CHAPTER

Toward Collaborative Software Engineering Leveraging the Crowd

8

Benjamin Satzger[1], Rostyslav Zabolotnyi[1], Schahram Dustdar[1], Stefan Wild[2], Martin Gaedke[2], Steffen Göbel[3], and Tobias Nestler[3]

[1]*Vienna University of Technology, Vienna, Austria*

[2]*Technische Universität Chemnitz, Chemnitz, Germany*

[3]*SAP AG, Dresden, Germany*

8.1 Introduction

Short, unpredictable business cycles and fluctuations, rapidly emerging technologies and trends, globalization, and the global interconnectedness provided by the Internet have increased the world's economical clock speed and competition among companies. IT has always been a spearhead industry and is influenced by changing circumstances sooner than other industries. Competition in software engineering is high because of low market-entry barriers. IT companies, start-ups, and freelancers are competing against each other for market share. The professional development of software requires technology expertise and knowledge about the application domain, and must be inexpensive and agile to bring the company in a strong position. As technologies evolve constantly and new trends appear in very short time frames, the knowledge and competence of development teams have to be kept up to date accordingly. Time-consuming staffing periods causing long development cycles are not acceptable. However, many applications must follow strict guidelines to ensure both security and high quality.

In the event specific competences are not available within a team, external knowledge has to be included. The hiring of new employees with the right skill set is the traditional instrument to enhance the development unit's competence and capacity. However, it is too slow and inflexible for modern rapidly developed projects, as traditional hiring binds additional resources for assessing the job applicants. Not only Human Resources personnel are involved, but also domain experts to test the technical know-how of job applications. All of this is a rather time-consuming process. Recent developments in IT, like the rise of social networks, Cloud computing, global software development, and the emergence of crowdsourcing services, promise to help companies to cope with the new requirements they are facing. Social networks can be leveraged to support people in loosely coupled and open team structures to efficiently collaborate; Web-based crowdsourcing allows outsourcing tasks by broadcasting to a large network of people, the crowd, via an open call.

The IT industry giants also are considering this trend; for example, Microsoft outsources key parts of its IT operations for cost minimization and business simplification (Computerworld, 2012).

Economics-Driven Software Architecture. DOI: http://dx.doi.org/10.1016/B978-0-12-410464-8.00008-8

According to an internal strategy document that leaked out in early 2012, IBM plans to employ a radically new business model (Spiegel, 2012). It involves letting the company run by a small number of core workers. A dedicated Web-based platform is used to attract specialists and to create a virtual "cloud" of contributors. Similar to Cloud computing, where computing power is provided on demand, IBM's people Cloud would allow leveraging a flexible, scalable workforce.

However, usage of external workers entails new challenges compared to classical in-house software development. The distributed, dynamic nature of the crowd has to be considered in the whole software development process. As crowd-conducted and agile software projects share some common characteristics such as focusing on individuals (Beck, 2001), we argue that success factors in agile software projects (Chow and Cao, 2008)—namely, project, organizational, people, process, and technical factors—are also significant in crowd-conducted ones. Solving the following questions is a key to enabling professional software development conducted by a flexible workforce, as described above:

1. How to manage and organize software projects that are conducted, at least in part, by the crowd?

   A new management methodology must be developed with support to leverage the collective intelligence of the crowd and, at the same time, assure quality. It allows monitoring and guiding agile crowd development teams. Involved crowd members are able to discuss the goal of their task with the team and are integrated in planning the current development tact. Ongoing development activities are observable, which allows product owners to control and guide the distributed, remote software development process.
2. How to create virtual teams consisting of suitable experts and delegate tasks (semi-) automatically?

   Building and managing a platform that helps companies maintain a flexible workforce is of paramount importance for leveraging the crowd in software development. Worker profiles based on their interests and observed performance/behavior in past projects are important for making informed decisions regarding quality assurance and team creation. The platform must enable and assist project managers to map tasks to suitable developers or virtual teams, which may be established on demand. Reliable workers providing good quality need to be fairly rewarded, to feel gratified and motivated (Ramlall, 2004; Bruce et al., 1999). It is important not only to find suitable workers/teams for tasks and to provide the customer with satisfying quality, but also to maintain a motivated base of IT experts and provide stimulus for learning required, but underrepresented, skills. Only a recurring, satisfied crowd staff is able to ensure high long-term quality and to avoid thebrain drain of high potential crowd developers or affected in-house software developers.
3. How to improve cooperation of developers within loosely coupled teams?

   Distributed development is already a reality in several industries. Within large IT companies, today development teams are often distributed (DiMicco et al., 2008; Damian et al., 2007; Sengupta et al., 2006; Rodriguez et al., 2010). Management can be located in one country while the main development workforce is spread over several locations. Communication barriers, cultural differences, and different time zones hinder efficient collaboration and teamwork. Furthermore, keeping track and understanding what developers have produced becomes complicated if they are located in different offices. Team members have to take over code written by others or start a new project by reusing existing artifacts from other developers. Problems grow in projects that forward tasks to external developers and freelancers, especially if they do not know each other.

Today, crowdsourcing platforms and Internet-based tools for collaborative working are available, and some popular software is being developed based only on a remote workforce. In this work we investigate current solutions, their limitations, and point to new developments that help to realize crowd-enabled software engineering.

8.2 State of the art

Software engineering involves quite a number of people in various roles (Carmel and Agarwal, 2001). Many circumstances influence the way developers create the components of the software. If the whole team is at the same location, the collaboration is not very difficult to organize (Noll et al., 2010). Nonetheless, more and more software development teams are distributed all over the world. In particular, open-source communities are loosely coupled and need a platform to organize their workflows and the entire development process (da Silva et al., 2010; Koch, 2009; Robles and Gonzalez-Barahona, 2006).

A couple of platforms are established and used by IT experts for global software development (Lanubile et al., 2010; Dabbish et al., 2012). On the one hand, several cloud platforms are focused on open-source software, for instance, SourceForge (SourceForge, 2012) or BerliOS (BerliOS, 2012). On the other hand, platforms such as GitHub (GitHub, 2012) or Assembla (The Next Generation, 2012) support private projects as well (Walsh, 2009). SourceForge provides tools such as version control systems for source code, a bug tracker, a wiki system, forums, and an infrastructure for downloads. BerliOS uses a similar approach but also takes also different interest groups in the area of open-source software into account. Their aim is to fulfill a neutral mediator function. GitHub has a more social-driven approach with activity streams and additional visualization features to show the activities of the community. Assembla goes even further and integrates project management tools, agile development techniques, and scrum boards (Assembla, Kanban/Scrum Task Board, 2012).

In addition to platforms that focus on software development, a couple of generic systems simplify collaboration and are used in distributed software development. Social networks play an important role for Internet society and are actively used for advertising and business activities (Bosari, 2012). Even though social networks usually are not designed for distributed software development, they still are actively used for communication (Madey et al., 2002). Web-based office suites such as Google Docs (Google, 2012), the Zoho office suite (Zoho, 2012), Adobe Buzzword (Adobe, 2012) or Microsoft Office 365 (Microsoft, Microsoft Office, 2012) are replacing more and more classical document flows and provide concurrent, distributed editing. Another interesting example is Apache Wave (formerly Google Wave) (Apache, Apache Wave, 2012): In 2009 it was announced as novel social collaboration portal, but it did not gain much popularity. However, the wave idea received positive feedback in business and development environments, and is being used (Fidelman, 2010) even though Google discontinued the original project.

One of the most famous software development projects, built by a community, is Linux. Instead of assigning particular people to do different parts of the software development, anyone who wants to add or change any part of the software is able to do so (Malone et al., 2010). It is common that the crowd not only develops, but also decides about new features to add and when they are ready for integration. Sometimes, however, a project coordinator or a team of core developers makes this decision (Leo Kelion, 2012).

Today, some steps of development can already be explicitly crowdsourced to specific platforms. Crowdsourcing design portals (99designs, 2012) allow leveraging the crowd to retrieve UI design concepts. Particular coding tasks can also be crowdsourced. Sites such as TopCoder (TopCoder, 2012) conduct challenges that developers try to solve in code. Software testing is another sphere where crowdsourcing is very beneficial and efficient (uTest, 2012): Crowdsourcing allows testing applications or systems in different operation system configurations, different usage plans, and different types of users, involving many people with low effort. Additionally, virtually any tasks can be submitted to generic crowdsourcing platforms—for example, Mechanical Turk (Amazon, 2012) which Amazon Web Services provides as part of their cloud computing platform. However, these platforms do not consider complex tasks,such as software development, or collaborative task processing.

