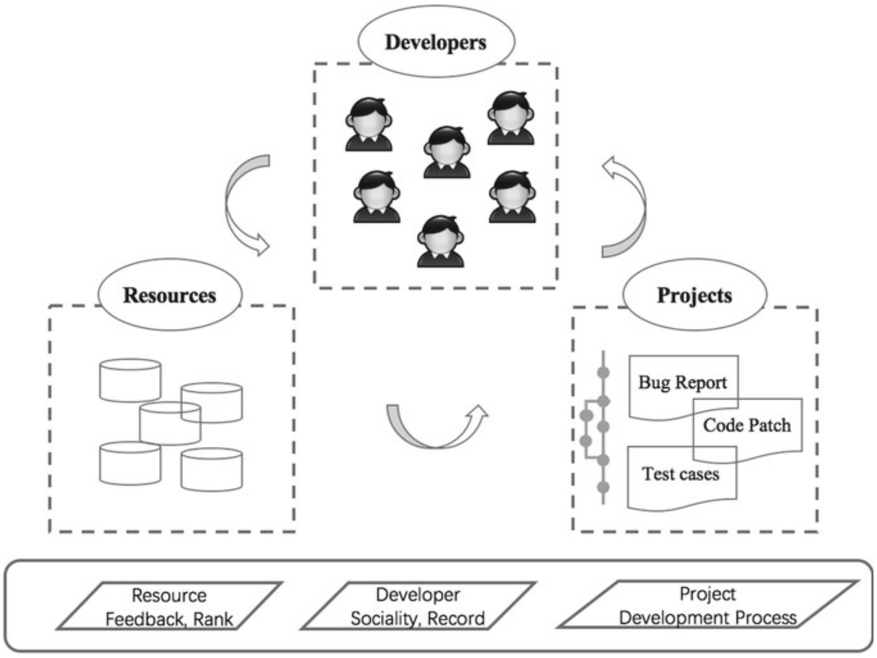


Fig. 9.6 Continuous evaluation of developers, projects and resources

Data management infrastructure: The construction and evolution of an open-source ecosystem is a data-driven process that also generates rich data. The data management infrastructure is in charge of data storage and providing a data access interface for upper levels. From the view of the source, there are mainly two types of data: the first is the data generated in the TRUSTIE community, such as project development data and user feedback, which are critical for guiding the construction and evolution of the OSS community ecosystem, and the second is the data collected from the Internet, such as the open-source application community and development community data, which provide reusable resources and empirical guidance.

Key technologies. The crowd-based methodology is the core and essence of TRUSTIE, which is supported by three groups of key technologies. *Crowd collaboration technologies* help extend the emphasis from only "professional-developer-centered" to "diverse-crowd-driven" and connect the small core team with large peripheral crowds for effective collaboration. *Open resource-sharing technologies* help transfer the "fragmented and disorderly" raw resources to "aggregated and ordered" ones and promote the effectiveness of resource sharing in and among teams. *Continuous evaluation technologies* transfer the traditional "static and single-dimension" analysis to the "dynamic and multidimensional" measure and evaluate the entities in the ecosystem continuously.

Systems and services. Driven by the key technologies, TRUSTIE was used to design and implement three subsystems that focus on various aspects of OCE construction and evolution. The *crowd-based learning platform* focuses on the professional development of developers in the ecosystem. This platform provides channels to introduce the incoming crowds and resources in the OSS community into a traditional classroom and to connect curriculum learning with standard project practices to help individuals develop their skills and prompt them to engage in OCE. The *open resource-sharing platform* collects and introduces Internet-scale external resources to the enclosed organizations and provides various channels such as resource retrieval and recommendation for effective resource sharing in and among teams. The *crowd-based collaborative development system* designs and embeds various mechanisms and services such as a development forum, process management, and code evaluation to connect the core team and peripheral crowds for software development.

9.4.2 Typical Support for Ecosystem Construction

The key factor for the construction and evolution of the OSS community ecosystem is "connection". The essence of crowd methodology is also "connection". This methodology emphasizes three types of connections and transformations: (1) connecting the peripheral crowds with the core team; (2) connecting crowd creation with business production activities; and (3) transforming the opuses created by crowds to the products managed by the core team. Figure 9.7 presents typical examples of the TRUSTIE support for such connections.

