Competition-Based Crowdsourcing Software Development: A Multi-Method Study from a Customer Perspective

Klaas-Jan Stol, Bora Caglayan, and Brian Fitzgerald

Abstract—Crowdsourcing is emerging as an alternative outsourcing strategy which is gaining increasing attention in the software engineering community. However, crowdsourcing software development involves complex tasks which differ significantly from the micro-tasks that can be found on crowdsourcing platforms such as Amazon Mechanical Turk which are much shorter in duration, are typically very simple, and do not involve any task interdependencies. To achieve the potential benefits of crowdsourcing in the software development context, companies need to understand how this strategy works, and what factors might affect crowd participation. We present a multi-method qualitative and quantitative theory-building research study. First, we derive a set of key concerns from the crowdsourcing literature as an initial analytical framework for an exploratory case study in a Fortune 500 company. We complement the case study findings with an analysis of 13,602 crowdsourcing competitions over a ten-year period on the very popular Topcoder crowdsourcing platform. Drawing from our empirical findings and the crowdsourcing literature, we propose a theoretical model of crowd interest and actual participation in crowdsourcing competitions. We evaluate this model using Structural Equation Modeling. Among the findings are that the level of prize and duration of competitions do not significantly increase crowd interest in competitions.

Index Terms—Crowdsourcing, software engineering, multi-method study, case study, sample study

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 [1], [2], [3]. There is an increasing trend towards globalization with a focus on collaborative methods and infrastructure [4], [5]. One emerging approach to getting work done is crowdsourcing, a sourcing strategy that emerged in the 1990s [6], but started to gain significant attention when the term "crowdsourcing" was coined in 2005 [7]. Driven by Web 2.0 technologies [8], [9], organizations can tap into a workforce consisting of potentially 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) perform those tasks that match their interests and abilities [10].

Crowdsourcing has been adopted in a wide variety of domains, such as design of T-shirts [11] and pharmaceutical research and development [12], and there are numerous crowdsourcing platforms through which customers and suppliers can find each other [13], [14]. One of the best known crowdsourcing platforms is Amazon Mechanical Turk (AMT) [15]. 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 [16], [17]. 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 [15].

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 [18]. However, there are examples of crowdsourcing complex tasks; for example, InnoCentive deals with problem solving and innovation projects, which may yield payments of thousands of US dollars [11].

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:

- *Cost reduction* [16], [19], [20] 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* [16], [21], [22], [23] 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.

- *K.-J. Stol is with the Department of Computer Science, University College Cork, Cork T12 YN60, Ireland, and with Lero—The Irish Software Research Centre, University of Limerick, Limerick V94 T9PX, Ireland. E-mail: k.stol@ucc.ie.*
- *B. Caglayan is with IBM Ireland, Dublin 4, Ireland. E-mail: bora.caglayan@ibm.com.*
- *B. Fitzgerald is with Lero—The Irish Software Research Centre, University of Limerick, Limerick V94 T9PX, Ireland. E-mail: bf@lero.ie.*

Manuscript received 29 Nov. 2016; revised 18 Oct. 2017; accepted 12 Nov. 2017. Date of publication 23 Nov. 2017; date of current version 21 Mar. 2019. (Corresponding author: Klaas-Jan Stol.)
Recommended for acceptance by D. Damian.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2017.2774297

- *Higher quality through broad participation* [20], [24], [25]: 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 competitions where the highest quality 'winning' solution is chosen;
- *Creativity and open innovation* [20], [21], [26], [27], [28], [29]: there are many examples of *"wisdom of the crowd"* creativity 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. This application of skills in a new domain is also referred to as *"near-field repurposing of knowledge"* [25, pt. II].

Given that the first three benefits listed above (cost, time and quality) directly address the three central problems of the so-called "software crisis" [30], it is not surprising that a number of authors have argued that crowdsourcing may become a common approach to software development [14], [31], [32], [33]. 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"* [12]. As Lakhani and Panetta [12] 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.

Crowdsourcing has received considerable interest from researchers in disciplines such as human-computer interaction and information systems, and more recently in software engineering (see Mao et al. [14] for an extensive overview). Thus far, most studies have presented analyses of developers and platforms including AMT and Topcoder [34], [35], [36], [37]. However, very few studies have studied crowdsourcing from a *customer perspective*, as a practical alternative approach to outsourcing software development. Studying crowdsourcing from this perspective is important in order to better understand how organizations can engage with this new and emerging type of "unknown workforce." Hence, our research goal was as follows:

Research Goal. To develop a better understanding of crowdsourcing as a software development strategy.

With this goal in mind, we first conducted an industry case study at a company that used Topcoder to crowdsource a large project, which we reported in our earlier paper published at the *36th International Conference on Software Engineering (ICSE'14)* [38]. The case study helped us to understand some of the tension points in crowdsourcing software development. The case study represents a "rich, empirical description of a particular instance" [39] of the phenomenon of competition-based crowdsourcing in a software development context. This article revises the ICSE'14 paper and extends it in several ways:

- We extend the case study analysis with a large-scale quantitative analysis of the Topcoder platform. As case studies are inherently limited in scope, this quantitative analysis helps to put the findings of the case study in perspective.
- Based on the specific findings that arose in the case study findings as well as the extant literature, we developed a theoretical model that represents some of the key factors that affect a crowd's interest and participation in crowdsourcing competitions.
- Using a data set of over 13,600 contests held on the Topcoder platform, we evaluated the model using structural equation modeling.