Service-oriented architecture (SOA) and, in particular, web services that enable cloud computing play a fundamental role in supporting flexible, cross-enterprise collaboration scenarios. In addition, several standards, specifications, and models render web services a convenient foundation for designing and deploying, but also monitoring and adapting dynamic service-oriented environments. For example, the Human-Provided Services (HPS) model (Dustdar and Bhattacharya, 2011) enhances the traditional SOA-based systems by enabling people to provide services with the very same technology used by implementations of traditional software-based services. An HPS interface allows humans to define and provide their services in a transparent way. Across this particular interface, they are able to participate in ad hoc as well as process-centric collaborations. Specifications such as WS-HumanTask (Agrawal et al., 2007) and BPEL4 People (Kloppmann et al., 2005), and additional extensions (e.g., (Schall et al., 2012), have been defined to address the lack of human interactions in service-oriented businesses.

All these tools and concepts pave the way for a global software development with distributed teams and demonstrate how to build a piece of software within a connected community. They simplify and improve distributed project development. However, there is no consistent solution integrating project management, crowd management, and collaboration support for general software development projects. In the next section we sketch a scenario to demonstrate the opportunities such an integrated approach would have to offer.

8.3 Benefits of software engineering from crowdsourcing

While the platforms and approaches mentioned above provide a good starting point, an automated environment for complex software engineering projects leveraging the crowd is still far in the future. An intelligent marketplace for development tasks is needed that binds social web technologies to existing development and project management platforms in order to foster a seamless crowd-enabled software engineering process. It supports all kinds of software development tasks (e.g., design, program, test, maintain) and incentive mechanisms to engage, motivate, and honor software developers to work on available tasks. Thus, companies are able to leverage the collective intelligence of developers from all over the world (external) or within large IT departments (internal) by platform-provided delegation mechanisms that are able to (semi-) automatically map tasks to solution capabilities, ranging from single individuals to the crowd. Developers and further stakeholders are supported to jointly work on open issues or unsolved problems, follow activities of

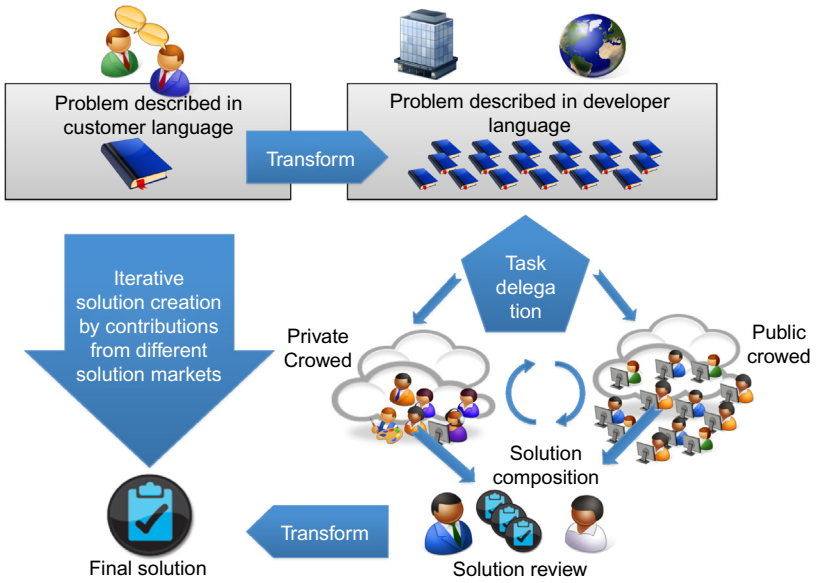

FIGURE 8.1

A crowd-enabled software development environment.

others, and share development results using communication paradigms known from social networks. Appropriate management instruments are provided to involve crowd members in ongoing development processes, to monitor their development progress, and to establish adequate quality assurance mechanisms for the integration of externally developed software fragments.

Figure 8.1 presents a scenario that exemplifies how crowd-enabled software engineering supports seamless integration of the crowd in the development process:

- **Domain-specific problem description:** A customer requests a solution to a problem via the crowd marketplace. Therefore, he or she needs to formalize the problem using a task-specification language, a nontechnical language to specify software development tasks, and their functional and nonfunctional properties. The project manager of an interested IT company gets in touch with the customer to specify the problem description further. Tools help to collaboratively define the requirements and the scope of the software.
- **Software architecture and development task definition:** The project manager translates the customer problem into a list of development tasks for functional and nonfunctional requirements and starts a new project. The software must be split up into components and services that need to be developed. Each of these components and services has a description that is typically quite informal at the beginning and will be refined into more formal definitions (e.g., interface descriptions) over time.
- **Task delegation:** Development tasks and related component and service definitions are the starting point for breaking work down into smaller pieces or tasks, which are then submitted via the crowd marketplace to a private or public crowd. The private crowd consists of employees,

organized or classified by organizational rules, while the public crowd consists of people outside the company, for example, specialized small companies or people in domain-specific communities, such as developers and testers on programming language-dedicated networks. Experts who understand the system and who can estimate the complexity of a task and its required prior knowledge and expertise should perform task creation and delegation. If a task requires deep background knowledge, it can be delegated to the internal crowd or to a person with a high level of expertise. If some required modules are highly interconnected, they can be offered as a combined task to a single person.

Based on the task and its constraints, different incentives are applied to motivate people to start creating a solution to the given problem. Tasks contain a description of work and of task execution environment (such as compiler, editor, testing software), criteria for testing the successful implementation of a solution, and general aspects regarding the task (e.g., milestones, due dates, rewards). It is possible that different people worldwide will compete on the same task, and the best solution wins; the discovery of people suitable for a task is performed by the platform.

In complex environments it is vital to consider the worker's expertise for task delegation. Lacking domain knowledge of software developers who are new to an area is a major source of misunderstandings and bugs in a traditional development and might be more pronounced in a crowd-sourced development. This problem is mitigated by deep integration of testing and by ensuring that developed code is actually working correctly. Such software testing results and votes of reviewers may provide feedback used by the platform to better estimate the capabilities of its members.

- **Iterative and collaborative software development:** Individual components and the resulting final solutions are developed in an iterative and evolutionary way, allowing early feedback from the customer regarding different stages of development. A notification engine informs developers about changes done by other developers and thus improves awareness and collaboration. Activity feeds, user-defined filters, and integrated development environment (IDE) integration allow developers to better understand the environment of their components and to monitor development progress in the context of such highly distributed projects. For example, developers can follow certain Java classes to get notification of changes made by other developers. Additionally, related information from social networks (e.g., Twitter, forums, or corporate-internal networks) is also made available in the activity feeds to the developers.

The whole approach is recursive; that is, every member of the crowd can also act as a customer—in other words accepting a request for work and delegating it to the private/public crowd.

In the next section we describe academia and industry efforts to tackle the three challenges identified in the Introduction and can help to realize crowd-enabled platforms aiming at professional software engineering.

8.4 Toward crowd-enabled software engineering

To implement crowd-enabled software engineering supporting introduced use cases, a number of research challenges, as discussed before, must be addressed. In the following we focus on approaches that, at least partially, try to solve the challenges identified in the Introduction.

8.4.1 Challenge 1: How to manage and organize software projects that are conducted, at least in part, by the crowd

While well-established methodologies and tools are typically used for managing traditional software projects (Institute, 2009), crowdsourced projects require different approaches. In addition to common project management challenges (Charette, 2005), managers of crowdsourced projects have to cope with issues such as a partially unknown workforce, fluctuating development practices, and communication barriers. The following subsections detail these problems and present ideas to solve them.

8.4.1.1 Project staffing

In traditional software projects conducted in small and consolidated companies, project managers usually know potential developers prior to the project setup. As a consequence, managers have at least an impression of each developer's skill set and performance when staffing the project. Using external workers such as freelancers is only considered if certain skills are rare or missing within one's own workforce. Large IT companies such as IBM and Microsoft employ thousands of developers distributed over many locations and product groups. A distributed workforce of this kind can be considered as an internal community or almost as a crowd. In crowd-conducted software projects, the workforce is only partially known to managers prior to the project setup because the staff is composed anew and dynamically, depending on the skills required to achieve the project goals. This carries many risks for project managers who have to manage their budgets, predict the project duration, estimate work performance, and, hence, determine the number of workers required to be involved and paid in the project.

Suitable tools have to support managers in their efforts to staff projects with qualified personnel by enabling them to express their needs for certain skills and qualifications in a simple and objective way. Today's professional network services such as LinkedIn (LinkedIn, 2012) and XING (Xing, 2012) allow searching for keywords in a person's online profile, retrieve short CVs, and show skills endorsed by other professional contacts. However, they do not offer facilities that go beyond these functions.

We propose a system that enables comparing potential staff members by their skill set, experience, and rating by former team members, managers, and customers. This system provides capabilities to organize and present the qualities of crowd workers on the one hand and to offer facilities to define project and job advertisements of the crowdsourcers on the other hand. In addition, the system allows project managers to define budget and time constraints in order to check the cost and availability of potential staff members. An approach to technically realize parts of this system is presented in section 4.2.1.