Connection between the core team and peripheral crowds: TRUSTIE incorporates various channels for connecting the core team and peripheral crowds. For example,

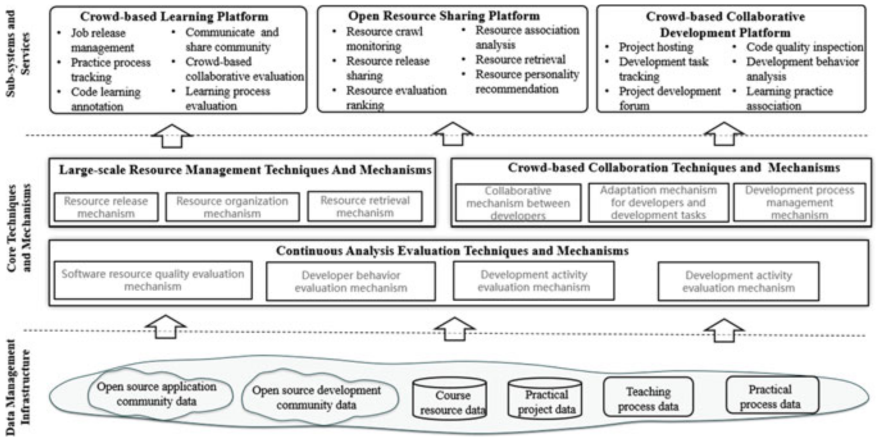

Sub-systems and Services
Crowd-based Learning Platform
Job release management
Practice process tracking
Code learning annotation
Communicate and share community
Crowd-based collaborative evaluation
Learning process evaluation
Open Resource Sharing Platform
Resource crawl monitoring
Resource release sharing
Resource evaluation ranking
Resource association analysis
Resource retrieval
Resource personality recommendation
Crowd-based Collaborative Development Platform
Project hosting
Development task tracking
Project development forum
Code quality inspection
Development behavior analysis
Learning practice association
Core Techniques and Mechanisms
Large-scale Resource Management Techniques And Mechanisms
Resource release mechanism
Resource organization mechanism
Resource retrieval mechanism
Crowd-based Collaboration Techniques and Mechanisms
Collaborative mechanism between developers
Adaptation mechanism for developers and development tasks
Development process management mechanism
Continuous Analysis Evaluation Techniques and Mechanisms
Software resource quality evaluation mechanism
Developer behavior evaluation mechanism
Development activity evaluation mechanism
Development activity evaluation mechanism
Data Management Infrastructure
Open source application community data
Open source development community data
Course resource data
Practical project data
Teaching process data
Practical process data

Fig. 9.7 TRUSTIE architecture

the discussion module is embedded in all three subsystems, which provides a convenient way for participants to communicate and form a micro-community. The task assignment mechanism connects them through tasks, and the resource-sharing mechanism connects them through resources.

Connection and transformation between creation and production: TRUSTIE opens both the project source code and the development process to the core team and peripheral crowds, providing corresponding mechanisms to connect the crowds' creation with business production. The crowds can express their requirements or comments (*innovation*) freely in TRUSTIE, and the core team can then be inspired to arrange corresponding tasks in the development plan (*task*). The crowds with necessary skills can also realize the innovations into source code (*innovation realization*) and submit the results to the core team, and the core team can merge the crowds' contributions into the product after reviews, or they can assign the task to the proper developer to implement (*task implementation*). The crowds can obtain and experience the product and share their feedback (*use and feedback*), and a large amount of feedback provides valuable evidence to rank and recommend reusable resources for software production.

Connection and transformation between opus and product: The outcome of crowd creation can be viewed as a type of *opus* that is inspiration-driven, and the outcome of business production is a type of *product* that is market-requirement-oriented. In the process of the connection and transformation between crowd creation and business production, opuses such as crowd innovation, code snippets, and feedback are connected and transformed into corresponding products such as development tasks, product code and reusable resources.

9.5 Application Scenarios

There have been many successful applications and practices based on crowd-based methodology, demonstrating the effectiveness of this approach. We briefly introduce two typical cases: practices in software companies and practices in online communities.