The remainder of this article is structured as follows. Section 2 defines crowdsourcing for software development as a distinct form of outsourcing that differs from peer production (cf. [40]) and opensourcing (cf. [41]). We contrast crowdsourcing with opensourcing and outsourcing and derive a definition of crowdsourcing in the context of software development. Section 3 outlines our multi-method research approach. Section 4 presents the results of the case study and complements the qualitative findings with analyses of the Topcoder platform. In Section 5 we develop and evaluate a theoretical model of crowd interest in participating and actual participation in competitions. In Section 6 we discuss implications and limitations of the study, and we conclude with some suggestions for future work.

2 Background

There are a number of crowdsourcing platforms specifically targeting software development (and related tasks such as testing [42], [43], [44]). The largest is Topcoder [45], which has over 1.2 million members. Crowdsourcing is sometimes considered similar to open source, but we argue there are a number of key differences between these strategies. To clearly distinguish these models, we position crowdsourcing software development in relation to outsourcing and opensourcing, and offer 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 the case study presented in Section 4.

2.1 Positioning and Defining Crowdsourcing Software Development

In earlier work, we positioned crowdsourcing as a separate strategy from outsourcing and open source [38], [46]; Table 1 extends this positioning, which clearly delineates crowdsourcing as a unique form of outsourcing. We conclude this section with our own definition.

There are numerous definitions for the term 'crowdsourcing' [48], [49]. Howe presented the following definition [7]:

> *"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 'sound-bite' version, defines crowdsourcing as the *"application of Open Source principles to fields outside of software"* [50]. 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 [41], and some authors consider this a form of crowdsourcing [51], [52]. Others argue that crowdsourcing differs from open source (and thus, opensourcing), in that the latter is a public good, whereas the

TABLE 1
Characterization of Crowdsourcing

	Competition-based Crowdsourcing	Opensourcing	Outsourcing
Locus of control	Customer prescribes the work to be done. Limited interaction between customer and workforce.	Control over an open source project lies with core team, or *benevolent dictator* in some projects. Meritocracy.	Customer decides project strategy and makes major decisions as to the work that is performed.
Nature of participation	Competitive, possibly collaborative depending on the platform	Collaborative, possibly independent and differing agendas and interests	Collaborative, all teams serve the same goal (though not always without conflicts)
Nature of the workforce	Usually individual developers who are unknown to the customer, but known to intermediary	Mix of individual open source contributors and potentially organizations that contribute. Unknown workforce, but relationships can develop over time.	Customer and supplier may build up relationship over time through regular/daily interaction. Supplier is typically an organization, not an individual.
Duration of engagement	Short, ad-hoc commitment for duration of competition	Typically prolonged commitment by community developers	Project-specific, contractual commitment. Mid-to long-term commitment
Customer / Initiator motivation	Overcoming short-term lack of resources; opportunity to tap into creativity of the crowd (innovation); cost reduction	Driven by trend of commodification of technology, cost sharing of maintenance effort	Resource saving by exploiting lower wages (in case of offshoring); strategic decision if software development not a core activity.
Developer / Supplier motivation	Extrinsic motivation (e.g., payments) or delayed extrinsic motivations (career prospective), internalized extrinsic motivations, e.g., learning [47]	Both intrinsic motivation (e.g., fun, sense of achievement, creative satisfaction) and extrinsic motivation (money, career prospective); increasingly developers employed by companies	Exclusively extrinsic; supplier provides services as a commercial activity.

former is focused on extracting economic value [34]. A more significant distinction between open source and crowdsourcing, is that the *locus of control* in the former is essentially with the crowd, and there is no overarching entity that coordinates the overall effort [8]. Open source projects tend to be self-organizing [53], 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. Even though many open source projects are moving more towards formal organization [54], [55], the locus of control remains largely with the crowd/open source community.

The nature of participation also differs from opensourcing and outsourcing. While different participation models exist, a typical feature of many crowsourcing platforms is the competition-based nature of participation. Topcoder is the largest platform for crowdsourcing software development and uses competitions, whereby a customer advertises a task that is then taken on by members of the crowd. The crowd member with the highest-rated submission wins the competition and receives payment. Thus, the competitive element and monetary incentive tend to preclude collaboration. Howe characterized crowdsourcing as *"outsourcing on steroids"* [11, p.46], suggesting that crowdsourcing is merely a form of outsourcing. However, the duplication of work being performed in parallel does not apply to outsourcing. The nature of the workforce also sets crowdsourcing apart as an outsourcing strategy with the crowd typically not known to the customer. A "traditional" outsourcing scenario is characterized by a contractual agreement with a specific (and thus known) supplier before the work is performed—over time, a relationship may build up between these two parties. In a crowdsourcing scenario, the crowd plays the role of "supplier" but it is not known in advance who will submit, and therefore which member of the crowd will be paid.

