Article Pre-Print
Please cite as: K Stol and B Fitzgerald (2014) Two's Company, Three's a Crowd: A Case Study of Crowdsourcing Software Development, in: Proceedings of the 36th International Conference on Software Engineering, Hyderabad, India.

Two's Company, Three's a Crowd: A Case Study of Crowdsourcing Software Development

Klaas-Jan Stol
Lero—The Irish Software Engineering Research Centre, University of Limerick, Ireland
klaas-jan.stol@lero.ie

Brian Fitzgerald
Lero—The Irish Software Engineering Research Centre, University of Limerick, Ireland
bf@ul.ie

ABSTRACT
Crowdsourcing is an emerging and promising approach which involves delegating a variety of tasks to an unknown workforce—the crowd. Crowdsourcing has been applied quite successfully in various contexts from basic tasks on Amazon Mechanical Turk to solving complex industry problems, e.g. InnoCentive. Companies are increasingly using crowdsourcing to accomplish specific software development tasks. However, very little research exists on this specific topic. This paper presents an in-depth industry case study of crowdsourcing software development at a multinational corporation. Our case study highlights a number of challenges that arise when crowdsourcing software development. For example, the crowdsourcing development process is essentially a waterfall model and this must eventually be integrated with the agile approach used by the company. Crowdsourcing works better for specific software development tasks that are less complex and stand-alone without interdependencies. The development cost was much greater than originally expected, overhead in terms of company effort to prepare specifications and answer crowdsourcing community queries was much greater, and the time-scale to complete contests, review submissions and resolve quality issues was significant. Finally, quality issues were pushed later in the lifecycle given the lengthy process necessary to identify and resolve quality issues. Given the emphasis in software engineering on identifying bugs as early as possible, this is quite problematic.

Categories and Subject Descriptors
K.6.3 [**Software Management**]: [Software development, Software process] D.2 [**Software Engineering**]: Management—*Programming teams*; K.4.3 [**Organizational Impacts**]: [Computer-supported collaborative work]

General Terms
Human Factors, Management

Keywords
Crowdsourcing, case study, challenges, software development

1. INTRODUCTION
Software engineering no longer takes place in small, isolated groups of developers, but increasingly takes place in organizations and communities involving many people [7, 79]. There is an increasing trend towards globalization with a focus on collaborative methods and infrastructure [10]. One emerging approach to getting work done is *crowdsourcing*, a sourcing strategy that emerged in the 1990s [35]. Driven by Web 2.0 technologies [16, 71], organizations can tap into a workforce consisting of anyone with an Internet connection. Customers, or *requesters*, can advertise chunks of work, or tasks, on a crowdsourcing platform, where suppliers (i.e., individual workers) select those tasks that match their interests and abilities [39].

Crowdsourcing has been adopted in a wide variety of domains, such as design and sales of T-shirts [43] and pharmaceutical research and development [56], and there are numerous crowdsourcing platforms through which customers and suppliers can find each other [23]. One of the best known crowdsourcing platforms is Amazon Mechanical Turk (AMT) [44]. On AMT, chunks of work are referred to as *Human Intelligence Tasks* (HIT) or *micro-tasks*. Typical micro-tasks are characterized as self-contained, simple, repetitive, short, requiring little time, cognitive effort and specialized skills. Crowdsourcing has worked particularly well for such tasks [50, 52]. Examples include tagging images, and translating fragments of text. As a result, remuneration of work is typically in the order of a few cents to a few US dollars [44].

In contrast to micro-tasks, software development tasks are often interdependent, complex, heterogeneous, and can require significant periods of time, cognitive effort and various types of expertise [51]. Yet, there are cases of crowdsourcing complex tasks; for instance, InnoCentive deal with problem solving and innovation projects, which may yield payments of thousands of US dollars [43].

A number of potential benefits have been linked to the use of crowdsourcing in general, and these would also be applicable in the context of software development specifically:

- *Cost reduction* [47, 50, 70] through lower development costs for developers in certain regions, and also through the avoidance of the extra cost overheads typically incurred in hiring developers;
- *Faster time-to-market* [50, 55, 69] through accessing a critical mass of necessary technical talent who can achieve follow-the-sun development across time zones, as well as parallel development on decomposed tasks, and who are typically willing to work at weekends, for example;
- *Higher quality* through broad participation [12, 70, 82]: the ability to get access to a broad and deep pool of development talent who self-select on the basis that they have the necessary expertise, and who then participate in contests where the highest quality 'winning' solution is chosen.
- *Creativity and open innovation* [13, 25, 26, 55, 70, 77]: there are many examples of "wisdom of crowds" creativity

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICSE'14, May 31 – June 7, 2014, Hyderabad, India.
Copyright 2014 ACM 978-1-4503-2756-5/14/05... $15.00.

whereby the variety of expertise available ensures that more creative solutions can be explored, which often elude the fixed mindset that can exist within individual companies, a phenomenon known as "near-field repurposing of knowledge" [82].

Given that the first three benefits above (cost, time and quality) directly address the three central problematic areas of the so-called "software crisis" [33], it is not surprising that a number of authors have argued that crowdsourcing may become a common approach to software development [8, 48]. The fourth benefit, that of tapping into the creative capacity of a crowd is captured well in a quote attributed to Sun Microsystems co-founder Bill Joy, namely that, "*No matter who you are, most of the smartest people work for someone else*" [56]. As Lakhani and Panetta [56] pointed out, completing knowledge-intensive tasks will become increasingly challenging in traditional closed models of proprietary innovation, if most of the knowledge exists outside an organization.

Research on crowdsourcing tends to focus on one of three perspectives: the worker (supplier) perspective, the system (crowdsourcing platform, e.g., AMT) perspective, and the requester (customer) perspective [88]. Studies of crowdsourcing software development, or what LaToza et al. [57] referred to as "Crowd Development," are scarce [59].

Similar to the confusion surrounding the term 'crowdsourcing' in general [16, 28, 67, 74], there is some confusion about what constitutes crowdsourcing in a software development context. In particular, crowdsourcing may be positioned as closely related to other strategies such as outsourcing [37] and opensourcing [1, 63]. For instance, open source is often cited as the 'genesis' of crowdsourcing [43, p.8, 48, 57], but others argue that open source is *not* a form of crowdsourcing [16]. Other terms that have been used as synonyms are 'peer production' [30, 40] and 'commons-based peer production' [48], both referring to the idea that software is developed by a group of peers. While these strategies are similar in some respects, there are significant differences (see also Section 2) that set crowdsourcing apart [78].

Furthermore, most studies aim to explain crowdsourcing by describing successful cases (e.g., [14]); as a result, there has been little attention to the challenges that may arise. Further research is needed to better understand the limits of crowdsourcing software development. This paper presents an in-depth industry case study of crowdsourcing software development at a multinational corporation. The goal of this study is to shed light on the key issues in crowdsourcing that are relevant to software development. Crowdsourcing is a multi-disciplinary research topic [15, 46], and to date very few studies exist in the software engineering domain [57, 59]. Studies to date have focused on contestants [3] and crowdsourcing platforms/companies [55], but have not investigated the perspective of a crowdsourcing *customer*, that is, an organization that uses a crowdsourcing platform to get software development work done. The study reveals a number of challenges that the case study organization encountered.

The remainder of this paper is structured as follows. Section 2 presents background on crowdsourcing, defines crowdsourcing for software development and identifies a number of key themes from the crowdsourcing literature that are of particular importance in a software development context. These themes provide an analytical framework for our study. Section 3 outlines our empirical research approach. Section 4 presents the results of our study. Section 5 discusses the key findings of our study and outlines a number of directions for future work.

2. BACKGROUND AND RELATED WORK

There are a number of crowdsourcing platforms specifically targeting software development (and related tasks such as testing [62, 83]). The largest is TopCoder [80], which has a community of over 500,000 developers. Other platforms include AppStori [2], and uTest [84], though the actual mechanism of matching customers and suppliers varies. Given the lack of clarity of what crowdsourcing means in a software development context, in particular in relation to outsourcing and opensourcing, we present a definition in Section 2.1. Section 2.2 identifies a set of key concerns in crowdsourcing that are specific to software development, and which provide a framework for our empirical study in Section 4.