8.4.1.2 Project monitoring and controlling

In recent years, agile project management and development methodologies have found their way more and more into the software engineering process. Focusing on principles such as individuals, working software, customer collaboration, and change management (Beck, 2001), they are most suitable for crowdsourced software projects. Both in traditional and in crowd-conducted software projects that gather customer needs, setting goals and scope and defining tasks are critical success factors. Solely applying traditional development approaches to crowd-conducted projects, however, is insufficient and dangerous. For instance, as crowdsourced projects are typically characterized by

their dynamic and flexible nature, the problem of managing changes while using static, preplanned, and long-ranging tasks and resource allocations would be even more prominent. Compared to traditional software projects in crowdsourced ones, definitions of tasks have to be even more precise and expressive. While employees for in-house undertakings undergo a number of training sessions in order to share the same work principles, speech, and standards, in crowdsourced projects this common understanding, knowledge basis, and working behavior cannot be assumed and is probably missing. In addition to expressive tasks descriptions, the tasks themselves must be small in size and self-contained that is, they cannot depend on other tasks. This allows re-delegating incompletely or incorrectly implemented tasks to other developers without heavily impacting the project schedule and related components. Project managers are enabled to easily replace crowd workers who do not seek to accomplish their tasks within the parameters they have agreed to. To ensure the completion of mission-critical tasks, project managers can assign such tasks to more than one crowd worker and use the best or fastest result. By applying agile methodologies in crowdsourced projects, fluctuating development practices of crowd workers and risks such as poor quality, bad reusability, or missing extensibility of results can be reduced to the size of a single work unit when adequate monitoring and controlling is in place.

Organizational network analysis as an agile software engineering modeling method allows managers to monitor and measure collaborative efforts across teams (Carroll et al., 2013). Well-established agile project management software such as Rational Team Concert (IBM, 2012), VersionOne (VersionOne, 2012) and TeamPulse (Telerik, 2012) allow managing human resources and work units. Those software products support project managers and developers in their daily work. They enable project managers to plan, organize, monitor, control, and review tasks as well as measure the project progress and team performance, and to generate charts and reports for the upper management. However, they are not enabled for crowdsourcing.

In order to assist project managers in crowdsourcing projects, we propose a toolkit extending today's agile project management software by (1) facilitating the preselection of suitable crowd workers, (2) applying a test-driven model to ensure the functionality and quality of work units and (3) providing semiautomatic feedback on the crowd workers' performance. Automatically detecting relevant keywords from task descriptions written by the project manager, as briefly described in Dustdar and Gaedke, 2011), helps to preselect suitable crowd worker candidates based on their matching skills. Such a method implicitly ensures the precise definition of tasks by the project manager and the expressive description of crowd workers' profiles (cf. the previous section on project staffing'). In case Scrum, as an agile project management approach, is applied and a new Sprint is to be planned, the tool supports project managers in that relevant tasks are preassigned with the best-fitting crowd workers and in accordance with budget and availability constraint settings as described above. Although tasks can be semiautomatically assigned to the most promising crowd worker, the quality of their solutions cannot be anticipated with certainty. Hence, we suggest implementing unit tests for each work item before dealing with the actual work item. Depending on the type and complexity of the crowdsourced project, the unit tests are created in-house or by the crowd. Quality of task implementations interfaces and the entire project's reusability and extensibility can be increased using a test-driven approach, that is, through detecting incorrect results by associated unit tests. Furthermore, the tool assists project managers in identifying crowd workers having low working standards, fluctuating development practices, or poor performance. This in turn provides feedback on crowd workers, allowing assessment of their suitability in future projects (cf. Figure 8.2).

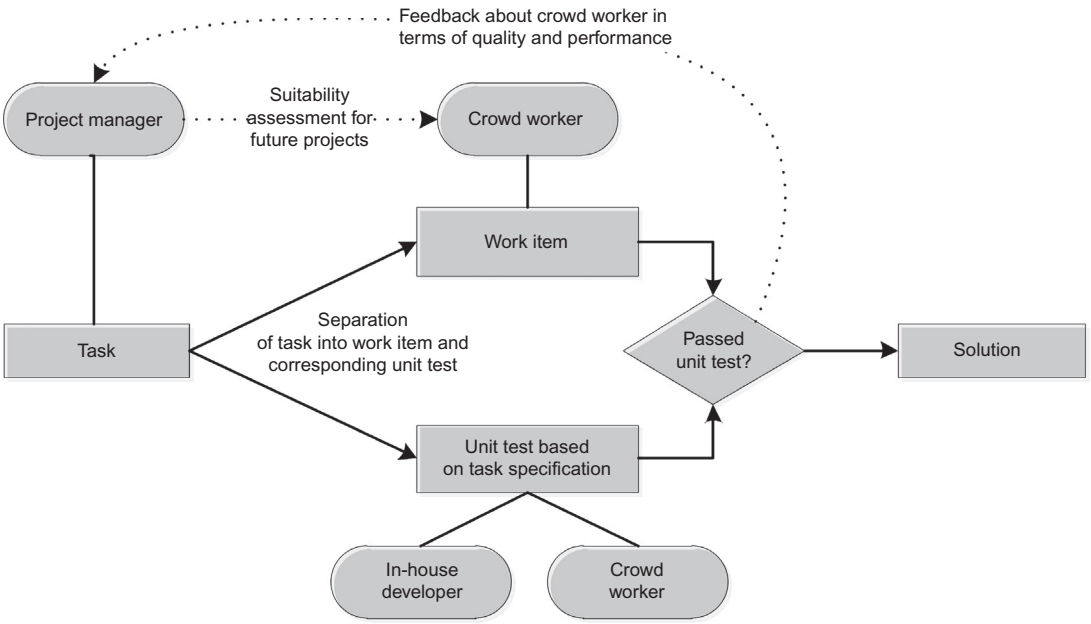

FIGURE 8.2

Quality assurance and assessment of crowd workers.

8.4.1.3 Project communication and documentation

Addressing the individual information needs of people involved in projects is typically implemented by means of a plan managing type, matter, and frequency of communication. Regular team meetings, status reporting, or review sessions are essential parts of a communication plan in traditional projects. In crowdsourced projects, however, such conventional communication methods are of limited usefulness as the staff is distributed, loosely coupled, and generally not continuously available over the entire project duration. For instance, team meetings do not make that much sense because external staff members—the crowd workers—are dynamically employed on a task basis. In line with the employment, we recommend communication on a task basis. As an example, when crowd workers need to discuss organizational issues in the context of their tasks, they have to contact the project manager using one of the preferred communication channels listed in the related profiles on the crowdsourcing platform. Different cultures and time zones can cause communication problems, which hinder an efficient collaboration. Technical issues requiring no special project knowledge can be raised by the crowd worker in a public, language-independent, collaboratively edited question and answer service such as the Stack Overflow (StackExchange, 2012). Although no direct communication between the crowd workers of a single project is needed (cf. section 4.1.3), we recommend that the project manager set up a wiki in the project preparation phase. Based on the flexible and evolving nature of a wiki, it is well suited to store the crowd-conducted documentation of the project. Besides the source code annotations of the crowd worker solving the

assigned task, more comprehensive documentation is beneficial for the project manager to understand and evaluate the solution.

8.4.2 Challenge 2: How to create virtual teams consisting of suitable experts and delegate tasks (semi-)automatically?

Crowdsourcing platforms are still relatively rudimentary and lack automated rewarding and task matching. Crowdsourcing of complex, collaborative tasks, such as software development, is not supported. Amazon Mechanical Turk (AMT) (Amazon, 2012) is a prominent representative for a generic crowdsourcing platform, which allows processing of tasks by crowd workers. Employers on the AMT, so-called *requesters* (service consumers), are encouraged to submit tasks, so-called *human intelligence tasks* (HITs), to the system. Those HITs mostly require minor effort (most of them can be processed in a few seconds up to few minutes), but they still need human intelligence, notably, transcription, classification, and categorization. The crowd of contributors, so-called *workers*, picks up these HITs, completes them, and gets a monetary reward (normally a few cents per HIT) if the quality of the requester accepts their solution (Ipeirotis et al., 2010).

In order to achieve Challenge 2, crowdsourcing platforms must be able to also support complex, cooperative software development tasks. Crowdsourcing capabilities must be seamlessly integrated into software management approaches and collaborative software development tools.

8.4.2.1 Auction-based crowdsourcing

Satzger et al. (2011, 2013) propose using auctions for task matching in crowdsourcing. Auctions are a popular way of trading goods on the Internet. Market laws achieve price discovery automatically. However, they are not used in current task-based crowdsourcing systems. The big advantage is that rewards for tasks are based on supply and demand; moreover, requesters do not have to "guess" a fair, competitive reward as with current platforms such as MTurk, but they may still define a maximum amount of money they are willing to pay. This prevents overpaying or underpaying; underpaying may be as bad as overpaying and result in "nonsellers" with the potential to cause delays, for example, in the context of business process execution.