Other dimensions that set crowdsourcing apart from opensourcing and outsourcing include the duration of engagement, and the motivations for both the customer and the 'supplier' (i.e., developers that perform the work) (see Table 1). Based on the characteristics in the table, our definition of crowdsourced software development is the following:

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

A variety of platforms exist for crowdsourcing software development [14], and the model of participation can vary. Topcoder, the largest platform in terms of number of members, organizes tasks as competitions, but others (e.g., oDesk) act as online marketplaces rather than "competitive arenas." Another variant is 'microtasking' [52], for which tasks tend to be independent. In this article, we focus on competition-based crowdsourcing.

2.2 Key Concerns in Crowdsourcing Software Development

In this section we derive a set of key concerns for crowdsourcing software development. While a few research frameworks have been proposed, these tend to focus on crowdsourcing as a general topic [56], [57], and not specifically to software development. A framework helps define the boundaries of a research area [58]. Drawing on the literature, 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 discusses these themes in detail.

2.2.1 Task Decomposition

A key issue in crowdsourcing is that work is decomposed into a set of smaller tasks [17], [59], [60]. This issue is highly relevant in outsourcing scenarios, and Herbsleb and Grinter [61] reminded us of Parnas' definition of a module as *"a responsibility assignment rather than a subprogram"* [62]. 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 [15], 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 [63]. Kulkarni et al. [60] termed this challenge the "workflow design problem." More efficient decompositions can lead to an increased parallelism [63]. LaToza et al. [64] proposed a platform ("CrowdCode") to support the decomposition of programming work into microtasks. 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 [21]. Tajedin and Nevo [65] 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 complex tasks, as is the case in software development, there is a need for coordination [18]. Malone and Crowston [66] 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 [67]. Although related to task decomposition discussed above, coordination is specifically concerned with communication, interdependencies and integrating various parts into a whole [63], [67], [68]. This characterization of coordination seems to assume that activities are conducted within an organization. Clearly, in crowdsourcing, participants who submit 'solutions' are independent agents and not part of the customer organization—in other words, members of the crowdsourcing platform cannot be assigned tasks. Instead, developers self-select tasks to work on; several researchers have focused on assisting the crowd by making recommendations based on their prior record [36], [69]. Because different tasks may be performed by many different workers, incompatibilities may arise among provided solutions [63].

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 greater the number of people involved, the greater the communication overhead [70]. Whether or not this applies in a crowdsourcing context depends on whether the work is done in a collaborative or competitive fashion [71]. Several platforms including Topcoder organize tasks as competitions; a winner (and runner-up) is selected based on peer-review of the submissions by the community [1].

2.2.3 Planning and Scheduling

In the case of crowdsourcing, tasks are allocated to an unknown workforce to complete, and as a result an organization relinquishes control of that particular work. This may speed up development, as tasks can be completed in parallel and independently of an organization's in-house workforce, particularly when payment is contingent on timely delivery. One of the promises of crowdsourcing is to shorten the product development cycle [72], [73]. In order to achieve this, it is important that the desired schedule of a crowdsourcing organization can be adhered to by the crowd. For example, a core challenge is to ensure that sufficient workers are available when needed [18]. While there may be extensive expertise within the crowd, very specific domain knowledge may not always be available at the moment it is needed. Such circumstances introduce a level of uncertainty as to whether or not the work will be completed on time [71]. Furthermore, it is important to ensure that sufficient time is given to developers, relating the issue of planning to the size and scope of a task.

2.2.4 Quality Assurance

Another suggested benefit of crowdsourcing is the potential for high quality submissions [20], [24], [25]. At the same time, there is a risk of 'noise' if the majority of submissions are of low quality [59], [74], making the task of assessing submission quality more cumbersome. 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"* [75]. Linus's Law refers specifically to testing and debugging, which is only one type of activity that can be crowdsourced on Topcoder, but development tasks also benefit from having 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' participate in a certain competition, a platform such as Topcoder is fully transparent. Prior to participating, contestants must register for a competition, and registrants can see who else has registered. An experiment on crowdsourcing microtasks suggests, however, that the greater the number of competition participants, the lower the quality of the work [76]. One characteristic sometimes ascribed to the crowd is that it consists mostly of amateurs [9], thus suggesting that the resulting quality of output may not be on par with professional work. However, Brabham suggested that this might be a myth [77].

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 almost 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"* [18]. Also, there can be disagreement about a solution; Kittur [16] 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, the scope of a task, or the (subjective) code quality of a submission. 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 example, experienced members of the community perform peer-reviews of the submitted software. Similar to peer-reviews in open source, such reviews are "truly independent" [53] 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' of the crowd has also been suggested to improve quality [60], [74]. Kulkarni et al. [60] found that letting the crowd plan amongst themselves without supervision by a requester was partially successful, but that intervention by a requester during the workflow could improve quality significantly. LaToza et al. [78] experimented with two-phased design competitions, allowing the crowd to "borrow" from initial submissions, which resulted in improved design quality.