2.1 Defining Crowdsourcing Software Development

There are numerous definitions for the term 'crowdsourcing' [28, 38]. Howe presented the following definition [42]:

Crowdsourcing is the act of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined, generally large group of people in the form of an open call.

A second definition offered by Howe, referred to as the 'Soundbyte' version, defines crowdsourcing as the "*application of Open Source principles to fields outside of software*" [41]. Both definitions are ambiguous in the context of software development. The phrase "*outsourcing [...] to an undefined, generally large group of people*" also applies to the concept of opensourcing [1], and some authors consider this a form of crowdsourcing [63]. Others argue that crowdsourcing differs from open source (and thus, opensourcing), in that the latter is a public good, whereas the former is focused on extracting economic value [3]. Brabham, however, identified a more significant distinction between open source and crowdsourcing, in that the locus of control in the former is essentially with the crowd, and there is no overarching entity that coordinates the overall effort [16]. Open source projects tend to be self-organizing, and while a core team can set out a roadmap, there is no 'control' in that roadmap tasks are assigned to the project's community. (For the same reason, Brabham argued that Wikipedia is not an example of crowdsourcing, since all articles are offered on the initiative of the articles' authors.) Even though, many open source projects are moving more towards formal organization [32], the locus of control remains largely with the crowd/open source community.

The incentive-based contests in which crowdsourcing typically takes place is also problematic in the case of crowdsourcing software development. While TopCoder, the primary platform for crowdsourcing software development operate on the basis of competitions, another platform, uTest, performs software testing using a global community of over 80,000 testers but does not use competitions. Another characterization is that crowdsourcing is "*outsourcing on steroids*" [43, p.46]. This suggests that crowdsourcing is merely a form of outsourcing. However, the duplication of work being performed in parallel does not apply to outsourcing. Overall, the key elements of crowdsourcing software development appear to be the following:

- *Nature of the work*, i.e. software development tasks which are inherently quite complex with many interdependencies, and not equivalent to the simple HITs performed on AMT, for example;
- *Locus of control*, i.e. the customer organization who has to specify the tasks and integrate the resulting output into the

organization's software development process, and who own the output;
- *Nature of the workforce*, i.e. a large and typically undefined group of external people, but with the requisite 'wide and deep' specialized knowledge to accomplish the task successfully.

Based on these, our definition of crowdsourced software development is the following:

The accomplishment of specified software development tasks on behalf of an organization by a large and typically undefined group of external people with the requisite specialist knowledge through an open call.

2.2 Key Concerns in Crowdsourcing Software Development

Given the nascent state of research on crowdsourcing software development, there is no commonly agreed upon framework that captures the key concerns of this topic [27, 74]. A framework can help to define the boundaries of a research area [72]. Drawing on the general literature on crowdsourcing, we synthesized a set of six key concerns which have particular relevance in a software development context: (1) Task Decomposition, (2) Coordination and Communication, (3) Planning and Scheduling, (4) Quality Assurance, (5) Knowledge and Intellectual Property, and (6) Motivation and Remuneration. The remainder of Section 2.2 presents the six themes in detail.

2.2.1 Task Decomposition

A key issue in crowdsourcing is that work is decomposed into a set of smaller tasks [45, 52, 54]. This issue is highly relevant in outsourcing scenarios, and Herbsleb and Grinter [36] reminded us of Parnas' definition of a module as "*a responsibility assignment rather than a subprogram*" [64]. What is of particular importance, given the interdependencies in software, is that different developers working on a project know how their code fits into the resulting software product, in terms of understanding interfaces and assumptions made.

Whereas in general-purpose crowdsourcing markets, such as AMT, tasks are typically small and independent [44], software development tasks are more complex and interdependent. Therefore, a key challenge is to find an appropriate decomposition of the software product into tasks that can be effectively crowdsourced [57]. Kulkarni et al. [54] termed this challenge the "*workflow design problem.*" More efficient decompositions can lead to an increased parallelism [57].

Furthermore, in decomposing a software project, there is a fine balance between providing a sufficiently detailed specification for the task being crowdsourced on the one hand, and stifling innovation with overly detailed specifications on the other hand [55]. Tajedin and Nevo [78] suggested that projects which can be decomposed into small modules with clear requirements and limited interdependencies are more likely to succeed.

2.2.2 Coordination and Communication

When crowdsourcing more complex tasks, as is the case in software development, there is a need for coordination [51]. Malone and Crowston [58] defined coordination as "*the process of managing dependencies among activities*." As such, coordination is concerned with directing efforts of individuals toward a common and explicitly recognized goal, and linking different parts of an organization together to achieve a set of tasks [53]. Although related to task decomposition discussed above, coordination is specifically concerned with communication, interdependencies and integrating various parts into a whole [53, 57].

The above characterization of coordination seems to assume that activities are conducted *within* an organization. Clearly, in crowdsourcing, participants who submit 'solutions' are not part of the crowdsourcing organization. In fact, interdependent tasks may be performed by different workers, potentially causing incompatibilities between the solutions provided [57].

In a software engineering context, the need for different developers to communicate is often related to Brooks' Law ("*adding manpower to a late software project makes it later*"), in that the more people are involved, the higher the communication overhead is [17]. Whether or not this applies in a crowdsourcing context depends on whether the work is done in a collaborative or competitive fashion [88]. For instance, TopCoder, the largest crowdsourcing platform for software development, organizes tasks as competitions; a winner (and runner-up) is selected based on a peer-review of the submissions by the community [7].

2.2.3 Planning and Scheduling

With crowdsourcing, one or more tasks are given to an unknown workforce to complete, and as a result an organization is letting go of control of that particular work. On the one hand, this may result in a timely delivery of completed work as it can be completed in parallel and independently of the organization's in-house workforce, and in particular if the tasks are competitions where payment depends on timely delivery. On the other hand, however, this introduces a level of uncertainty as to whether or not the work will be completed on time [88]. One of the promises of crowdsourcing is to shorten the product development cycle [14, 85]. In order to achieve this, it is important that the desired schedule of a crowdsourcing organization can be adhered to by the crowd. For instance, a core challenge is to ensure that sufficient workers are available when needed [51]. While there may be extensive expertise within the crowd, very specific domain knowledge may not always be available at the moment it is needed. Furthermore, it is important to ensure that sufficient time is given to coders, relating the issue of planning to the size and scope of a task. Lakhani et al. [55] reported that TopCoder "*community members worked best when contests lasted less than two weeks.*" Too large or long-lasting projects could result in decreased interest from the community, and thus fewer submissions.

2.2.4 Quality Assurance

Another claim made by crowdsourcing advocates is that the quality of submissions is high [12, 70, 82]. At the same time, there is a risk of 'noise' in submissions, where solutions are of a low quality [24, 45]. In a software development context, the idea that input from a wide variety of developers helps in finding and fixing defects is better known as Linus's Law, or, "*given enough eyeballs, all bugs are shallow*" [66]. Closely related to this is the idea that there is a wide variety of expertise within a developer community. The challenge lies in attracting sufficient contestants, under the assumption that given enough contestants, the required expertise will be present. Whereas AMT is non-transparent, in that contestants do not know how many 'competitors' there are for a certain competition, a platform such as TopCoder is fully transparent. Prior to participating, contestants must register for a certain competition. Findings from a recent study suggest, however, that the greater the number of contest participants, the lower the quality of the work [49]. One characteristic sometimes ascribed to the crowd is that it consists mostly of amateurs [71], thus suggesting that the resulting quality of output may not be on

par with professional work. However, Brabham points out that this is a myth [15].

Quality assurance is a key concern in software development, whether the software is developed in-house or by external parties. Of particular concern in crowdsourcing is that a customer has no knowledge of the developers that deliver the software, nor of the process that they might follow, and therefore has no control over these aspects. Crowd developers may "*satisfice, minimizing the amount of effort they expend*" [51]. Also, there can be disagreement about a solution; Kittur [50] distinguished 'subjective' tasks for which there is no single right answer, and 'objective' tasks that can be easily verified. While software either fulfills a set of requirements or not, disagreements may still arise regarding certain functionality or the scope of a task. Furthermore, quality attributes of submissions, such as performance and maintainability of the code may still vary. One approach to quality control is peer-review. At TopCoder, for instance, members of the community perform peer-reviews of the submitted software. Similar to peer-reviews in open source, such reviews are "*truly independent*" [31] given that the peer-reviewers would usually not know the creator of the work, and would therefore be unlikely to be either positively or negatively biased. A certain level of 'shepherding' the crowd has also been suggested to improve quality [24, 54]. Kulkarni et al. [54] found that letting the crowd plan amongst themselves without supervision of a requester was partially successful, but that intervention by a requester during the workflow could improve quality significantly.