Figure 8.3 illustrates how requesters and workers interact with such an auction-based marketplace. At first, a requester submits a new task to the platform. The requester provides a description of the task and the maximum amount of money (reward) he or she is willing to pay. The marketplace then creates an auction for this task, inviting only those workers to the auction whose skills match those specified by the requester. Workers interested in processing the task can place their bids in the auction. After a predefined amount of time the auction closes, determines a winner, and informs the worker about winning the auction. It is now the job of the worker to process the task in time and to submit the result to the platform. Assuming the task has been processed in time, the marketplace informs the requester, who in turn provides a rating for the worker based on the quality of the processed task. If the quality conforms to the predefined agreement, the requester transfers a defined amount of money, based on the outcome of the auction and on a possible fee for using the crowdsourcing platform, to the marketplace. Considering the requester's feedback, the marketplace updates the skill profile of the worker and finally pays the worker for his or her effort.

The auctioning mechanism can be extended so that it does not blindly look solely at prices but instead also includes bid re-ranking techniques to ensure quality. In addition to the pure mapping

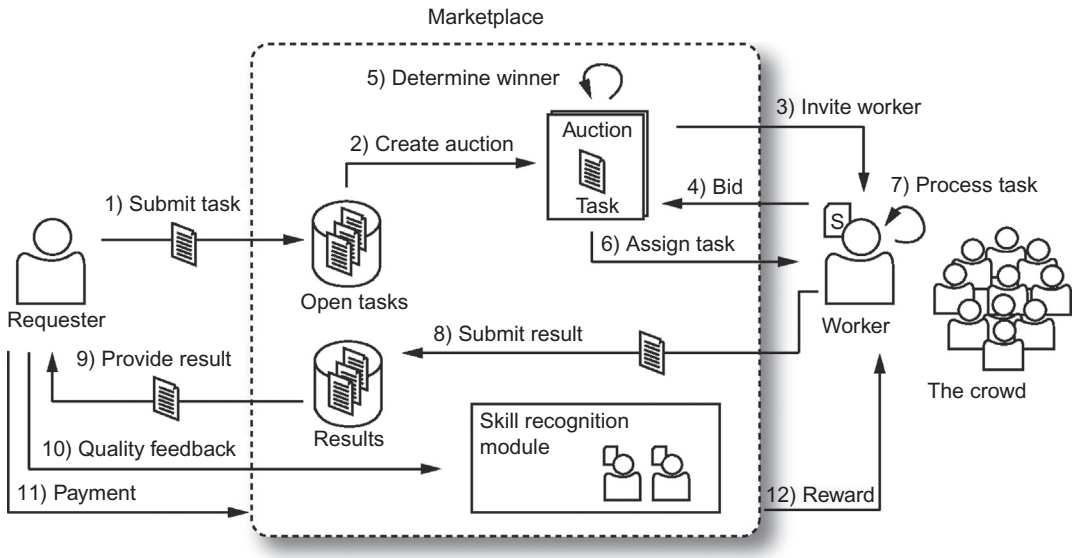

FIGURE 8.3

Auction-based crowdsourcing process.

of tasks to workers, also addressed is the issue of how to build and manage an automated crowdsourcing platform. For establishing a successful crowdsourcing environment, it is important to maintain a motivated base of crowd members and provide the stimulus for learning required skills. Only a recurring, satisfied crowd staff is able to ensure high quality and high output. We propose a skill evolution model that encourages new and existing crowd workers in developing capabilities and knowledge needed by requesters. All standard processes in the crowdsourcing platform are automated and free from intervention, which allows handling a vast amount of tasks and makes it compatible with a SOA approach. To ensure sustainability, the model is designed not only to maximize the benefit of requesters, but also to take the welfare of workers into account.

8.4.2.2 Crowdsourcing of workflows

In the future, companies will increasingly use crowdsourcing to address a flexible workforce. However, how to carry out business processes leveraging crowdsourcing remains an open issue. Most task-based crowdsourcing platforms simply provide requesters the possibility to publish simple task descriptions into a database to which all workers have access. A task description in AMT, for instance, consists of a title, textual description, expiration date, time allotted, keywords, required qualifications, and monetary reward. Business processes or workflows, on the other hand, describe a logical structure between tasks that crowdsourcing platforms cannot handle. The main problem is, however, that people book tasks voluntarily in crowdsourcing, which means the only way to influence the booking and execution times of single tasks is either to change incentives or to modify other aspects of a task, for example, define a later deadline.

Khazankin et al. (2012) describe an enterprise crowdsourcing approach that executes business processes on top of a crowdsourcing platform. For each single task in the business process, we reason about the optimal values for incentives (typically monetary reward) when crowdsourcing them. The goal is to carry out the business process with minimal investments before the deadline. During execution of the business process, we constantly monitor the progress and adjust incentives for tasks that a worker has not yet booked. Our approach for calculating optimal values is based on mining historical data about task executions, which influence higher rewards have on the booking time, analyzing the current status of the business process, and quadratic programming, which is a mathematical optimization formulation that can be solved efficiently.

The main goal is to ensure the timely execution of business processes that contain crowdsourced tasks while minimizing the expenses associated with crowdsourcing rewards. Expenses can be reduced by setting lower rewards for tasks, but if the reward is too low, a task might remain not booked for too long, if booked at all. Such situations can significantly affect the execution of the process, and become a reason for missed deadlines. The main idea of our approach is to find a most beneficial trade-off between rewards and booking/processing times for crowdsourced tasks within a business process.

Figure 8.4 shows a framework for deadline-driven reward optimization for processes containing crowdsourced tasks. The estimator collects statistical data from platform logs and estimates the functional dependency between incentives and booking/processing times for each type of task ①. The optimizer retrieves structure and state of processes ③, booking state of already published tasks ④ and functional dependencies ②, and determines optimal values, mainly for setting monetary rewards. These values are further used by the publisher, ⑤ which announces tasks to the platform at an appropriate time and updates them if needed ⑥. The optimization and publisher are activated periodically, thus realizing the adaptive behavior.

8.4.2.3 Games with a purpose

Some software development tasks do not require special education and can be solved by almost anyone, at least if they are presented accordingly. Such tasks could be distributed with existing crowdsourcing platforms. However, alternatives are available that promise to get solutions cheaper, faster, and sometimes even with better quality. An interesting crowdsourcing approach based on different forms of incentives is games with a purpose, that is, human-based computation games that outsource certain computation logic to human players in an entertaining way (von Ahn, 2006). Note that this idea is somehow different from the idea of "serious gaming" that describes games whose main purpose is to educate or improve some skill, instead of entertaining (Johnson, 2007).

This niche is actively being developed and focuses on possibilities to wrap crowdsourcing tasks into a game that presents computational challenges to humans in an entertaining way. Workers are motivated and interested in the task-solving process because it is fun. Especially if played within a community, it is often additionally motivating to get the highest score, which commonly represents the best solution for the stated problem. Foldit (2012), a set of online challenges, but also GWAP (2012) and Phylo (McGillUniversity, 2012) belong to this category.

We propose reorganizing the game with a purposed approach by giving game developers more control over tasks that can be integrated into games, which can result in fun-to-play, state-of-the-art games that elegantly disguise tasks by sophisticated in-game challenges. If the crowdsourcing task

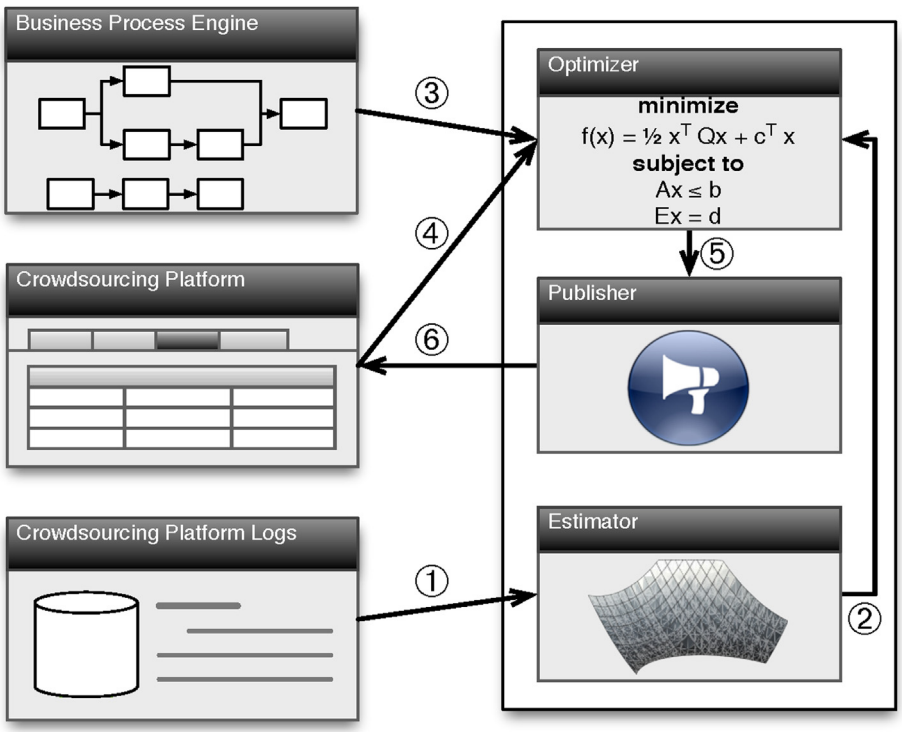

FIGURE 8.4

Architecture of the crowd-enabled workflow engine.

is to detect the location of a specific object on an image, the player can be told that pointing the correct object will open a secret door. If the crowdsourcing task is to classify the content of a photo, it could be presented simply as enemy, and observing which weapon, spell, or potion a player picks for attacking or defending would allow learning about the image category.