2.2.5 Knowledge and Intellectual Property

Software development is a knowledge-intensive activity, and knowledge management is therefore an important topic within the software engineering field [79], [80], [81]. 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 [82]. A high level of turnover may lead to schedule and cost overruns [83], which in turn jeopardizes a successful outcome of a competition.

One type of knowledge of particular concern in crowdsourcing software development tasks is intellectual property (IP) [21], [84]. IP 'leakage' and the consequent loss of competitive advantage is a challenge in adopting crowdsourcing [28]. 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 [85], [86]. 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 protect 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 [86], for example, if the solution contains open source code with the restrictive GNU Public License (GPL) license. These issues expose a customer to a variety of risks.

2.2.6 Motivation and Remuneration

A final consideration in crowdsourcing is that of motivation [59], [87], [88] and remuneration [35], [89], [90], [91], [92], [93], [94]. Motivation is a topic that has received considerable attention in the software engineering literature, given that it is reported to be a major factor in project success [95], [96]. Motivational factors can be external or intrinsic. Extrinsic factors are conditions surrounding a job [97], whereas intrinsic factors relate to the job itself (e.g., having fun, gaining a sense of achievement). The compensation of a certain crowdsourcing task should depend on the expected duration and the complexity of a task. Tasks can vary in complexity; as mentioned above, tasks on Amazon Mechanical Turk are often called 'micro-tasks,' which can be as simple as tagging an image taking only a few seconds. 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 [15]. One claimed benefit of crowdsourcing is that it can greatly reduce cost [21]. Yet, determining an appropriate price is a key challenge for crowdsourcing in general [90], [98], and also for software development specifically [21], [35], [92].

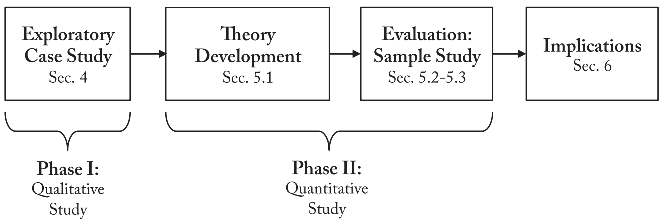

Exploratory Case Study Sec. 4
Theory Development Sec. 5.1
Evaluation: Sample Study Sec. 5.2-5.3
Implications Sec. 6
Phase I: Qualitative Study
Phase II: Quantitative Study

Fig. 1. Design of the two-phased study.

3 Research Design

The goal of our study is to develop an understanding of crowdsourcing as an outsourcing strategy in the context of software development. We adopted a multi-method research approach in this study, consisting of two phases (see Fig. 1). Using such a multi-method approach helps to ameliorate the shortcomings of research strategies and is becoming increasingly prevalent [41], [99], [100], [101], [102].

Phase I of our study is an exploratory qualitative industry case study of a global company who used Topcoder to crowdsource a software development project. Exploratory case studies are appropriate to explore contemporary phenomena that have not previously been researched [99], [103]. Indeed, to the best of our knowledge, this is the first qualitative case study that investigates crowdsourcing from an enterprise customer perspective. (We note that several published studies have investigated crowdsourcing from different perspectives, or using quantitative methods—these were discussed in Section 2.) The qualitative case study helps "bring to life" the crowdsourcing phenomenon from a customer's perspective, something that is often missing in purely quantitative analyses. While rich in context and "thick narrative description," case studies are limited in that findings are not statistically generalizable to other settings. However, the goal of "phenomenon-driven" exploratory case studies is not to test theory—instead, such case studies are highly appropriate for *theoretical* rather than *statistical* generalization and developing understanding of key concepts [103], and benefit from exploiting *"unusual research access"* [39]. Section 4 presents the results of this first phase.

Phase II complements the qualitative study of Phase I through a theory development strategy combined with an evaluative quantitative sample study. This phase builds on some of the key findings of the case study as well as the literature in a theory development approach, resulting in a

TABLE 2
Key Strengths and Weaknesses of Employed Research Strategies

	Exploratory case study	Sample study
Strengths	Rich context to facilitate understanding	Findings are generalizable (within certain boundaries depending on the sample)
	Facilitates study of phenomena in a natural setting	Suitable to evaluate relationships between a fixed number of variables
Weaknesses	Findings not statistically generalizable	Not amenable for 'discovery' of concepts and understanding
	Inability to manipulate variables	Inflexible design; analyses limited to available data; once data is collected, the research design does not allow much change

theoretical model consisting of a set of hypotheses, which are evaluated on a large data set from the Topcoder platform using Structural Equation Modeling (SEM). Sample studies typically use cross-sectional data from a large number of organizations, developers, or other units of analysis—often these would be collected through a survey. In our study, we collected data on 13,602 contests from the Topcoder platform. Sample studies such as these facilitate the testing of hypotheses on a limited number of variables. Such fixed-design studies are useful to establish relationships, which is not possible in exploratory qualitative case studies. However, as Kaplan and Duchon pointed out, the *"stripping of context buys 'objectivity' and testability at the cost of a deeper understanding of what actually is occurring"* [104] (as cited by Gable [99]). Section 5 reports the results of Phase II.