2.2.5 Knowledge and Intellectual Property

Knowledge management has long been recognized to be an important topic within the software engineering field [4, 9, 21]. A key difference with traditional outsourcing is that there is no single supplier that develops an in-depth understanding of the problem domain of a crowdsourced project; rather, the continuous turnover of workers is an inherent characteristic of crowdsourcing [20].

One type of knowledge of particular concern in crowdsourcing software development tasks is that of knowledge and intellectual property (IP) [55, 86]. IP 'leakage' and the consequent loss of competitive advantage is a challenge in adopting crowdsourcing [26]. Organizations may be hesitant to provide too many details on a certain task (i.e., module or component) that is crowdsourced, yet sufficient detail in the specification is necessary for developers in the crowd to understand what the crowdsourcing organization is requesting. Another issue that may arise is ownership of inventions [18, 46]. Tasks on general-purpose platforms such as AMT are arguably relatively simple (requiring little human intelligence), and thus IP concerns do not loom large. Software development, however, is a highly creative process, and organizations will want to ensure they can patent any potential inventions that emerge with no confusion in relation to ownership. A third issue can arise when workers submit solutions that are not theirs [46], for instance, if the solution contains open source code with the restrictive GNU Public License (GPL) license. This may be a risk for crowdsourcing customers as it affects their product.

2.2.6 Motivation and Remuneration

A final consideration in crowdsourcing is that of motivation and remuneration [19, 22, 29, 40, 45, 59, 60, 65]. Motivation is a topic that has received considerable attention in the software engineering research field, given that it is reported to be a major factor in project success [6, 11]. Motivational factors can be external or intrinsic. Extrinsic factors are conditions surrounding a job [5], whereas intrinsic factors relate to the job itself (e.g., having fun, gaining recognition and a sense of achievement). Obviously, the compensation of a certain crowdsourcing task will depend heavily on the expected duration and the complexity of the task. Tasks can vary in complexity, from so-called 'micro-tasks,' such as tagging an image which takes only seconds, to more time-consuming tasks such as transcribing audio. Clearly, software development tasks are complex and time-consuming, and contestants will expect significant remuneration, as opposed to the average cost of micro-tasks on AMT, most of which are below one US dollar [44]. One claimed benefit of crowdsourcing is that it can greatly reduce cost [55]. Yet, determining an appropriate price is a key challenge for crowdsourcing in general [29, 75], and also for software development specifically [55, 59].

3. RESEARCH SETTING AND METHODS

In this section we describe the research approach adopted in this study. Section 3.1 presents the background of the case study organization and the TopCoder crowdsourcing platform. Section 3.2 describes the research method and data collection and analysis.

3.1 Background of the Case

3.1.1 TechPlatform Inc.

TechPlatform Inc. (TPI - a pseudonym) is a global player offering services and solutions in the cloud. The company employs several tens of thousands of people worldwide, with 400 sales offices, and partners in more than 75 countries. In 2012, TPI sought to investigate the use of crowdsourcing in its software development function at the instigation of a senior executive.

3.1.2 TopCoder

The platform through which TPI is crowdsourcing its software development is TopCoder (TC). TC is the largest software development crowdsourcing platform and its community has grown more than ten-fold, from 50,000 to 612,000 members between 2004 and 2014. However, a recent estimate suggests that only 0.7% of registered members had participated in development.

TC has an extremely impressive customer list of blue chip companies. In promoting their services, TC suggests that customers can "*Try more often. Succeed more often. Spend Less.*" TC offers a platform which facilitates what is termed the three pillars of Digital Creation: (1) front-end innovation; (2) software development, and (3) algorithms and analytics. For this study, we focus on the software development pillar.

TC accomplishes software development tasks for customers through a series of competitions. The TC community breaks down customer projects into atomized units of work that comprise the entire build, and these work units are accomplished through competitive contests, whereby the TC community compete and submit solutions. The TC community is structured into Program Managers who oversee customer projects and choose co-pilots within the TC community to act as an interface between customers and TC developers, and to help choose winners for the various contests.

Co-pilots are experienced TC community members who have proven themselves in the past on the TC platform. They manage the technical aspects of crafting and running competitions through to successful delivery. TC suggests that the co-pilots can do the technical heavy lifting and process management, allowing the customer to be the "*conductor of a world-wide talent pool*" [81].

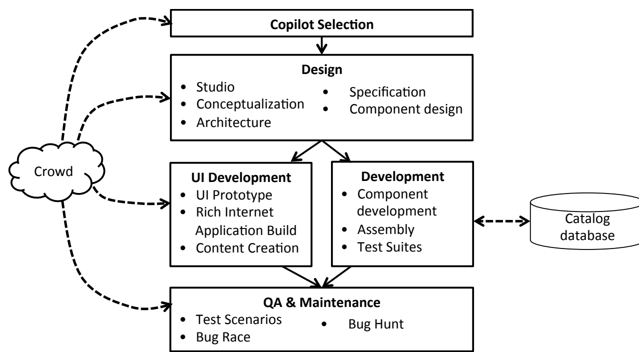

Figure 1. TopCoder competition types and phases (adapted from Mao et al. [59] and TopCoder.com).

The TC software development methodology comprises a number of different competition types, organized in a number of categories, as illustrated in Figure 1.

3.2 Methods and Analysis

The goal of our study was to investigate crowdsourcing in a software development context from a crowdsourcing customer perspective, to better understand this process and the challenges associated with it. To that end, we conducted an in-depth case study at the case company. Case study research is particularly suited to study real-world phenomena that cannot be studied separately from their context [87]. Case study research has become increasingly popular as a method in software engineering research [68], as it provides rich insights into contemporary phenomena (e.g., distributed development [36], open source software development [61]). For this study we conducted a number of face-to-face, semi-structured interviews with key informants who were involved with the TC crowdsourcing initiative. These included the Divisional CTO at the visited location, a software architect, a software development manager, a program manager and a project manager. Prior to the study, we developed an interview guide that was based on the crowdsourcing themes discussed in Section 2. The face-to-face interviews were conducted during three half-day workshops on the premises of the company. In addition, we conducted two teleconference interviews each involving two TPI staff members who played key roles in the crowdsourcing process. Interview sessions lasted between one and two hours each. During the research process, we sent several early drafts of this paper to key participants of the study—a form of member checking [68], and this also provided opportunities to seek clarifications when necessary. Data were analyzed using qualitative methods as described by Seaman [73]. All interviews were transcribed, resulting in 112 pages of text. The analysis consisted of coding the transcripts using the six themes identified in Section 2.2 as seed categories. The transcripts were analyzed in parallel by both authors and several analytical memos were written. The memos established an audit trail of the analysis, and facilitated a process of peer debriefing for the researchers. Besides drawing from the interview data, we also drew from a number of internal documents prepared by the company, which facilitated a process of triangulation among data sources. Other sources included documentation on the crowdsourcing schedules, project documentation that TPI stored on an internal wiki, and contest information drawn from the TopCoder website. Further details of the design and execution of our study are described in our study protocol [76].

4. CROWDSOURCING AT TPI

The application which TPI selected for crowdsourcing was Titan, a web application to be used by TPI field engineers when migrating from one platform to another as part of a customer engagement. Within TPI a technical decision was taken that future development should use HTML5, and this was the technology chosen for the front end, which was replacing the desktop application. The back-end services were based on a similar technology set used by the previous desktop-based solution. Thus, TPI were keen to leverage HTML5 expertise from the large global TC community. Figure 2 illustrates the breakdown of the development work in terms of what was to be done by TPI, and what was to be done by TopCoder. It should be noted that the dimensions of the figure do not reflect the actual amount of work. Given that a lot of TPI domain-specific knowledge is required for back-end development, this is retained as part of the TPI development responsibility.