The main idea is to split today's unnatural merger between the institution that creates the crowdsourcing task and the game developers. Professional game studios and indie game creators would benefit from the cooperation with institutions that need to process human-only solvable puzzles. On the other hand, having experts embed tasks into games greatly increase chances of acceptance.

As shown in Figure 8.5, at the core of the idea is the mediator, a puzzle broker that mediates typical crowdsourcing puzzles to game developers of high-quality games. In contrast to paid crowd workers, a crowd of players solves the tasks transparently in the context of the game. The puzzle broker can offer better reliability and throughput to its customers because of a big player base. This helps game developers to provide cheaper or even free games to its customers. Additionally, developers and players benefit as the game content is highly varying and new puzzles must be solved each time even for the same level or quest.

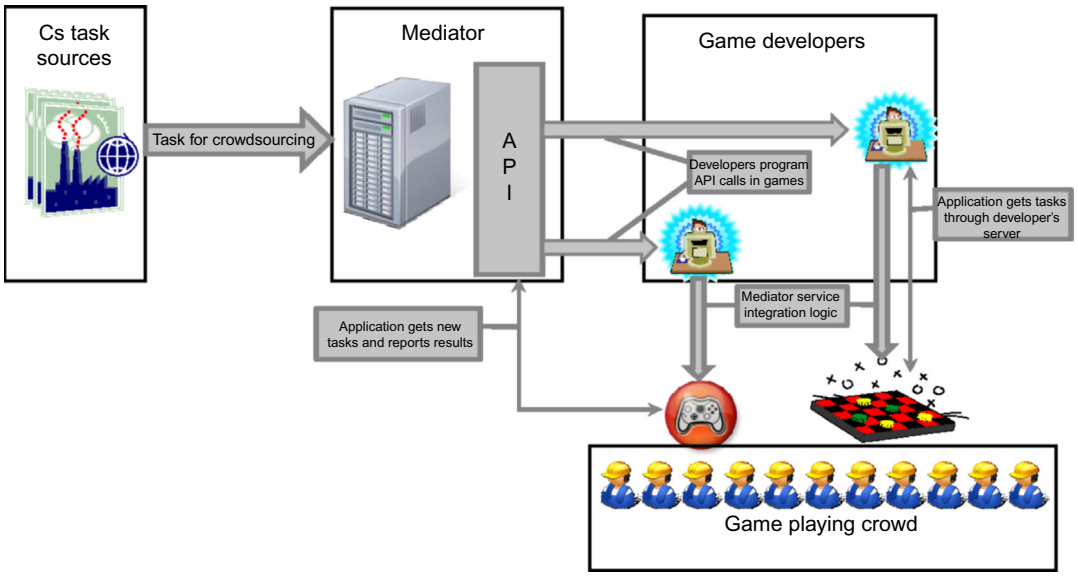

FIGURE 8.5

Better integration of crowdsourcing tasks into games with a purpose.

8.4.3 Challenge 3: How to improve cooperation of developers within loosely coupled teams?

The development of high-quality software products in a loosely coupled, web-based, collaborative setting, as we envision it for crowd developed, leads to a range of new requirements that existing development tools can only partly fulfill. Agile development typically requires the co-location of all involved stakeholders. Within crowd development setups, this constraint cannot be fulfilled due to the distribution of the crowd developers. Indeed, the actual source code development does not necessarily require the presence of all developers in one physical room. However, the early phases of a software development project are characterized by discussions and brainstorming in order to define the scope, functionality, and quality of the product. These meetings require the attendance and active participation of all stakeholders (e.g., back-end developer, quality engineer, UI designer, mobile app expert) in order to create a common understanding within the development team. During the development phase, distributed co-development between people who may not be part of the same company or organization increases the need for sophisticated update and notification mechanisms in order to follow the latest development and understand the full context. The following subsections discuss potential ways of cooperation and highlight our research and development results that promise to improve the collaboration within such distributed work environments.

8.4.3.1 Crowd-enabled collaboration

Unlike in traditional projects, where common procedures, routines, and practices such as regular team meetings, brainstorming sessions, or assisting co-workers are essential parts of the daily

business, in crowdsourced projects the workers do their job in a decoupled manner. As a result of such loosely coupled and distributed team structures, it is difficult to implement measures that will enable collaboration between team members and strengthen a team's cohesiveness, identity, and work principles (Kittur et al., 2013; Reason and Andersen, 1991). Company policies rigidly defining labor standards, security guidelines, or the tool chain are unsuitable for crowd-enabled undertakings. This is because sufficient compliance implementation and control are difficult to realize because of the diversity of the work environments of the crowd workers. While in-house project developers can share data via dedicated servers, in crowd-conducted software development data have to be managed differently in order to cope with restrictions in terms of accessibility and exchangeability.

Because of these facts, a flexible collaboration environment for facilitating discussion, decision making, and data management is required. Going beyond the capabilities of established social collaboration platforms such as Jive (Jive, 2012) or Yammer (Microsoft, 2012), we recommend an environment that enables crowdworkers in the creation of highly customizable and tailored workspaces. Workspaces are individual compositions of predefined components serving specific purposes. This includes traditional collaboration and communication capabilities (e.g., telephony services, instant messaging, voting, whiteboard, document upload) as well as specific services tailored to meet the need in distributed development projects (e.g., task overview, access to documentation material, knowledge database). Crowd workers temporarily integrated into development projects are able to join the collaboration platform and design their individual workspaces out of these components. In this way, they include components that are only valuable for them as well as shared components that foster collaboration and awareness between co-workers. As an example, a crowd worker could see other developers currently working on the same project or area and ask for support via a component offered by the crowdsourcer.

The technical foundation for such a collaboration environment has been developed within the OMELETTE research project (Chudnovskyy et al., 2012a). Based on sound technologies provided by the open-source projects Apache Rave (Apache, Apache Rave, 2012) and Apache Wookie (Apache, Apache Wookie Project, 2012), a collaboration platform as described above has been developed. The platform enables users to create, customize, and share workspaces out of OpenSocial (2012) and W3C (2012). In order to offer crowd workers these forms of advanced communication and collaboration capabilities, a comprehensive set of widgets ranging from video chats, conference calls, and screen-sharing solutions over shared text-editors, collaborative whiteboards to calendars and contact lists have been developed. Workspace descriptions can be exported using Open Mashup Description Language (OMDL) (2012), which allows it to be used as the basis for other workspaces or as a workspace template. As a data management platform, we recommend an extensible and flexible system, such as the WebComposition/Data Grid Service (DGS) (Chudnovskyy et al., 2012b). The DGS facilitates, for instance, the handling of heterogeneous data by dedicated and specialized data storage engines as well as the definition of fine-grained access control rights for crowd workers using the emerging WebID (W3C, WebID 1.0—W3C, 2005) standard. The combination of both platforms constitutes a solid foundation for crowd-enabled collaboration.

8.4.3.2 Collaborative requirements engineering

Requirements engineering is a crucial phase at the beginning of every product development to define the scope of development together with customers. As mentioned in the introduction of this

section, the early phases of a software development project and in particular the requirements engineering are characterized by discussions and brainstorming in order to define the scope, functionality, and quality of the product. Therefore it is essential to support active collaboration between all co-workers in order to establish a common understanding of the envisioned product. In the traditional waterfall model, requirements engineering is only executed in the first phase of development, followed by analysis, design, and the actual software development. Many existing solutions (e.g., Briggs and Grünbacher, 2002; Davis, 2010) focus on rather complex (nonagile) methodologies for requirements engineering that have not been widely adopted in the software industry. These existing solutions do not seem to be feasible for crowd development because they are quite inflexible. Instead, agile methods such as User Story Mapping (Maurer and Hellmann, 2013) together with agile development approaches such as SCRUM or Kanban (Leffingwell, 2010) have been established in the software industry in recent years. They repeat requirements engineering techniques in several iterations during the whole software development process. This way findings and customer feedback from the ongoing project can be used for further planning. The result of the requirements engineering process is captured in a so-called Prioritized Product Backlog. The product backlog contains a set of user stories that describe the product features from a user's perspective. The individual user stories can be used as a means to distribute development tasks to the crowd if they have the right granularity.