The strengths and weaknesses of the research strategies employed in the two phases are summarized in Table 2 (please note this table is not exhaustive). The case study and sample study are alternative and complementary strategies rather than competing [99], [105]. The remainder of Section 3 presents the design of these two phases in more detail.

3.1 Phase I: Design of the Case Study

The goal of the case 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. Case study research is particularly well suited to study real-world phenomena that cannot be studied separately from their context [103], and has become increasingly popular as a method in software engineering research [106] (cf. studies on distributed development [61] and open source software development [107]). This section outlines the setting (Section 3.1.1) and qualitative methods (Section 3.1.2) that we employed in our industry case study. To better understand and interpret the case study findings, we include comparative quantitative analyses of the Topcoder platform—Section 3.1.3 describes these analyses in detail.

3.1.1 Setting

TechPlatform Inc. (TPI—a pseudonym) is a Fortune 500 company 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.

The platform through which TPI crowdsourced its software development is Topcoder, which is the largest software development crowdsourcing platform and its community has grown exponentially, from 50,000 to over 1.2 million members between 2004 and 2017. Topcoder has an extremely impressive customer list of blue chip companies. In promoting their services, Topcoder suggests that customers can *"Try more often, Succeed more often, Spend Less"* [108]. Topcoder 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 [108]. For this study, we focus on the software development pillar. Topcoder accomplishes software development tasks for customers through a series of competitions. The Topcoder 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 Topcoder community compete and submit solutions.

Initially, Topcoder employed Program Managers to oversee customer projects and assist customers as a project "liaison," but several years ago the platform introduced a "self-service" model to save costs [21]. This direct model involves "co-pilots" within the Topcoder community to act as an interface between customers and crowd developers, and to help choose winners for the various competitions. Co-pilots are experienced "elite" Topcoder community members who have proven themselves in the past on the Topcoder platform [25]. They manage the technical aspects of crafting and running competitions through to successful delivery. Topcoder 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"* [109].

3.1.2 Qualitative Data Collection and Analysis

We conducted a number of face-to-face, semi-structured interviews with key informants at TPI who were involved with the Topcoder 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 a research protocol [110]. 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 [106]. Member checking is a recommended tactic to ensure that findings are indeed "experienced" or "felt by" the participants of a study [111]. This also provided opportunities to seek clarifications when necessary.

Data were analyzed using qualitative methods as described by Seaman [112]. All interviews were transcribed, resulting in 112 pages of text. The analysis consisted of

TABLE 3
Case Study Data Sources

Interviews during three site visits and two teleconference calls	Divisional CTO Software architect Software development manager Program manager Project manager
Company documentation	Contest schedule and status documentation Internal TPI Wiki documentation Internal report on key challenges in the crowdsourcing project
Data from crowdsourcing platform	Contest information from Topcoder platform

coding the transcripts using the six themes discussed in Section 2.2 as seed categories or "analytical bins" [113]. The derivation of these themes took place prior to data collection, which represents a key difference with Grounded Theory studies [114]. The transcripts were analyzed in parallel by two 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. The analysis of the interview transcripts from different interviews represents triangulation among informants, which helps to establish dependability of the findings. 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. These sources included documentation on the crowdsourcing schedules, project documentation that TPI stored on an internal wiki, and contest information drawn from the Topcoder website. Table 3 summarizes the data sources for the case study.

3.1.3 Quantitative Data Collection and Analysis

In order to contextualize the findings of the case study, we analyzed data that we collected through Topcoder's public API. The goal of the quantitative analyses was to contextualize the case study findings to better understand whether the TPI contests were 'atypical' or exceptional in terms of the technologies used, range of reward prizes, and duration. Thus, the quantitative analyses provide comparative background information. We collected all the publicly available data in November 2016. All data were stored in a SQLite database, and analyzed with Python and the *R* statistical package.

We filtered the data based on a few criteria. First, we removed the challenges with a first prize of less than 100 US dollars. The rationale for this is that we considered these challenges as trivial, and not representative of the competitions that companies normally post. We found that these challenges usually had very low monetary rewards (between US $0.00-1.00 prize). During our analysis, we identified a user named "analysis." This user made a total of 4,697 submissions, which is ten times as many as the second most active user, and represents 7.3 percent of the total number of submissions. Furthermore, whenever this user submitted, no other submissions were made by anyone else. As these competitions are not representative, we decided to remove these. Finally, we only considered competitions that had successfully finished, which led to a further reduction of the data set. Our final Topcoder platform sample contained data on 13,602 competitions. During our analysis, we encountered a small inconsistency in the extracted data. Approximately 1.5 percent of the (distinct) registrants registered or participated in a competition before registering as a member of the platform. We adjusted the registration date as the date of first activity in the system for these registrants. Given that this only affected a very small percentage, we retained these entries in our data set.