Similarly in the front-end, topics such as migration planning, importing and the scripting engine were retained for development by TPI. The two activities that are part of the TC crowdsourced development are asset modeling and automation testing. Modeling refers to the arrays and switches that need to be migrated and thus have to be modeled (i.e. created and configured) in the Titan application. Automation testing complements unit and integration testing which is designed by TPI developers, and refers to the testing designed by QA to test the front-end GUI interaction with the back-end. As can be seen in Figure 2, this development activity will be carried out almost entirely by TC. The small portion that will be developed by TPI involves a “Gold Standard” which will be made available subsequently as a template for the TC community to indicate how TPI would like automation testing to be done. The following sub-sections draw on the framework in Section 2.2 to discuss the TC crowdsourcing development for the TPI web application.

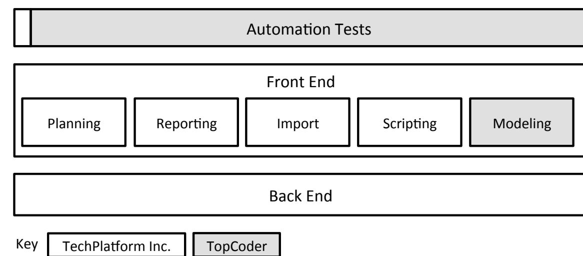

Figure 2. Work decomposition between TPI and TC.

4.1 Task Decomposition

The choice as to what parts of the product were appropriate for crowdsourcing was not entirely trivial for TPI. Code and executables which were self-contained would be easier to merge and hence were more suitable for crowdsourcing. However, if code from TC had to be directly merged with code being developed in-house, this would be more problematic. The decision as to what work to crowdsource was primarily based on internal resources (or lack thereof) and the amount of domain knowledge required for a certain task. Tasks that required the least amount of domain knowledge were deemed most suitable.

Table 1. Titan development phases and specifications.

Phase		Panels	Documents	Pages
1	Dashboards	40	NA	NA
2	Flagship product I	18	15	196
3	Flagship product II	33	19	543
4	Network devices	14	11	161
5	Legacy and third-party	23	17	131

TPI divided the project into five development phases, listed in Table 1. The first dashboards phase was the front-end which involved the high-level dashboard interface pages, e.g., for customer creation, project creation and navigation. The next two development phases involved configuration of TPI's flagship product. Following this, Phase 4 was concerned with the various network devices which also form part of the migration configuration. Finally, Phase 5 dealt with the low-end legacy products and various third party solutions that also need to be migrated. In order to minimize the modifications that would need to be made to the TC code after delivery, TPI made the header and footer browser code available to TC developers. This was to ensure this standard format would be maintained by all TC developers. For the Titan application, TPI's policy was to only use HTML5 where a feature was supported by all platforms to increase portability. Initially, there was an expectation that the TC community would deliver some innovative HTML5 code. However, the TPI requirement that HTML5 features would have to be supported by all browser platforms resulted in a very small proportion of all potential HTML5 features being available for use by TC developers. The expected innovation from the "crowd" was thus precluded by the TPI specification.

In order to minimize integration effort later on, the architect had wanted to let TC developers work against a real back-end core as opposed to stub services. However, by the time development with TC started, the core was not ready and stubs were used during most development contests. Consequently, this integration effort was pushed back to a later stage in the development process, which was not ideal.

For traditional in-house development, TPI developers had internalized a great deal of information in relation to coding standards and templates, and technical specifications. However, many of the coding standards and templates were documented informally and not stored centrally on the internal wiki installation. This scattering of information and URLs prevented it from being packaged as a deliverable for TC developers. A great deal of extra work was necessary to ensure that this information was made explicit in the requirements specification for the external TC developers. Most of the effort was related to the technical specifications. Table 2 lists the number of documents and the total number of pages of specifications written for each of the five phases defined by TPI. The architect liaising with TC described the situation as follows:

"It feels like we've produced a million specification documents, but obviously we haven't. The way we do specifications for TopCoder is entirely different to how we do them internally."

4.2 Coordination and Communication

From the TC perspective, the software development process consists of a number of interrelated phases (see Figure 1 above). While the TC process is essentially a waterfall one, an agile development process, based on Scrum, was in use at TPI. Synthesizing these different development processes was problematic. TC development had to be assigned to a Scrum team within TPI, and TC contributions needed to be subsequently injected into the appropriate sprints. The architect summarized the central problem as follows:

"We are an agile shop and we are used to changing our minds. This can be a problem with TC when we tell them one thing in one contest, but have changed our mind in the next contest."

There were also quite a number of layers in the engagement model between TC and TPI. Firstly at the TC end, a co-pilot liaised between the TC developer community on the one hand, and TPI personnel on the other hand. Furthermore, a platform specialist and the TPI account manager were involved, effectively overseeing the co-pilot and recommending changes at that level. In this case, following some problems, a new co-pilot was selected with a tendency to be more proactive than his predecessor.

Within TPI, the choice of personnel to interact with the TC co-pilot was a difficult decision. While TC would prefer a single point of contact within the customer organization, there were significant management and technical issues involved, thus requiring senior people from TPI on both the management and technical end. A senior *TC program manager* was appointed specifically for all programs being developed with TC. This manager ensured that management were aware of any scheduling issues that could arise, for example, and also ensured that training was provided. However, there was also a specific *Titan program manager*, and thus there was inevitably some overlap between both roles. On the technical side, a senior architect was allocated to coordinate the TC development for the Titan project. This role of *TC liaison* which had daily contact with the TC community was considered to be problematic within TPI, given the considerable pressure to answer questions which was also very time consuming. There was some concern within TPI about allocating such a senior resource to this liaison role given the significant cost. The Software Development Manager described the situation from a resource allocation perspective:

"To have a single point of contact for the project on our side, the contact needs to have both technical skills and project management skills to be able to manage the requirements, competitions and questions from TopCoder technical community members. It used a very valuable resource and in this project they had to use up some time from other developers to address all the questions coming back from TopCoder."

At the initial stage, this liaison role involved answering questions on the TC Forums. There was significant time pressure involved since a time penalty applied if forum questions were not answered in a timely fashion by TPI, which would mean that the original committed delivery date for TC development would be pushed out. Also, the architect estimated the time answering questions on the TC Forums to be at least twice as long as would be the case with internal development:

"There are a lot more questions than with internal development. However, there is no informal communication mechanism. You cannot yell at the person in the next cubicle and get the answer very quickly."

In contrast to distributed development which typically involves other developers from the same organization, the only relationship which tended to build over time was that with the TC co-pilot. There was no real opportunity to build up a relationship with any of the TC developers, as interaction was filtered through a number of layers. Another structural coordination issue arose in that TPI allocate architects to products, and the desire to get the TC project

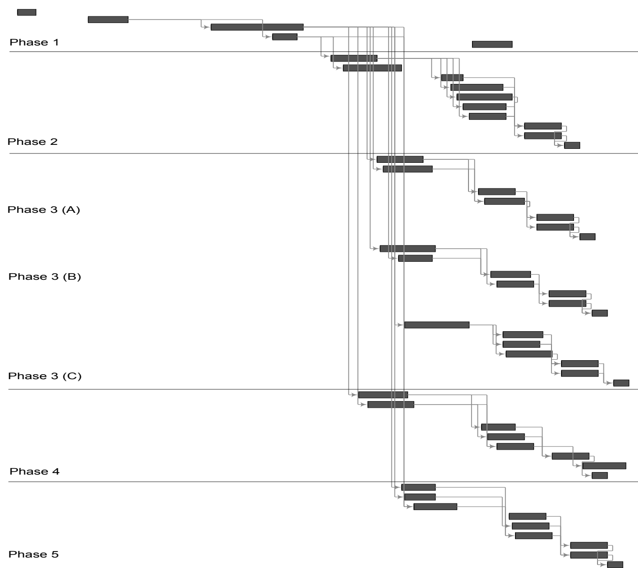

Figure 3. Gantt chart of TC contests.

completed resulted in two additional architects working on the project. This was seen as a sub-optimal resource allocation, given that the architect role was a somewhat scarce and extremely valuable resource.

TPI also had a so-called "tactical" Scrum team that could be assigned to different tasks more flexibly in that they were not formally assigned to projects on a long-term basis, as was the case with the normal Scrum teams at TPI. This tactical team could deal with TC contributions when they arrived. However, in some cases a normal Scrum team would also be assigned to the project, and in these cases involvement of the tactical Scrum team would not then be necessary. Overall, there was extra overhead and duplication of work on the project in that two teams had to become familiar with the project and deliverables. These two teams also had to communicate with each other. To address this issue, TPI dropped the use of the tactical team, and instead scheduled time in the project sprints to integrate the deliveries from TC.