Based on this general overview of requirements engineering for crowd development, we can identify three main challenges:

Defining a set of user stories with the right granularity: User stories should be ideally self-contained without dependencies on other user stories. Then development can be done in parallel by different developers with low interference and communication demand. However, this ideal is often difficult to achieve in practice.

Supporting collaboration of remote teams: Agile techniques for requirements engineering typically advocate co-located teams for most efficient work and make heavy use of paper, whiteboards, blackboards, sticky notes, and the like. However, crowd development is fundamentally distributed in nature. Thus, tools fostering communication and collaboration between remote developers are required. The kind of tools ranges from general-purpose tools such as videoconferences, online chats, wikis, and forums to specially tailored tools for specific collaboration or communication tasks.

Supporting iterative refinement of prioritized product backlogs: At any point in time during product development, it must be easily possible to refine and change existing requirements in the form of the prioritized product backlog.

While the aforementioned OMELETTE collaboration environment addresses the second challenge, dedicated tool support is required to address the other two. The following examples constitute the results of our real-time collaboration research in the context of agile techniques. Both applications share a collaborative character, meaning that they enable multiple distributed web users to work on a shared content in real time.

The *affinity diagram* (Holtzblatt and Beyer, 1993) is a well-known technique for gathering and structuring large amounts of data or ideas by a group of people. For example, results from customer interviews can be captured to derive key results from it. In the context of requirements engineering, affinity diagrams can be used in early phases to obtain more insights into the new product and,

thus, prepare the definition of the product backlog. The actual process behind affinity diagrams is quite simple:

1. Record each idea or whatever data on a card. All involved people can do this in parallel.
2. Put all cards on a wall and group related cards to clusters. People can discuss the clustering.
3. Find names for clusters.

In order to leverage this well-known technique for crowd development, we have developed a web-based collaboration tool (see Figure 8.6) that allows distributed teams to follow the same process with virtual cards. The focus of the tool development is on usability to support an easy, intuitive, and fast-working mode that tries to come close to the work with physical cards, pens, and walls. Users are able to add or remove cards, edit the content of cards, and group related cards in clusters.

User story mapping (Patton, 2009) is an established agile technique used to define the scope of an envisioned product. It involves the whole team, including internal and external stakeholders, and its goals are to produce a common understanding of the product, create user stories, and visualize the backlog as a structured map.

Similar to the presented affinity diagram tool, we have developed a web-based tool (Figure 8.7) that supports creating and editing user story maps. Most operations can be performed through drag and drop, similar to the working mode with physical cards. By also supporting tablet computers-such as the iPad, user interactions become even more intuitive and close to the physical model.

Although agile techniques such as user story mapping or affinity diagrams are usually done with pen and paper, dedicated (web-based) tools offer several advantages:

- Results are available in electronic form and can easily be archived.
- Results can be exported to other tools. For example, we have developed an interface to the popular backlog management tool JIRA (Atlassian, 2012) for the user story mapping tool, which allows exporting user stories into JIRA.

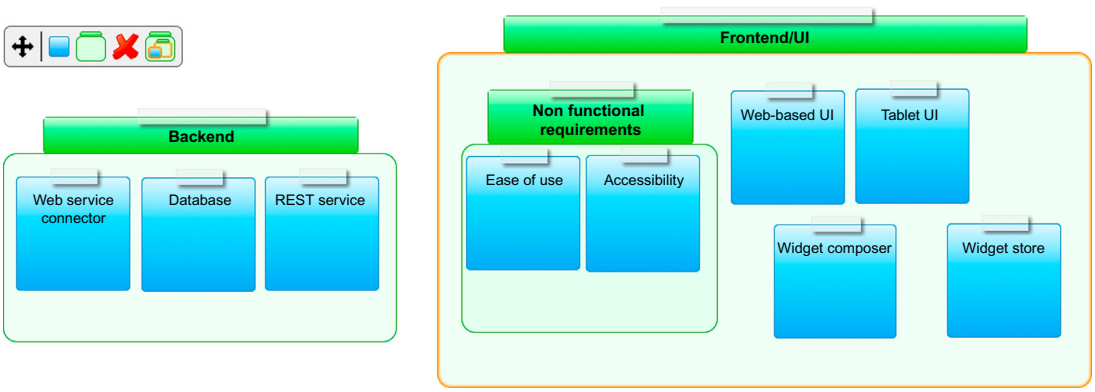

FIGURE 8.6

Affinity diagram tool.

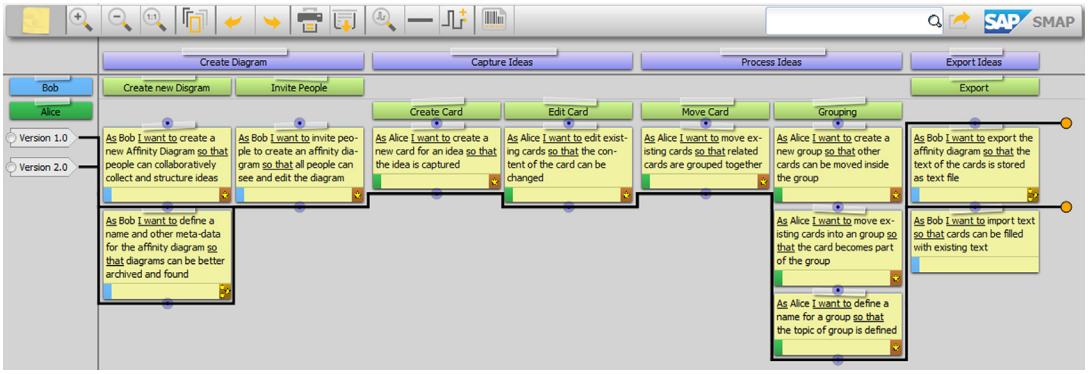

FIGURE 8.7

User story mapping tool.

- Data can be changed more easily. There is no need for striking through or writing text to a new card. Additionally, a virtual wall has no size limitation that one would face using a real wall.

However, these general advantages apply for almost all virtualized tasks and do not sufficiently differentiate our work from existing web applications that cover the same functionality (e.g., SilverSories (Agile Story Mapping Tool | Silver Stories, 2011) or CardMapping (Jeremy Lightsmith, 2011) for user story mapping). As signaled in the introduction of the tools, we advanced the state of the art by adding real-time collaboration support for multiple distributed users. As crowd workers by definition cannot physically be present in workshops or brainstorming sessions, tools must eliminate this limitation as much as possible. Therefore, the presented tools utilize real-time collaboration based on operational transformation mechanisms (Ellis and Gibbs, 1989). This technology allows multiple crowd workers to interact on shared artifacts such as a user story or a single sticky note in real time. All changes are immediately synchronized and propagated to all active participants of a session. Errors are resolved automatically in order to secure one consistent state. Furthermore, the tools visually create awareness of users' interactions. In order to support developers in the implementation of this kind of collaborative applications, we have created two different development approaches. While the first one allows the transformation of existing single-user web applications into collaborative multiuser applications (Heinrich et al., 2012), the second approach enables developers to build new collaborative applications from scratch using an annotation-based development framework (Heinrich, 2013).

8.4.3.3 Social development

One of the challenges for globally distributed crowd developers is to follow the mutual development progress. Everyone involved needs to be up to date regarding changes done by others in order to (1) ensure the transparency of work, (2) improve the awareness of the actual situation, and (3) ease collaboration among all involved people. Currently, following the mutual development progress works at the level of file changes being committed to a version control system such as

GIT or SVN. Developers need to scan through all changes and manually find the relevant ones, which is time consuming and might miss important changes.

Therefore, we propose a social layer on top of traditional Version Control Systems (VCS). Transferring established concepts from social networks, such as activity feeds, gives crowd developers a better understanding of the work others do and helps them monitor development progress in the context of such highly distributed projects. Using sophisticated subscription and filtering mechanisms, the amount of relevant information for the developer can be minimized and the developer does not need to check all changed files of a commit. Moreover, it is beneficial to connect data from social networks and dedicated developer communities with the actual development activities or even particular code changes in order to get context-sensitive, helpful information. Finally, all involved developers need to be able to exchange ideas and share information within the public or private cloud using feed update mechanisms.

In order to receive notifications about all changes done by others, crowd developers can "follow" (as in Twitter) not only their peers but also artifacts of the source code such as a certain Java class or even single methods within Java classes. Compared to change sets in traditional VCSs that are only based on line numbers in text files, the system reports software changes at the level of known programming language concepts (e.g., classes, methods, member variables) in the form of activity feeds.

These feeds can be enriched with additional information from social networks (e.g., Twitter, forums, corporate-internal networks, online communities). Developers discussing issues and trying to understand error messages and exceptions frequently use these information sources. Linking this information with a specific change set helps other developers of the crowd to understand the background and context of the change. Consider, for example, a solution to a specific problem posted in online communities that can now be easily linked to a dedicated code snipped in order to give other developers the necessary background information to understand the actual implementation.