3.2 Phase II: A Theoretical Model of Crowdsourcing Software Development

In the second phase, we drew from the extant literature on crowdsourcing, and the findings from the first phase to develop a set of hypotheses which are integrated into a single theoretical model to increase our understanding of crowdsourcing software development. The model is evaluated using Structural Equation Modeling [115] (discussed in detail below). SEM is a powerful statistical approach but has been rarely used in software engineering studies to date. Notable exceptions are a study on quality, effort, and governance in open source [116], and a study of teamwork quality and project success [117]. Therefore, one of this article's contributions is to illustrate the SEM approach in developing and evaluating a theoretical model on a phenomenon within the software engineering domain.

SEM is a second-generation statistical approach. In so-called first-generation statistical methods including multiple regression and ANOVA, parameters are typically estimated using Ordinary Least Squares (OLS). The goal of OLS is to find coefficients that minimize the average squared distance between the data points and a regression line [115]. While the overall goal of SEM is similar, namely to identify coefficients that represent a best-fit with the observed data, what is used as "observed data" are the observed covariances between variables and their variances [115]. For this reason, this type of SEM is sometimes referred to as CBSEM (covariance-based SEM), to distinguish it from Partial Least Squares (PLS) SEM. Instead of OLS, the default algorithm for estimating coefficients in SEM is Maximum Likelihood (ML); in our study we use a robust variant of ML (Section 5.2 provides further details).

In SEM, the researcher specifies a theoretical (hypothesized) model by defining a number of interrelated hypotheses; based on this, a variance–covariance matrix is generated. A second variance–covariance matrix is generated based on a set of sample data. The goal of SEM, then, is to test whether the two matrices are different: if they are, then the sample data do not support the researcher's theoretical model. Consequently, a non-statistically significant difference (using χ^2) between the two matrices indicates that the theoretical model fits the empirical observations. Further details on the mechanisms of SEM are beyond the scope of this article, but we refer interested readers to several excellent reference works that are available [115], [118], [119].

All coefficients in the structural equation model are estimated simultaneously. Thus, the significance and strength of relationships in the structural equation model should be assessed in the context of the model as a whole. Evaluating the hypothesized relationships is only valid when the model

TABLE 4
Top 10 Most Popular Technologies on the Topcoder Platform

Rank	Technology	Frequency
1	Java	4,019
2	JavaScript	3,371
3	HTML	2,333
4	CSS	2,181
5	Node.js	1,234
6	Angular.js	1,047
7	HTML5	1,031
8	iOS	772
9	J2EE	653
10	C#	479

itself represents a good fit, that is, the theoretical model is not significantly different from the observed model.

The structural equation model was developed by defining a set of constructs and relationships that comprise our theory [120]. Specifically, we focused on a number of salient concepts identified in the case study, including contest reward, contest duration, and the degree of "parallelization" of contests. We operationalized our constructs using singular metrics. In that sense, our model is a *path model* [118, p. 5], which is one type of structural equation model. A path model is essentially a regression model; however, regression is limited to a single dependent variable that is predicted or explained by one or more independent variables. A path model does not have this constraint and thus allows for more complex models.

The structural equation model was implemented using the *lavaan* library for the statistical package *R*, version 0.5-23 [121]. The structural equation model was then evaluated using a set of fit criteria. In particular, the theoretical model is evaluated using three types of criteria [118]:

- *Measures of model fit*, such as the Root Mean Square Error of Approximation (RMSEA);
- *Statistical significance* of individual parameter estimates for the model's paths;
- *Direction and magnitude* of parameter estimates; in particular, evaluating whether or not the direction (indicated by the parameter's sign) makes sense.

A more detailed presentation and discussion of the structural equation model follows in Section 5, but first we present our exploratory case study in Section 4.

4 Findings from the Case Study

The application which TPI selected for crowdsourcing was *Titan* (a pseudonym), a web application to be used by TPI field engineers when migrating from one platform to another as part of a customer engagement. Titan is used to support IT departments during the migration of machine contents from one machine to another and consists of several components, including legacy components that were not replaced. The latter implement the core functionality for migration operations. The part that was crowdsourced is therefore best characterized as a front-end information system which would have to be integrated with the legacy components. 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 current desktop application. TPI did not have extensive experience of HTML5 in-house and were therefore very keen to innovate by leveraging HTML5 expertise from the large global Topcoder community. Table 4 lists the Top 10 most used technologies on the Topcoder platform, which lists HTML5 as the seventh most-used. The table shows several other languages and libraries primarily used for the web such as JavaScript, CSS and Angular.js. TPI's decision to use crowdsourcing for front-end development and to seek HTML5 expertise is therefore quite justifiable given the characteristics of the Topcoder platform.

Sections 4.1, 4.2, 4.3, 4.4, 4.5, and 4.6 draw on the themes presented in Section 2 to discuss the key findings of the TPI case study. For each theme, we provide information from the Topcoder platform sample data set to put the case study findings in context. Section 4.7 summarizes the case study findings.