4.3 Planning and Scheduling

The Titan project comprised more than fifty TC competitions. These competitions involved a total of 695 contest days, with an average length of competition of just over 13 days.[1] The shortest completion time for a competition was 4 days while the longest competition took 32 days to complete. As discussed above, TPI had structured the overall development of the Titan product into five phases. The average duration across these development phases is 80 days, with the longest development duration (90 days) for the front-end HTML5 panels in the first phase, and the shortest development duration (69 days) for the final phase involving the low-end legacy and third-party arrays. Table 2 lists the duration of each phase, the number of competitions per phase, and the average length of a competition per phase. (Note that competitions overlap in practice, so that a phase's duration is not merely the product of the number of competitions and their length).

Figure 3 presents a Gantt diagram that shows the planning of all contests. The figure shows the dependencies between the various contests, which are of varying types (see Figure 1). For instance, assembly contests must be completed before any test suite contests can start. This dependency corresponds to a waterfall process.

Some of the specific timings and the granularity of possible decisions for TC development were somewhat problematic for TPI. For example, TC allows a customer five days to accept or reject a deliverable. According to the architect, this was often not long enough to analyze and fully test the deliverable, and it was difficult to get these reviews done in time internally. A further difficulty arose in that TC deliverables must be accepted as a whole, or rejected as a whole, with no middle ground. It would be better from TPI's point of view if more flexible granularity was possible in that certain parts of deliverables could be accepted and partial payment made for these acceptable parts. Because TPI did not want to deter TC developers from bidding on future competitions, there was a tendency to accept code, even with some defects. There was an additional warranty period of 30 days, but integrating fixes under this warranty would pose considerable overhead in receiving, checking and integrating new code with an active code base which would more than likely have undergone significant further modification internally within TPI in the interim. Furthermore, when issues were escalated within the 30-day warranty, the resolutions were generally not satisfactory to TPI. Overall, a single longer initial acceptance period of 15 days would probably be more beneficial to TPI than the two current periods of five and 30 days, respectively. Another issue related to planning and scheduling arose when TPI had to wait for a contest to finish, while the main application was evolving, causing possible integration issues. TPI's schedule was also jeopardized by several contests failing due to a lack of submissions. These contests had to be rescheduled thus causing a delay in TPI's schedule. When rescheduled, there was only a single submission in one case, despite more than 30 registrants indicating an interest.

As already discussed, TPI perceived the need to run multiple competitions in parallel so as to shorten the development time, and therefore chose to run their development phases concurrently. However, this clearly had implications for coordination. For example, as can be seen in Figure 3, there were interdependencies between the products produced in the various development phases. This also led to duplication of functionality in the JSP and CSS code.

Table 2. Contest Duration per Phase.

Phase	No. contests	Avg. length	Phase duration
1	5	17.6	90
2	10	14.5	80
3	21	12.0	81
4	8	13.1	80
5	9	10.1	69

4.4 Quality Assurance

Much research in software engineering has focused on identifying and eliminating errors as early as possible in the development process, on the well established basis that errors cost exponentially more to rectify, the later they are found in the development cycle [11]. However, the structure of the TC development process made it difficult to preserve this, as it shifted

[1] These durations are calendar days and thus include weekend days. This is justifiable as the TC community tend to treat weekends as working days. If weekends are excluded, the total number of workdays is 548, the average is just under 10 workdays, minimum of 3 workdays and a maximum of 23 days.

Table 3. Raised, Resolved, Outstanding and Awaiting Issues.

Issue status	Number
Raised	506
- Resolved	- 367
- Outstanding	- 139

QA issues towards the back-end of the development process, after coding has been completed. As the Development Manager expressed it:

"Crowdsourcing focuses on requirements and relaxes the quality process at the onset of the project, so now all the emphasis on managing the quality comes at the QA cycles later in the project, and that tends to be more expensive

The number of defects identified was quite significant. Table 3 shows the number of issues raised, resolved and outstanding at the time of our study. While many issues were of a cosmetic nature, and therefore fairly trivial, the sheer volume of issues required considerable time and attention from developers within TPI. Furthermore, as more contests were finished and software delivered back to TPI, the rate of new issues was increasing as well. Figure 4 shows this trend over time, and suggests a growing pressure on TPI developers to address these issues.

There was also a problem with lack of continuity. TC developers do not remain idle at the end of competitions, and may thus not be free to continue with TPI development in subsequent tasks. In fact, TPI experienced problems with bugs which had previously been identified being re-introduced to code after it went back for further development with TC. Partly this was due to how TC developers used the source code control tool. This added to the critical perception expressed by the Divisional CTO, when he contrasted it with the investment one would be prepared to make when using remote development teams for development, in describing crowdsourcing as being "a fleeting relationship."

Given that the combination of technical and specific domain expertise was considered by TPI to be quite rare (based on experience in recruiting developers), TPI took some initiatives to improve the quality of crowdsourced contributions. For example, a virtual machine with a sample core application was made available as an image that could easily be downloaded and run. This was used by the TC development community both in development and as a final test or demonstrator for code they developed. Prior to this, TC code testing was done with stubbed-out service calls to the back-end, but there was a concern within TPI that TC code would not necessarily run smoothly when connected fully to the back-end. When the code for the initial HTML5 high-level panel applications was produced by TC, there were some quality issues, for instance, the same header was repeated in every file. TPI took this code and further developed it to a "Gold Standard," at the level required by TPI. This was delivered back to the TC community as a template for future development. This tactic was extended to prepare sample code for a web application that could act as a template for the TC community. This included a parent project object model (build script), source code compliant with all TPI code standards, unit and integration tests, automation tests, and instructions for deployment and setup.

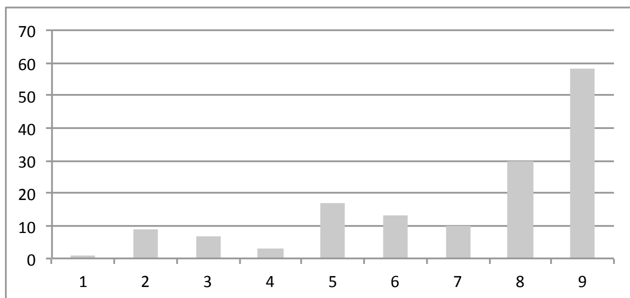

Figure 4. Trend of new issues raised (last 9 weeks).

4.5 Knowledge and Intellectual Property

The "fleeting relationship" mentioned earlier also has consequences for knowledge management and IP. According to the architect involved in the project, the lack of depth in the relationship with contestants meant that:

"there is a limited amount of carry-over knowledge. We will get a few contestants that will participate in multiple contests, but they won't build up domain knowledge in the way that an internal person would."

Also, given that there is no single supplier as would be the case in a traditional outsourcing scenario, any intellectual property relating to specifications and product knowledge is more widely exposed simply by virtue of its being viewed by the 'crowd' of potential developers. Table 4 shows the total number of registrants, and the total number of submissions per contest type (see Figure 1). The table shows that there were considerable numbers of potential participants (each of whom would have access to the contest specifications), but that the number of submissions was significantly lower – almost 90% of those registered for a contest did not actually submit anything to that contest. In other words, making detailed product and specification information available, which is necessary to achieve the benefit of tapping into the crowd's wisdom and creativity, seems (in this case) not to be as fruitful as one would hope given the limited numbers of submissions.

TPI chose a pseudonym to disguise their participation on the TC platform. This was to obfuscate the fact that the work was for the TPI platform as it was felt that developers from competing organizations might be working for TC in their spare time. TPI took advantage of the standard Competition Confidentiality Agreement (CCA) which TC use with their development community. TPI will not do business with certain countries, for example, and this can be policed through the CCA which identifies the home location of TC developers. TPI were still concerned about the extent to which proprietary information may be exposed in TC competitions. To address this, TPI plan to identify the "Secret Sauce" which should not be shared without very careful consideration. This would include the source code for the flagship and legacy applications, libraries and binaries from other TPI business units, performance calculation formulae, hardware specifications and business rules (e.g., Drools).

Table 4. Total number of registrants and submissions per contest type.