Indeed, following code artifacts and ongoing tasks of developers creates a huge amount of activity feeds. The question that arises is how a single developer can consume this flood of information in a nondisruptive manner. Integration into the actual development environment in a tight and context-sensitive way significantly reduces the disruption caused by switching tools (e.g., IDE, browser) or even devices. Features to summarize retrieved information as well as the ability to explore the repository structures and code evolution (submitted change sets) using easy navigation, search and filtering capabilities (e.g., only changes from last week, only changes from developer X) are considered in the proposed social layer created on the following components:

VCS Connector: The notification system requires a generic interface and specific adapters for selected VCSs (e.g., SVN, GIT, Perforce) in order to gather information about ongoing development activities. These VCSs serve as one primary source for the update and notification mechanism.

Program Language Analyzer: The classification of programming languages (e.g., statically/dynamically typed, OO, functional) constitutes the foundation for a generic programming language model (class, method, interface, member variables). This leads to a set of sophisticated analyzer components for specific programming languages (e.g., Java, C#) by using and extending available open-source frameworks. The result can be used to monitor ongoing development activities, create (semi-) automatic status updates, and follow code artifacts.

Artifact Linking and Notification Mechanism: A linking mechanism between code artifacts, developers, and additional information sources (e.g., Wiki, discussion, Twitter) is required in order to realize the traceability of development changes and connect the actual code with external information. Therefore, connectors to suitable social networks have to be created that enable developers to either retrieve relevant information from dedicated communities or publish status updates and information related to their current development tasks. Furthermore, each developer needs to have a profile that links to his or her social network profiles.
IDE Integration: Additional side panels, plug-ins, or entries in context menus enrich the existing development environment with new capabilities in an integrated manner. For staying focused on their actual implementation tasks, developers can apply filters and use search tools to decide what information is propagated in which way. Without leaving the work environment, they also can reach out to their peers in the crowd or consume the content of community pages using integrated browser views.

In summary, crowd development requires new ways of collaboration in order to ensure the time and quality of a product. The efficient usage of the combined intelligence of a large group of developers, the seamless integration of knowledge stored in social communities, the required tool-supported awareness of actions and changes done by others, as well as the general form of social interaction during the development phase are the key differentiators in traditional software development.

8.5 Conclusion

Within the current trends toward cloud computing, many IT companies are interested in new ways of dealing with the workforce. Inspired by emerging success stories of crowdsourcing, IT companies are trying to improve their business processes with the help of this new paradigm. However, applying crowdsourcing comes with a set of new challenges that are gaining the attention of research and development.

The distributed nature of crowdsourcing requires new methodologies for project management. This problem is partially solvable by adapting agile development techniques using existing social collaboration approaches, but as of today, no complete solution is available.

Another challenge is how to distribute tasks, motivate workers, and set up team structures dynamically in order to produce high-quality software. Existing platforms, such as Amazon Mechanical Turk (Amazon, 2012), provide fundamental features for task delegation, but they leave issues such as task generation, worker evaluation, and motivation unsolved. We present possibilities to automate the crowdsourcing of business processes and different ways to motivate workers through auction-based or entertainment-based task distribution.

In crowd-enabled software development, workers typically have to collaborate remotely during various development steps. Web-based social collaboration tools, for example, forums, instant messaging platforms, online office suites, or social networks, provide rudimentary features for collaborative software development. However, new collaboration tools are emerging that can be used to create crowd-enabled IDEs and management tools.

However, this chapter does not claim to present a complete list of the challenges that face companies trying to apply crowdsourcing technologies. As is true of any new technology, crowdsourcing comes with other challenges that are not directly related to the development process. For example, it is an open research topic how companies have to handle legal issues such as intellectual property rights, payment of rewards, taxation, and business secret protection. Even though all of these and further issues are very important and vital for a company's adoption of crowdsourcing , they are out of the scope of this chapter, which, rather, focuses on the core challenges.

Overall, the benefits of crowd-enabled software engineering are beginning to be acknowledged, and to some degree similar concepts are already in use. However, there is no holistic approach allowing crowd-enabled project management, crowd management, and integrated tool support. The concepts presented in this chapter try to solve these problems and show the current research efforts that can help to improve today's experience with crowd-enabled software engineering.

References

99designs, 2012. Logo Design, Web Design and More. |. Retrieved from 99designs. <http://99designs.com/>.

Adobe, 2012. Adobe Buzzword. Retrieved from Adobe Buzzword. <http://www.adobe.com/uk/acom/buzzword/>.

Agile Story Mapping Tool | Silver Stories, 2011. Retrieved from Silver Stripe Software. <http://toolsforagile.com/silverstories/>.

Agrawal, A., Amend, M., Das, M., Ford, M., Keller, C., Kloppmann, M., et al., 2007. Web Services Human Task (WS-HumanTask), Version 1.0. Available at: <http://incubator.apache.org/hise/WS-HumanTask_v1.pdf>.

Amazon, 2012. Amazon Mechanical Turk. Retrieved from: <https://www.mturk.com/>.

Apache, 2012. Apache Rave—The Apache Software Foundation! Retrieved from Apache Rave: <rave.apache.org/>.

Apache, 2012. Apache Wave—Welcome to Apache Wave (incubating). Retrieved from Apache Wave: <http://incubator.apache.org/wave/>.

Apache, 2012. Apache Wookie Project. Retrieved from Apache Wookie: <wookie.apache.org/>.

Assembla, 2012. Kanban/Scrum Task Board with Collaboration Tools Configuration. Retrieved from Assembla: <https://www.assembla.com/catalog/101-kanban-scrum-task-board-with-collaboration-tools-package>.

Assembla, 2012. The Next Generation Agile Project Management Tools | Assembla. Retrieved from Assembla: <https://www.assembla.com>.

Atlassian, 2012. Issue and Project Tracking Software | Atlassian. Retrieved from Atlassian JIRA: <http://www.atlassian.com/software/jira>.

Beck, K.A., 2001. Manifesto for Agile Software Development. Retrieved from Manifesto for Agile Software Development: <http://agilemanifesto.org>.

BerliOS, 2012. BerliOS: The Open Source Mediator. Retrieved from BerliOS: <http://www.berlios.de/>.

Bosari, J., 2012. The developing role of social media in the modern business world. <http://www.forbes.com/sites/moneywisewomen/2012/08/08/the-developing-role-of-social-media-in-the-modern-business-world>.

Briggs, R., Grünbacher, P., 2002. Easywinwin: managing complexity in requirements negotiation with GSS. Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)—Volume 1. IEEE Computer Society, Washington, DC, pp. 21.1. <http://dl.acm.org/citation.cfm?id=820887>.

Bruce, A., Pepitone, J.S., Formisano, R.A., Peitone, J.S., 1999. Motivating Employees. McGraw-Hill, New York.

Carmel, E., Agarwal, R., 2001. Tactical approaches for alleviating distance in global software development. Softw. IEEE 18 (2), 22–29.

Carroll, N., Richardson, I., Whelan, E., 2013. Service science: exploring complex agile service networks through organisational network analysis. In: Wang, X., Ali, N., Ramos, I., Vidgen, R. (Eds.), Agile and Lean Service-Oriented Development: Foundations, Theory, and Practice. Information Science Reference, Hershey, PA, pp. 156–172. Available from: http://dx.doi.org/doi:10.4018/978-1-4666-2503-7.ch008.

Charette, R., 2005. Why software fails. IEEE Spectr. 42–49.

Chow, T., Cao, D.-B., 2008. A survey study of critical success factors in agile software projects. J. Syst. Softw. 81 (6), 961–971.

Chudnovskyy, O., Nestler, T., Gaedke, M., Daniel, F., Fernández-Villamor, J.I., Chepegin, V., et al., 2012a. End-user-oriented telco mashups: the omelette approach. Proceedings of the 21st International Conference Companion on World Wide Web. ACM, New York, pp. 235–238.

Chudnovskyy, O., Wild, S., Gebhardt, H., Gaedke, M., 2012b. Data portability using web composition/data grid service. Int. J. Adv. Internet Technol. 4 (3 and 4), 123–132.

Computerworld, P.T., 2012. Microsoft signs outsourcing pact with Indian giant Infosys. Microsoft Signs Outsourcing Pact with Indian Giant Infosys. <http://www.computerworld.com/s/article/9175442/Microsoft_signs_outsourcing_pact_with_Indian_giant_Infosys>.

da Silva, F., Costa, C., Franca, A., Prikladinicki, R., 2010. Challenges and solutions in distributed software development project management: a systematic literature review. 2010 5th IEEE International Conference on, Global Software Engineering (ICGSE), pp. 87–96.

Dabbish, L., Stuart, C., Tsay, J., Herbsleb, J., 2012. Social coding in github: transparency and collaboration in an open software repository. Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work, pp. 1277–1286.