4.1 Task Decomposition

The choice as to what parts of the product were appropriate for crowdsourcing was not entirely trivial for TPI. First, the decision as to what work to crowdsource was primarily based on internal resources (or lack thereof). Second, code and executables which were self-contained (without interdependencies) would be easier to merge and hence more suitable for crowdsourcing. If code from Topcoder had to be directly merged with code being developed in-house, this would be more problematic. The final factor taken into account was the amount of domain knowledge required for a certain task. Tasks that required the least amount of domain knowledge were deemed most suitable for crowdsourcing.

In order to minimize the modifications that would need to be made to the Topcoder code after delivery, TPI made the header and footer browser code available to crowd developers. This was to ensure this standard format would be maintained by all crowd developers. For the Titan application, TPI's policy was to only use HTML5 where a feature was supported by all browser platforms to increase portability. Initially, there was an expectation that the Topcoder 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 crowd developers. The expected innovation from the "crowd" was thus precluded by the TPI specification.

The TPI project consisted of 44 successful competitions which fell into different categories. Table 5 lists the numbers of competitions per category, as well as how the competitions in our overall Topcoder platform sample were distributed. For example, the most prevalent category in the sample is "Assembly Competition" with 3,426 competitions, representing just over 25 percent.

In order to minimize integration effort later on, TPI sought to have crowd developers work with a real back-end core as opposed to stub services. However, by the time development with Topcoder started, the core was not ready and stubs were used during most development competitions. 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

TABLE 5
Competition Types and Frequency

Competition type	Case study	Topcoder sample
Architecture	6	791
Assembly Competition	23	3,426
Bug Hunt	–	536
Code	–	1,581
Conceptualization	–	246
Content creation	–	104
Copilot posting	1	514
Design	–	1,010
Development	–	1,042
First2Finish	–	2,477
Specification	–	237
Test Scenarios	–	207
Test suites	7	124
UI Prototype	7	1,212
Other	–	95
Total	44	13,602

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 crowd developers. A great deal of extra work was necessary to ensure that this information was made explicit in the requirements specification for the external crowd developers. A total of 1,061 pages of specification were written by TPI for their 44 Topcoder competitions. This contrasted with the belief that almost no extra documentation would have needed to be written if the development was done in-house. The architect liaising with Topcoder 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

Table 6 lists the distinct number of registrants and submitters for both the case study and our overall Topcoder platform sample. The table shows that the "crowd" registered for all TPI competitions consists of 182 distinct registrants, though there were only 37 distinct submitters. We observe that this low percentage of registrants who eventually submit is fairly consistent with the overall platform level. The number of distinct submitters at 4,516 represents a relatively small portion of the number of developers who registered for at least one of the 13,602 competitions in our data set. Our data set includes only the 20,747 Topcoder members who were involved in the 13,602 competitions in our data set. We cannot draw conclusions about the other registered members on Topcoder as we found no activity to report in our data set.

The Topcoder competition-based approach effectively represents a waterfall approach to software development. TPI, however, were using an agile development process based on Scrum. Combining these different waterfall and agile development processes was problematic. Development contributions from Topcoder had to be assigned to a Scrum team within TPI, and crowd contributions had to be

TABLE 6
Distinct Registrants and Submitters

	Case study	Topcoder sample
Distinct registrants	182	20,747
Distinct submitters	37	4,516
Submission rate	20.3%	16.3%

subsequently injected into the appropriate sprints. A TPI 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 Topcoder when we tell them one thing in one competition, but have changed our mind in the next competition."

There were also quite a number of layers in the engagement model between Topcoder and TPI. First at the Topcoder end, a co-pilot liaised between the crowd developer community on the one hand and TPI personnel on the other hand. Furthermore, a Topcoder platform specialist was involved in liaising with TPI and overseeing the co-pilot and recommending changes at that level.

Within TPI, the choice of personnel to interact with the Topcoder co-pilot was a difficult decision. While Topcoder would prefer a single point of contact within the customer organization, there were significant management and technical issues involved, thus requiring a great deal of dedicated resources from TPI on both the management and technical end, some at a very senior (and thus costly) level. A senior Topcoder Program Manager was appointed within TPI specifically for all programs being developed with Topcoder. This program manager ensured that management were aware of any scheduling issues that could arise, for example, and also ensured that training was provided. A specific Titan Program Manager was also appointed in TPI, and inevitably there was some overlap between this role and the previous one.

On the technical side, a Senior Architect was allocated at TPI to coordinate the Topcoder development for the Titan project. This role of Topcoder liaison who had daily contact with the Topcoder community was considered to be problematic within TPI, given the considerable pressure to answer questions in a timely fashion. 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, the liaison role involved answering questions on the Topcoder 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

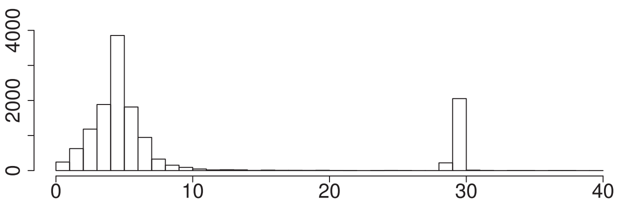


Fig. 2. Distribution of competition duration in days for the Topcoder platform sample (x-axis trimmed to improve readability).

delivery date for crowd development would be pushed out. Also, it was quite tedious to answer questions using the narrow communication channel of the chat forum. The architect estimated the time answering questions on the Topcoder 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."*

Another structural coordination issue arose in that TPI allocate architects to products, and the desire to get the Topcoder project 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 crowd 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 significant extra coordination overhead and duplication of work on the project in that two teams had to become familiar with the project context and content, and related 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 the crowd.