Type	Registrants	Submissions	%Sub/Reg
Copilot	13	6	46%
Studio	34	7	21%
Architecture	90	12	13%
Assembly	476	36	8%
Test Suite	8	1	13%
UI Prototype	99	22	22%
Total	**720**	**84**	**12%**

4.6 Motivation and Remuneration

Given a potential development community of a half million members, TC would claim to have broad and deep enough expertise to ensure a healthy competition rate. However, TPI have had to cancel some competitions because of a lack of participation and there had been a number of others with just a single contestant. The fact that TPI used a pseudonym does appear to be significant in that well known companies do attract TC developers more readily and TPI would certainly be a very well known company globally. The TC pricing structure was quite complex, and an overview of the cost so far for the Titan project is shown in Table 5 (all numbers rounded). At the top level, there was a monthly platform fee to TC. For TPI this was a monthly fee of $30,000. This allowed access to the TC component catalog containing more than 1,500 software solutions. TC estimates that approximately 60% of client projects can be solved through reusing components from this catalog. However, TPI were not in a position to leverage this catalog, since a lot of their IT product stack has already been developed, as the software development manager explained:

"We have our technology stack built and a lot of our software is already written for that. So the TopCoder catalog is not much use to us. There's no real bang for the buck for us there."

The co-pilot who was the principal liaison between TC and TPI typically cost $600 per contest. There was an initial specification review before the contest begins, and this cost $50. The individual contest pricing was also quite complex. In the case of TPI, first prizes for contests ranged from $200 up to $2,400, depending on the size and complexity of a contest. A second prize of 50% of the first prize was paid to the runner up in each contest, but this prize would *only* be paid if the quality rating of the submission were at least 75 (out of 100). If this score were less than 75, the runner-up would only receive Digital Run points (discussed below).

There was also a Reliability Bonus which was paid to the winning submission. The calculation of this bonus is quite detailed, but basically it can be up to 20% of the first prize, depending on the past successful track record of the winning contestant (i.e., his/her reliability – does a contestant actually submit after registering?). In addition, there was a cost of 45% of the first prize to support the TC Digital Run, an initiative whereby TC share money with the TC development community based on the monthly contest revenue and proportional to the number of points that TC developers have amassed in contests. The Digital Run is an additional mechanism to motivate potential contestants to participate even if they assess their chance of winning to be low. Following the contests, three reviewers from the TC community evaluated submissions and this cost approximately $800 on average. Finally, TC charged a 100% commission equal to the total development costs above. Overall, the total average cost per competition so far was approximately $6,200 (excluding the monthly platform fee).

In comparison with traditional development in-house, the Program Manager was of the opinion that TC development was less effective due to the lack of domain knowledge of the crowd and the indirect nature of the communication with developers. The primary reason for working with TC was the need to get development done more rapidly than would be possible with the existing level of internal resources.

However, given the planning and schedule statistics above, it is clear that the expectations in relation to a more rapid development time-frame were not fully realized.

Table 5. Overview of cost to date.

	Description	Average per contest
A	Monthly platform fee	$30,000[a]
B	Member prizes	
	- *First place prize*	$1,160
	- *Second place prize*[b]	$351
	- *Digital Run*[c]	$500
	- *Reliability Bonus*[d]	$176
	Review Board	
	- *Spec review*	$50
	- *Competition Review board*	$800
	Management	
	- *Co-pilot fees*	$600
C	Matching fees in B above to TC	100%

a. *Per month*
b. *Second prize is paid only if submission rating > 75.*
c. *Digital Run is 45% of the first prize; does not apply to bug hunts.*
d. *Reliability Bonus is up to 20% depending on winner's rating.*

5. DISCUSSION AND CONCLUSION

5.1 Discussion of Results

Crowdsourcing is an emerging topic and several benefits have were discussed in Section 1. Research suggests that crowdsourcing can be a viable option in a variety of situations, but very few studies so far have focused on crowdsourcing in a software development context.

The TopCoder crowdsourcing platform represents a significant 'market' of supply and demand for software development tasks. TopCoder claims many benefits can be achieved in terms of quality, cost, speed and flexibility [55]. However, the results of our study suggest that these benefits are not easy or automatic to realize. The TPI case identifies a number of significant challenges that the company had not foreseen prior to embarking on the crowdsourcing approach.

In relation to the basic issues of cost, time and quality, while we do not yet have a definitive direct comparison with a similar development project done in-house, it is certainly the case that the TPI development staff are not convinced that the TC model offers clear advantages in relation to cost, time and quality.

In all, 128 panels were designed, coded and tested, and although the work was not fully completed at time of writing, the estimated development cost paid directly to TC will be several hundred thousand US dollars, more than TPI expected to pay. Also, while the amount of work to be done by TC developers represented a significant part of the whole project, the complexity of the UI panels is arguably simple, in that it does not require significant business domain knowledge. Yet, TPI spent significant time and effort on writing specification documentation, much more so than if the software was being developed internally. This TPI internal effort has not been factored directly into the costs incurred, nor has any of the subsequent interaction and coordination effort of TPI personnel. The time-scale for this development work was of considerable magnitude, as shown in Table 2 and discussed in Section 4.3. However, it is particularly difficult to make precise effort estimates for a crowdsourced project: it is not possible to determine the actual effort spent by TC developers on a contest. There is a fixed end date for contests regardless of when contestants actually finish the work involved. Furthermore, in the (quite common) case of multiple contestants, efforts will vary across contestants, and some contestants may start on a submission but not finish. Also, comparing TC development effort

with in-house development is complicated due to varying factors, such as the overhead imposed on TC developers to understand the context and domain of the contest work at hand. Finally, in relation to quality, even though the front-end development done by the crowd was of relatively low complexity, the data presented in Table 3 and Figure 4 above illustrate that a significant number of issues have been raised.

An important consideration also is that TC's formulation of the software development process is effectively a waterfall approach, despite widely accepted wisdom that the waterfall model is not well suited to the rapid pace of change in modern development contexts. Agile and iterative methods are becoming increasingly popular in industry, including in domains where they have long been considered unsuitable [34], suggesting that these methods offer significant benefits over the waterfall model. The waterfall process also has serious consequences in that quality assurance practices are pushed to the end of the development process. While this can partly be addressed by adding a requirement to include unit tests, integration of the task is still done at a later stage, after a competition has finished.

Overall, TPI are of the opinion that crowdsourcing is limited in the areas in which it is suitable. Areas such as storyboards, GUI design, and even icon design, worked well for TPI. These areas seem to be quite self-contained without interdependencies. However, when there were dependencies between deliverables, and back and forth communication was necessary, the situation was quite different. Crowdsourcing competitions are effectively 'black-boxed,' meaning that while a competition is ongoing, a customer has limited means to communicate with TC developers. While there can be frequent communication with contestants prior to commencement of a contest, once it starts, communication is through a co-pilot, who acts as a proxy and thus inserts an additional interaction layer.

The results and insights of this case study suggest a number of open questions that we believe need further attention.

Contestants who are not familiar with the TopCoder software development process may not be as successful as other contestants who have extensive experience with the process. Archak [3] referred to this as a 'cold-start' effect. Similarly, crowdsourcing customers may also experience this. The duration of TPI's engagement with TC has been less than a year, and some of the challenges encountered may have been due to this lack of experience. Thus, one significant contribution to software engineering research would be to conduct longitudinal studies of organizations that have used a crowdsourcing approach for software development.

Of particular interest also would be studies that could compare a crowdsourced project with internal development. Such studies would need to focus on comparing key attributes including effort, cost, quality, and thus could address whether or not crowdsourcing does, in fact, deliver cheap and high-quality software in a short time-to-market. However, as noted earlier, these constructs need to be well operationalized to make a sound comparison.

We are aware of a few limitations of our study. It comprised a single case study, and therefore the issue of generalizability merits consideration. Clearly, the experiences and opinions from the participants in our study were specific to the case at TPI, and no statistical generalization can be drawn from this. However, the goal of this case study was to provide an in-depth account of a real-world case of crowdsourcing software development. Crowdsourcing is a topic that has been studied extensively in other domains where it appears to offer a variety of benefits. Our case study of crowdsourcing software development, however, suggests that there are significant challenges in a software development context. Another issue that merits attention during qualitative data analysis is that of 'multiple realities,' i.e., the unavoidable fact that understanding of reality is based on an individual subjective interpretation of the data, and that different individuals may interpret the same data in different ways. We used a number of tactics to address this. Firstly, the research process established an audit trail consisting of the interview transcripts and an extensive set of memos, which we revisited regularly. Secondly, to ensure correctness of the data we triangulated across a number of different data sources (interviews, documentation and TC website data). Furthermore, we also conducted 'member checking' by sending several earlier drafts of this paper to key participants to elicit feedback and clarification.