9.5.1 Practices in Software Companies

Neusoft is one of the leading IT solution and service providers in China. This company faces many challenges in increasing productivity due to its large volume of employees such as the reuse of company assets, cross-team collaboration, shortening of the development cycle, reduction of costs, and reduction of defect rates.

To consolidate the Neusoft production platform, we provide a new software development environment named TRUSTIE CDE that is based on the crowd-based methodology. This platform takes advantage of the mechanisms of the crowd method:

Table 9.1 Practical examination of the enhanced platform

	Exp. scale	Exp. domain	Reuse rate	Collaboration efficiency	Rate of defect reduce
Exp. 1	• 300 persons • 507 man months • 8 projects	Application software A: Health insurance B: Health information	↑70%	↑65%	↓31.5% for A ↓35.4% for B
Exp. 2	• 100 persons • 6 projects	Application software Tax	↑20%	↑45.69%	↓20%
Exp. 3	• 400 persons • 60 months	Application software Navigation	↑121 components in 11 categories	↑63.64%	↓18.7%
Exp. 4	• 261 persons • 6 projects • 12 months	Application software E-Government	326 software resources	↑41% for design ↑24.5% for coding	↓17.2% for requirement ↓17.8% for design ↓16.9% for coding
Exp. 5	• Millions of LOC in projects • 36 months	Infrastructure software Cloud computing	↑20%	↑30%	↓25%

the large-scale sharing of assets, cross-team collaboration, flexible production lines, and user feedback tracking. These mechanisms integrate collective wisdom to help the core teams in Neusoft make effective decisions. Several experiments have been conducted on more than 20 large software projects to examine the effect of the new platform. As shown in Table 9.1, we find that the crowd methodology can significantly improve the reuse rate, collaboration efficiency, and software quality in these projects.

9.5.2 Practices in Online Communities

Based on the crowd methodology, TRUSTIE fosters prosperous online communities centered around open sharing and collaborative development, as shown in Fig. 9.8. This framework has become a well-known software development and innovation ecosystem in China.

Currently, there are more than 3,900,000 projects and 14,200,000 posts in resource-sharing services, as shown in Fig. 9.9a. The data are collected from the most popular open-source communities and knowledge-sharing communities all over