Damian, D., Izquierdo, L., Singer, J., Kwan, I., 2007. Awareness in the wild: why communication breakdowns occur. Second IEEE International Conference on Global Software Engineering, 2007. ICGSE 2007, pp. 81–90.

Davis, A., 2010. Requirements Bibliography. Retrieved from: <http://www.reqbib.com/>.

DiMicco, J., Millen, D.R., Geyer, W., Dugan, C., Brownholtz, B., Muller, M., 2008. Motivations for social networking at work. Proceedings of the 2008 ACM conference on Computer Supported Cooperative Work, pp. 711–720.

Dustdar, S., Bhattacharya, K., 2011. The social compute unit. Internet Comput. IEEE 15 (3), 64–69.

Dustdar, S., Gaedke, M., 2011. The social routing principle. Internet Comput. 15 (4), 80–83.

Ellis, C.A., Gibbs, S.J., 1989. Concurrency control in groupware systems. Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data. ACM, New York, pp. 399–407.

Fidelman, M., 2010. 5 Powerful Project Management Features You can only do on UnaWave (Google Wave).

Foldit, 2012. Solve Puzzles for Science | Foldit. Retrieved from Foldit: <http://fold.it/>.

GitHub, 2012. GitHub. Retrieved from: <https://github.com/>.

Google, 2012. Google Docs—Online documents, spreadsheets, presentations, surveys, file storage and more. Retrieved from Google Docs: <http://docs.google.com>.

GWAP, 2012. gwap.com—Home. Retrieved from GWAP: <http://www.gwap.com/gwap/>.

Heinrich, M.G., 2013. Exploiting annotations for the rapid development of collaborative web applications. Proceedings of the 22nd International Conference on World Wide Web. International World Wide Web Conferences Steering Committee, pp. 551–560.

Heinrich, M., Lehmann, F., Springer, T., Gaedke, M., 2012. Exploiting single-user web applications for shared editing: a generic transformation approach. Proceedings of the 21st International Conference on World Wide Web. ACM, New York, pp. 1057–1066.

Holtzblatt, K., Beyer, H., 1993. Making customer-centered design work for teams. Commun. ACM 36 (10), 92–103.

IBM, 2012. IBM—Rational Team Concert. Retrieved from IBM - Rational Team Concert: <http://www-03.ibm.com/software/products/us/en/rtc/>.

Institute, P.M., 2009. A guide to the project management body of knowledge (PMBOK Guides). Project Management Institute.

Ipeirotis, P., Provost, F., Wang, J., 2010. Quality management on amazon mechanical turk. Proceedings of the ACM SIGKDD Workshop on Human Computation, pp. 64–67.

Jeremy Lightsmith, J.P., 2011. Card Mapping. Retrieved from Card Mapping: <http://cardmapping.com/>.

Jive, 2012. Social Collaboration for Social Business. Retrieved from Jive Software: <www.jivesoftware.com/>.

Johnson, W.L., 2007. Serious use of a serious game for language learning. Front. Artif. Intell. Appl. 158, 67.

Khazankin, R., Satzger, B., Dustdar, S., 2012. Optimized execution of business processes on crowdsourcing platforms. 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom ′12). IEEE, pp. 443–451.

Kittur, A., Nickerson, J.V., Bernstein, M., Gerber, E., Shaw, A., Zimmerman, J., et al., 2013. The future of crowd work. Proceedings of the 2013 conference on Computer Supported Cooperative Work, pp. 1301–1318.

Kloppmann, M., Koenig, D., Leymann, F., Pfau, G., Rickayzen, A., von Riegen, C., et al., 2005. Ws-bpel extension for people—bpel4people. Joint white paper, IBM and SAP.

Koch, S., 2009. Exploring the effects of SourceForge. net coordination and communication tools on the efficiency of open source projects using data envelopment analysis. Empirical Softw. Eng. 14 (4), 397–417.

Lanubile, F., Ebert, C., Prikladnicki, R., Vizcaino, A., 2010. Collaboration tools for global software engineering. Softw. IEEE 27 (2), 52–55.

Leffingwell, D., 2010. Agile software Requirements: Lean Requirements practices for teams, Programs, and the Enterprise. Addison-Wesley, Boston, MA.

Leo Kelion, B.N., 2012. Linus Torvalds: Linux succeeded thanks to selfishness and trust. Retrieved from BBC News: <http://www.bbc.co.uk/news/technology-18419231>.

LinkedIn, 2012. LinkedIn. Retrieved from LinkedIn: <http://www.linkedin.com/>.

Madey, G., Freeh, V., Tynan, R., 2002. The open source software development phenomenon: an analysis based on social network theory. Proc. Am. Conf. Inf. Syst. 1806–1813, Dallas, Texas.

Malone, T., Laubacher, R., Dellarocas, C., 2010. The collective intelligence genome. IEEE Eng. Manage. Rev. 38 (3), 38.

Maurer, F., Hellmann, T.D., 2013. People-centered software development: An overview of agile methodologies. Softw. Eng.. Springer, New York, 185–215.

McGillUniversity, 2012. PHYLO | DNA Puzzles. Retrieved from PHYLO: <http://phylo.cs.mcgill.ca/>.

Microsoft, 2012. Microsoft Office—Microsoft Word, Outlook & Excel—Office.com. Retrieved from Microsoft Office 365: <http://office.microsoft.com>.

Microsoft, 2012. Yammer: The Enterprise Social Network. Retrieved from Yammer: <https://www.yammer.com/>.

Noll, J., Beecham, S., Richardson, I., 2010. Global software development and collaboration: barriers and solutions. ACM Inroads 1 (3), 66–78.

OMDL, 2012. Open Mashup Description Language. Retrieved from Open Mashup Description Language: <omdl.org/>.

OpenSocial, 2012. OpenSocial Specification. Retrieved from OpenSocial | OpenSocial Foundation: <http://opensocial.org/>.

Patton, J., 2009. User Story Mapping. Retrieved from User Story Mapping. <http://www.agileproductdesign.com/presentations/user_story_mapping>.

Ramlall, S., 2004. A review of employee motivation theories and their implications for employee retention within organizations. J. Am. Acad. Bus. 5 (1/2), 52–63.

Reason, J., Andersen, H., 1991. Errors in a team context. Draft Paper for Mohawk Stresa Workshop.

Robles, G., Gonzalez-Barahona, J., 2006. Geographic location of developers at SourceForge. Proceedings of the 2006 International Workshop on Mining Software Repositories, pp. 144–150.

Rodriguez, J.P., Ebert, C., Vizcaino, A., 2010. Technologies and tools for distributed teams. Softw. IEEE 27 (5), 10–14.

Satzger, B., Psaier, H., Schall, D., Dustdar, S., 2013. Auction-based crowdsourcing supporting skill management. Information Systems Journal (IS), Elsevier 38 (4), 547–560.

Satzger, B., Psaier, H., Schall, D., Dustdar, S., 2011. Stimulating skill evolution in market-based crowdsourcing. 9th International Conference on Business Process Management (BPM ′11). Springer, New York, pp. 66–82.

Schall, D., Satzger, B., Psaier, H., 2012. Crowdsourcing tasks to social networks in BPEL4People. World Wide Web. Springer.

Sengupta, B., Chandra, S., Sinha, V., 2006. A research agenda for distributed software development. Proceedings of the 28th International Conference on Software Engineering, pp. 731–740.

SourceForge, 2012. SourceForge—Download, Develop and Publish Free Open Source Software. Retrieved from SourceForge: <http://sourceforge.net/>.

Spiegel, D., 2012. Frei schwebend in der Wolke. Frei schwebend in der Wolke.

StackExchange, 2012. Stack Overflow. Retrieved from Stack Overflow: <http://stackoverflow.com/>.

Telerik, 2012. TeamPulse—Telerik. Retrieved from TeamPulse: <www.telerik.com/agile-project-management-tools>.

TopCoder, I., 2012. Home of the World's Largest Development Community. Retrieved from TopCoder: <http://www.topcoder.com/>.

uTest, 2012. Software Testing | uTest. Retrieved from uTest: <http://www.utest.com/>.

VersionOne, 2012. Agile Project Management Software, Agile Tools, Scrum Tools, Agile Software, Scrum Software | VersionOne. Retrieved from VersionOne: <http://www.versionone.com/>.

von Ahn, L., 2006. Games with a purpose. Computer 39 (6), 92–94.

W3C, 2012. Packaged Web Apps (Widgets) - Packaging and XML Configuration, second ed. Retrieved from W3C Widgets Specification: <http://www.w3.org/TR/widgets/>.

W3C. 2005. WebID 1.0 - W3C. Retrieved from WebID 1.0: <www.w3.org/2005/Incubator/webid/spec/>.

Walsh, R., 2009. The web startup success guide. Apress.

Xing, 2012. XING - The professional network | XING. Retrieved from XING: <http://www.xing.com/>.

Zoho, 2012. Zoho Office Suite. Retrieved from Zoho Office Suite: <http://www.zoho.com/>.