5.2 Conclusion

Crowdsourcing software development is a distinct and emerging approach to software development. Contrary to traditional outsourcing strategies that are characterized by contracts between two parties – the customer and supplier – crowdsourcing introduces a third party of unknown magnitude (sometimes very small in fact) and diversity, namely the crowd. Rather than a single supplier, there can be any number of contributors. This has clear implications for task decomposition, coordination and communication, planning and scheduling, QA, knowledge and IP, and motivation and remuneration.

This topic has received very limited attention from the software engineering research community. In this light, the contribution of this paper is threefold. Firstly, this paper provides a definition of crowdsourcing in a software development context that takes into consideration specific characteristics of software development tasks, as opposed to the small-grained and simple human intelligence tasks found on crowdsourcing platforms such as Amazon Mechanical Turk. Given the current lack of agreement on a general definition, this proposed definition can help to better focus and classify future research studies. For instance, our definition complies with Brabham's argument that open source is not a form of crowdsourcing, and thus, studies of opensourcing should not be classified as crowdsourcing studies. Secondly, based on a review of the literature on crowdsourcing, we derived a number of key concerns that are of particular importance in a software development context. Further research could use this framework to replicate the study at different organizations and different crowdsourcing platforms. Furthermore, each of the six themes in the framework can be used as a focus and starting point for further research. For instance, task decomposition and coordination are two important themes that warrant in-depth studies in their own right. Finally, to the best of our knowledge this paper presents one of the first in-depth industry case studies on crowdsourcing software development. Indeed, with a few exceptions ([3, 55]) there are no in-depth studies of crowdsourcing software platforms.

6. ACKNOWLEDGEMENTS

We thank the anonymous reviewers for useful suggestions. This work was supported, in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero—the Irish Software Engineering Research Centre (www.lero.ie).

7. REFERENCES

[1] Ågerfalk, P.J. and Fitzgerald, B. 2008. Outsourcing to an unknown worforce: Exploring opensourcing as a global sourcing strategy, *MIS Quarterly*, 32, 2.

[2] AppStori. http://www.appstori.com.

[3] Archak, N. 2010. Money, Glory and Cheap Talk: Analyzing Strategic Behavior of Contestants in Simultaneous Crowdsourcing Contests on TopCoder.com. *Proc. WWW*.

[4] Aurum, A., Jeffery, R., Wohlin, C. and Handzic, M. 2003. *Managing Software Engineering Knowledge*, Springer.

[5] Baddoo, N. and Hall, T. 2002. Motivators of Software Process Improvement: an analysis of practitioners' views, *J Syst Softw*, 62, 2, 85-96.

[6] Beecham, S., Baddoo, N., Hall, T., Robinson, H. and Sharp, H. 2008. Motivation in Software Engineering: A systematic literature review, *Inform Software Tech*, 50, 9-10.

[7] Begel, A., Bosch, J. and Storey, M.A. 2013. Social Networking Meets Software Development: Perspectives from GitHub, MSDN, Stack Exchange, and TopCoder, *IEEE Software*, 30, 1.

[8] Begel, A., Herbsleb, J.D. and Storey, M.-A. 2012. The Future of Collaborative Software Development. *Proc. Computer Supported Cooperative Work*.

[9] Bjørnson, F.O. and Dingsøyr, T. 2008. Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used, *Inform Software Tech*, 50, 11.

[10] Boehm, B. 2006. A View of 20th and 21st Century Software Engineering. *Proc. International Conference on Software Engineering.* Shanghai, China. ACM. 12-29.

[11] Boehm, B.W. 1981. *Software Engineering Economics*, Pearson Education.

[12] Bonabeau, E. 2009. Decisions 2.0: The Power of Collective Intelligence, *MIT Sloan Manage Rev*, 50, 2, 45-52.

[13] Boudreau, K.J., Lacetera, N. and Lakhani, K.R. 2011. Incentives and Problem Uncertainty in Innovation Contests: An Empirical Analysis, *Management Science*, 57, 5.

[14] Brabham, D.C. 2008. Crowdsourcing as a Model for Problem Solving: An Introduction and Cases, *Convergence*, 14, 1.

[15] Brabham, D.C. 2012. The Myth of Amateur Crowds: A critical discourse analysis of crowdsourcing coverage, *Information, Communication & Society*, 15, 3.

[16] Brabham, D.C. 2013. *Crowdsourcing*, MIT Press.

[17] Brooks, F.P. 1995. *The mythical man-month: essays on software engineering*, Addison-Wesley.

[18] Chanal, V. and Caron-Fasan, M.L. 2010. The Difficulties involved in Developing Business Models open to Innovation Communities: the Case of a Crowdsourcing Platform, *M@n@gement*, 13, 4.

[19] Chandler, D. and Kapelner, A. 2013. Breaking monotony with meaning: Motivation in crowdsourcing markets, *Journal of Economic Behavior & Organization*, 90, 123-133.

[20] Dabbish, L., Farzan, R., Kraut, R. and Postmes, T. 2012. Fresh Faces in the Crowd: Turnover, Identity, and Commitment in Online Groups. *Proc. CSCW*. ACM.

[21] Desouza, K.C. and Evaristo, J.R. 2004. Managing Knowledge in Distributed Projects, *Commun. ACM*, 47, 4.

[22] DiPalantino, D. and Vojnovic, M. 2009. Crowdsourcing and all-pay auctions. *Proc. 10th ACM Conf. Electronic Commerce*.

[23] Doan, A., Ramakrishnan, R. and Halevy, A.Y. 2011. Crowdsourcing systems on the World-Wide Web, *Commun. ACM*, 54, 4.

[24] Dow, S.P., Kulkarni, A., Klemmer, S.R. and Hartmann, B. 2012. Shepherding the Crowd Yields Better Work. *Proc. Computer-Supported Cooperative Work*. ACM.

[25] Ebner, W., Leimeister, M., Bretschneider, U. and Krcmar, H. 2008. Leveraging the Wisdom of Crowds: Designing an IT-supported Ideas Competition for an ERP Software Company. *Proc. 41st Hawaii International Conference System Sciences*.

[26] Erickson, L.B. 2012. Leveraging the Crowd as a Source of Innovation: Does Crowdsourcing Represent a New Model for Product and Service Innovation? *Proc. SIGMIS Computers and People Research*. ACM.

[27] Erickson, L.B., Petrick, I. and Trauth, E.M. 2012. Organizational Uses of the Crowd: Developing a Framework for the Study of Crowdsourcing. *Proc. SIGMIS-CPR.*

[28] Estellés-Arolas, E. and González-Ladrón-de-Guevara, F. 2012. Towards an Integrated Crowdsourcing Definition, *Journal of Information Science*, 38, 2.

[29] Faridani, S., Hartmann, B. and Ipeirotis, P.G. 2011. What's the Right Price? Pricing Tasks for Finishing on Time. *Proc. AAAI Workshop on Human Computation*.

[30] Feller, J., Finnegan, P., Fitzgerald, B. and Hayes, J. 2008. From Peer Production to Productization: A Study of Socially Enabled Business Exchanges in Open Source Service Networks, *Inform Syst Res*, 19, 4.

[31] Feller, J. and Fitzgerald, B. 2002. *Understanding Open Source Software Development*, Pearson Education Ltd.

[32] Fitzgerald, B. 2006. The Transformation of Open Source Software, *MIS Quarterly*, 30, 3.

[33] Fitzgerald, B. 2012. Software Crisis 2.0, *IEEE Comput.*, 45.

[34] Fitzgerald, B., Stol, K., O'Sullivan, R. and O'Brien, D. 2013. Scaling Agile Methods to Regulated Environments: An Industry Case Study. *Proc. 35th International Conference on Software Engineering.* San Francisco, CA, USA. IEEE.

[35] Greengard, S. 2011. Following the Crowd, *Commun. ACM*, 54, 2, 20-22.