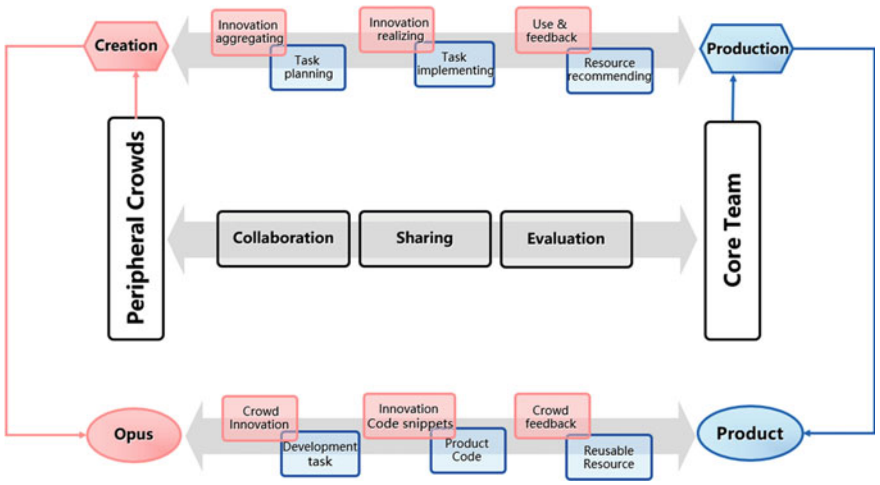


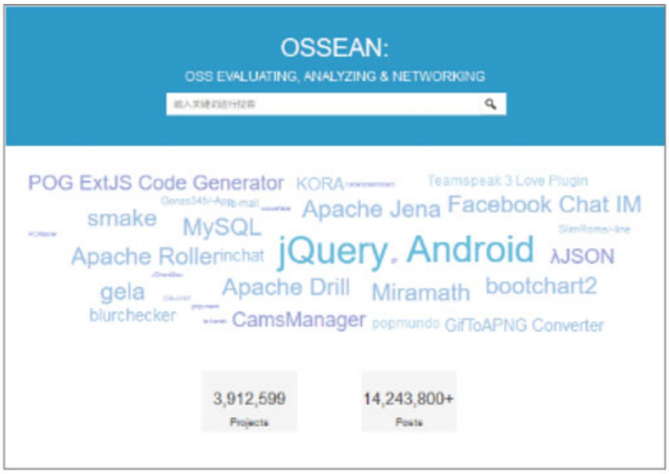
Fig. 9.8 Three types of connections and transformation in TRUSTIE

the world. TRUSTIE analyses and connects the large-scale data entities in different communities and then provides searching, evaluation and ranking services for OSS. Also, there are more than 52,000 users, 6800 repositories, and 2100 online software-engineering-related classes hosted in TRUSTIE. The typical user interface of a code repository is shown in Fig. 9.9(b).

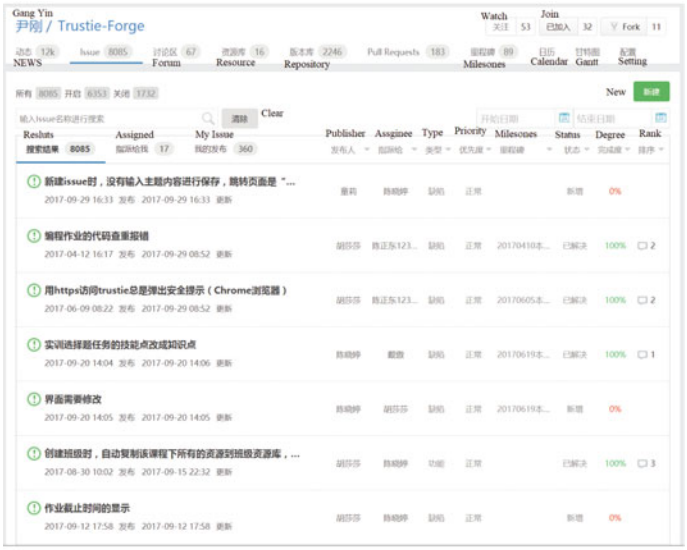
9.6 Conclusion

In the Internet era, our daily lives have been redefined by software-driven technologies. Driven by massive decentralized crowds, OSS has achieved unprecedented success without strict centralized control. We study the core mechanisms behind the rapid development of OSS comprehensively and compare its development patterns with those of traditional software engineering approaches. We propose a crowd-based methodology to bridge the two paradigms of engineering and crowd wisdom methods, which enables crowd-oriented collaboration among internal development teams and external crowds by combining software innovation and software manufacturing.

The crowd-based methodology consists of three important components: crowd collaboration, open resource sharing, and continuous evaluation. Based on the crowd-based methodology, we built the TRUSTIE environment, which embeds multiple technologies and mechanisms to support the modeling and construction of the OSS community ecosystem. Over nearly ten years of evolution, TRUSTIE has enabled the formation of three typical and interconnected communities for crowd learning, open sharing, and collaborative development. The practices in software companies and



(a) Open sharing community


Gang Yin
尹刚 / Trustie-Forge

Watch 关注 53 · Join 已加入 32 · Fork 11

动态 12k NEWS · Issue 8085 · 讨论区 67 Forum · 资源库 16 Resource · 版本库 2246 Repository · Pull Requests 183 · 里程碑 89 Milesones · 日历 Calendar · 甘特图 Gantt · 配置 Setting