[36] Herbsleb, J.D. and Grinter, R.E. 1999. Splitting the Organization and Integrating the Code: Conway's Law Revisited. *Proc. 21st Int'l Conf Software Engineering*.

[37] Herbsleb, J.D. and Mockus, A. 2003. An Empirical Study of Speed and Communication in Globally Distributed Software Development, *IEEE Trans Softw Eng*, 29, 6.

[38] Hetmank, L. 2013. Components and Functions of Crowdsourcing Systems—A Systematic Literature Review. *Proc. 11th Int'l Conf. Wirtschaftsinformatik*.

[39] Hoffmann, L. 2009. Crowd Control, *Commun. ACM*, 52, 3.

[40] Horton, J.J. and Chilton, L.B. 2010. The Labor Economics of Paid Crowdsourcing. *Proc. Conf. Electronic Commerce*.

[41] Howe, J. http://www.crowdsourcing.com.

[42] Howe, J. 2006. The Rise of Crowdsourcing, *Wired*, 14.

[43] Howe, J. 2008. *Crowdsourcing: Why the Power of the Crowd Is Driving the Future of Business*, Crown Business.

[44] Ipeirotis, P.G. 2010. Analyzing the Amazon Mechanical Turk marketplace, *XRDS*, 17, 2, 16-21.

[45] Ipeirotis, P.G. and Paritosh, P.K. 2011. Managing Crowdsourced Human Computation. *WWW*.

[46] Jouret, G. 2009. Inside Cisco's Search for the Next Big Idea, *Harvard Business Review*, 87, 9, 43-45.

[47] Kaufmann, N., Schulze, T. and Veit, D. 2011. More than fun and money. Worker Motivation in Crowdsourcing - A Study on Mechanical Turk. *Proc. 17th AMCIS*.

[48] Kazman, R. and Chen, H.-M. 2009. The Metropolis Model: A new Logic for Development of crowdsourced systems, *Commun. ACM*, 52, 7.

[49] Kinnaird, P., Dabbish, L., Kiesler, S. and Faste, H. 2013. Co-Worker Transparency in a Microtask Marketplace. *Proc. Computer Supported Coordination Work*.

[50] Kittur, A. 2010. Crowdsourcing, Collaboration and Creativity, *XRDS*, 17, 2.

[51] Kittur, A., Nickerson, J.V., Bernstein, M.S., Gerber, E.M., Shaw, A., Zimmerman, J., Lease, M. and Horton, J.J. 2013. The Future of Crowd Work. *Proc. CSCW*. ACM.

[52] Kittur, A., Smus, B., Khamkar, S. and Kraut, R.E. 2011. CrowdForge: Crowdsourcing Complex Work. *Proc. ACM Symposium on User Interface Software and Technology*.

[53] Kraut, R.E. and Streeter, L.A. 1995. Coordination in Software Development, *Commun. ACM*, 38, 3.

[54] Kulkarni, A., Can, M. and Hartmann, B. 2012. Collaboratively Crowdsourcing Workflows with Turkomatic. *Proc. Computer-Supported Cooperative Work*.

[55] Lakhani, K.R., Garvin, D.A. and Lonstein, E. 2010. TopCoder (A): Developing Software through Crowdsourcing, *Harvard Business School 610-032*.

[56] Lakhani, K.R. and Panetta, J.A. 2007. The Principles of Distributed Innovation, *Innovations: Technology, Governance, Globalization*, 2, 3.

[57] LaToza, T.D., Towne, W.B., van der Hoek, A. and Herbsleb, J.D. 2013. Crowd Development. *Proc. 6th CHASE Workshop.* San Francisco, CA, USA. IEEE.

[58] Malone, T.W. and Crowston, K. 1994. The Interdisciplinary Study of Coordination, *ACM Comput Surv*, 26, 1.

[59] Mao, K., Yang, Y., Li, M. and Harman, M. 2013. Pricing Crowdsourcing-Based Software Development Tasks. *Proc. 35th International Conference on Software Engineering*.

[60] Mason, W. and Watts, D.J. 2009. Financial Incentives and the `Performance of Crowds'. *Proc. KDD-HCOMP*. ACM.

[61] Mockus, A., Fielding, R. and Herbsleb, J.D. 2000. A Case Study of Open Source Software Development. *Proc. ICSE*.

[62] Musson, R., Richards, J., Fisher, D., Bird, C., Bussone, B. and Ganguly, S. 2013. Leveraging the crowd: how 48,000 users helped improve Lync performance, *IEEE Softw.*, 30, 4.

[63] Naparat, D. and Finnegan, P. 2013. Crowdsourcing Software Requirements and Development: A Mechanism-based Exploration of `Opensourcing'. *Proc. 19th AMCIS*.

[64] Parnas, D.L. 1972. On the criteria to be used in decomposing systems into modules, *Commmun. ACM*, 15, 12.

[65] Pilz, D. and Gewald, H. 2013. Does Money Matter? Motivational Factors for Participation in Paid- and Non-Profit-Crowdsourcing Communities. *Proc. 11th Int'l Conf. Wirtschaftsinformatik*.

[66] Raymond, E.S. 2001. *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly Media.

[67] Rouse, A.C. 2010. A Preliminary Taxonomy of Crowdsourcing. *Proc. Australasian Conference on Information Systems (ACIS)*.

[68] Runeson, P., Höst, M., Rainer, A. and Regnell, B. 2012. *Case Study Research in Software Engineering: Guidelines and Examples*, Wiley.

[69] Savage, N. 2012. Gaining Wisdom from Crowds, *Commun. ACM*, 55, 3, 13-15.

[70] Schenk, E. and Guittard, C. 2009. Crowdsourcing: What can be outsourced to the crowd, and why?

[71] Schenk, E. and Guittard, C. 2011. Towards a Characterization of Crowdsourcing Practices, *Journal of Innovation Economics*, 1, 7.

[72] Schwarz, A., Mehta, M., Johnson, N. and Chin, W.W. 2007. Understanding Frameworks and Reviews: A Commentary to Assist us in Moving Our Field Forward by Analyzing Our Past, *Database Adv Inform Syst*, 38, 3.

[73] Seaman, C. 1999. Qualitative Methods in Empirical Studies of Software Engineering, *IEEE Trans Softw Eng*, 24, 4.

[74] Simula, H. 2013. The Rise and Fall of Crowdsourcing? *46th Hawaii International Conference on System Sciences*.

[75] Singer, Y. and Mittal, M. 2013. Pricing Mechanisms for Crowdsourcing Markets. *Proc. WWW*.

[76] Stol, K. and Fitzgerald, B. 2014. Research Protocol for a Case Study of Crowdsourcing Software Development. *http://staff.lero.ie/stol/publications,* University of Limerick.

[77] Surowiecki, J. 2005. *The Wisdom of Crowds: Why the Many Are Smarter Than the Few*, Abacus.

[78] Tajedin, H. and Nevo, D. 2013. Determinants of success in crowdsourcing software development. *Proc. SIGMIS Computer and People Research.* Cincinnati, OH, USA.

[79] Tamburri, D., Lago, P. and van Vliet, H. 2013. Organizational social structures for software engineering, *ACM Comput Surveys*, 46, 1.

[80] TopCoder. http://www.topcoder.com.

[81] TopCoder. http://www.topcoder.com/whatiseoi/.

[82] TopCoder. 2010. 10 Burning Questions on Crowdsourcing: Your starting guide to open innovation and crowdsourcing success. http://www.topcoder.com/blog/10-burning-questions-on-crowdsourcing-and-open-innovation.

[83] Tung, Y.-H. and Tsenga, S.-S. 2013. A novel aproach to collaborative testing in a crowdsourcing environment, *J Syst Softw*, 86, 8.

[84] uTest. http://www.utest.com.

[85] Vukovic, M. 2009. Crowdsourcing for Enterprises. *SERVICES*.

[86] Wolfson, S.M. and Lease, M. 2011. Look Before You Leap: Legal Pitfalls of Crowdsourcing. *ASIST Annual Meeting*.

[87] Yin, R.K. 2003. *Case Study Research*, 3rd ed., SAGE.

[88] Zhao, Y. and Zhu, Q. 2012. Evaluation on crowdsourcing research: Current status and future direction, *Inf Syst Front*, April.