所有 8085 开启 6353 关闭 1732 · New 新建

输入Issue名称进行搜索 · 清除 Clear · 开始日期 · 结束日期

Resluts 搜索结果 8085 · Assigned 指派给我 17 · My Issue 我的发布 360

	Publisher 发布人	Assginee 指派给	Type 类型	Priority 优先度	Milesones 里程碑	Status 状态	Degree 完成度	Rank 排序
新建issue时，没有输入主题内容进行保存，跳转页面是 "… 2017-09-29 16:33 发布 2017-09-29 16:33 更新	董莉	陈晓婷	缺陷	正常		新增	0%	
编程作业的代码查重报错 2017-04-12 16:17 发布 2017-09-29 08:52 更新	胡莎莎	陈正东123...	缺陷	正常	20170410本...	已解决	100%	2
用https访问trustie总是弹出安全提示（Chrome浏览器） 2017-06-09 08:22 发布 2017-09-29 08:52 更新	胡莎莎	陈正东123...	缺陷	正常	20170605本...	已解决	100%	2
实训选择题任务的技能点改成知识点 2017-09-20 14:04 发布 2017-09-20 14:06 更新	陈晓婷	[illegible]	缺陷	正常	20170619本...	已解决	100%	1
界面需要修改 2017-09-20 14:05 发布 2017-09-20 14:05 更新	陈晓婷	胡莎莎	缺陷	正常	20170619本...	新增	0%	
创建班级时，自动复制该课程下所有的资源到班级资源库，… 2017-08-30 10:02 发布 2017-09-15 22:32 更新	胡莎莎	陈晓婷	功能	正常		已解决	100%	3
作业截止时间的显示 2017-09-12 17:58 发布 2017-09-12 17:58 更新	胡莎莎	陈晓婷	缺陷	正常		新增	0%	

(b) Collaborative development community

Fig. 9.9 The online communities in TRUSTIE. **a** Open sharing community. **b** Collaborative development community

communities show that the crowd-based methodology and the TRUSTIE environment can strongly support ecosystem modeling and construction and bring substantial benefits to practical research institutions and business enterprises.

References

1. G.E. Moore, Cramming more components onto integrated circuits. IEEE Solid-State Circuits Soc. Newsl. **20.3**, 33–35 (2006). Reprinted from Electronics, vol. 38, no 8, pp. 114 ff, 19 Apr 1965
2. G.E. Moore, Cramming more components onto integrated circuits. Proc. IEEE **86**(1), 82–85 (1998)
3. P. Naur, R. Brian (eds.), Software engineering: Report on a conference sponsored by the NATO SCIENCE COMMITTEE, Garmisch, Germany, 7th to 11th October 1968. Nato (1969)
4. R.S. Pressman, *Software Engineering: A Practitioner's Approach* (Palgrave Macmillan, London, 2005)
5. W.W. Royce, Managing the development of large software systems: concepts and techniques, in *Proceedings of the 9th International Conference on Software Engineering* (IEEE Computer Society Press, 1987)
6. B.W. Boehm, *Software Engineering Economics*, vol. 197 (Prentice-Hall, Englewood Cliffs (NJ), 1981)
7. R. Niekamp, *Software Component Architecture* (Gestión de Congresos-CIMNE/Institute for Scientific Computing, TU Braunschweig, 2005)
8. P. Clements, N. Linda, *Software Product Lines* (Addison-Wesley, Boston, 2002)
9. F. Brooks, H.J. Kugler, *No Silver Bullet* (1987)
10. B.D. Software, N. Bridge, *Future of Open Source Survey Results* (2015)
11. M. Zhou, Q. Chen, A. Mockus, F. Wu, On the scalability of Linux kernel maintainers' work, in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering and (ESEC/FSE 2017)* (ACM, New York, NY, USA, 2017), pp. 27–37
12. M. Zhou, A. Mockus, X. Ma, L. Zhang, H. Mei, Inflow and retention in OSS communities with commercial involvement: a case study of three hybrid projects. ACM Trans. Softw. Eng. Methodol. (TOSEM) **25**(2), 13 (2016)
13. E. Kalliamvakou, D. Damian, K. Blincoe et al., Open source-style collaborative development practices in commercial projects using GitHub, in *ICSE* (2015), pp. 574–585
14. D. Rushkoff, *Open Source Democracy: How Online Communication is Changing Offline Politics*, vol. 10753 (Demos, 2003)
15. J. Surowiecki, The wisdom of crowds: why the many are smarter than the few and how collective wisdom shapes business. *Economies, Societies and Nations* **296** (2004)
16. Y.W. Ye, K. Kishida, Toward an understanding of the motivation of open source software developers, in *Proceedings of 25th International Conference on Software Engineering* (2003), pp. 419–429