In contrast to distributed development which involves other developers from the same organization, the only relationship which tended to build over time was that with the Topcoder co-pilot. There was no real opportunity to build up a relationship with any of the crowd developers, as interaction was filtered through a number of layers.

4.3 Planning and Scheduling

The Titan project comprised more than fifty Topcoder competitions, of which 44 were successfully completed. From the customer's perspective, these competitions involved a total of 695 calendar days during a period of approximately eight months (for each competition, we counted the number of days for all competitions, some of which were run in parallel, yielding a grand total of 695). The competitions had an average duration of just over 13 days, which includes the time needed for review of the submissions. The shortest completion time for a competition was four days while the longest competition took 32 days to complete. The actual competition duration from the developer's perspective was considerably shorter, as this would be the difference between the registration deadline and the submission deadline. Fig. 2 presents the distribution of competition duration in days for the Topcoder platform sample, with a major peak around a week, and a smaller peak around one month. The average duration for just the TPI competitions was 5 days and 17 hours ($\sigma = 21\ h$), which is very close to the mode of the duration (see Fig. 2). This is not significantly shorter than the average of 9 days and 7 hours for the whole sample (Mann-Whitney U, $p = 0.106$).

Some of the specific timings and the granularity of possible decisions for crowd development were somewhat problematic for TPI. For example, Topcoder 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 on time internally. A further difficulty arose in that deliverables must be accepted as a whole, or rejected as a whole, with no middle ground, even for submissions with minor defects. 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 crowd developers from bidding on future competitions, there was a tendency to accept all submissions, even those 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 competition to finish, while the main application was evolving, causing possible integration issues. TPI's schedule was also jeopardized as two of its competitions failed due to a lack of submissions.[1] These competitions 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 can be seen from Table 6 above, this seeming discrepancy between registering for a competition and actually submitting is not uncommon. Fig. 3 presents a scatter-plot of the number of registrants versus the number of submissions for the competitions in our sample.

As already discussed, TPI perceived the need to run multiple competitions in parallel so as to shorten the development time. However, this clearly had implications for managing and coordinating the handling of submissions by TPI. For example, there were interdependencies between the deliverables produced in the various competitions. This also led to duplication of functionality in some of the code.

1. The reasons are unknown, which was in fact one of the motivations for conducting the quantitative comparison with other Topcoder competitions in the first place.

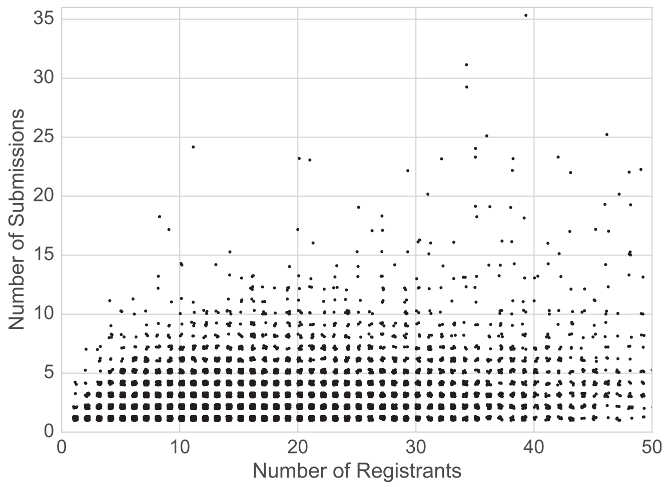


Fig. 3. Correlation between registrants and submissions for the Topcoder platform sample.

Fig. 4 presents the number of active challenges for TPI as well as the total number of active challenges on the Topcoder platform. TPI held several competitions in parallel, up to five competitions during August 2013. The total number of active challenges on Topcoder varied between 20 and approximately 75 during the same time period. Running competitions in parallel should allow a customer to get work done more quickly, but may 'dilute' the available workforce for a specific competition as developers may have to choose which competition to work on at any given time.

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 [96]. However, the structure of the Topcoder development process made it difficult to preserve this, as it shifted QA issues towards the back-end of the development process, after coding had 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."*

There was also a problem with lack of continuity. Crowd developers do not remain idle at the end of competitions, and may thus not be available for subsequent TPI competitions. In fact, TPI experienced problems with bugs which had previously been identified and fixed, but were re-introduced after the code went back for further development with the crowd, as inevitably different developers would work on the code and no organizational learning in the usual sense was taking place. This added to the critical perception expressed by TPI's Divisional CTO, when he characterized the interaction with crowd developers as *"a fleeting relationship"*:

> *"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."*

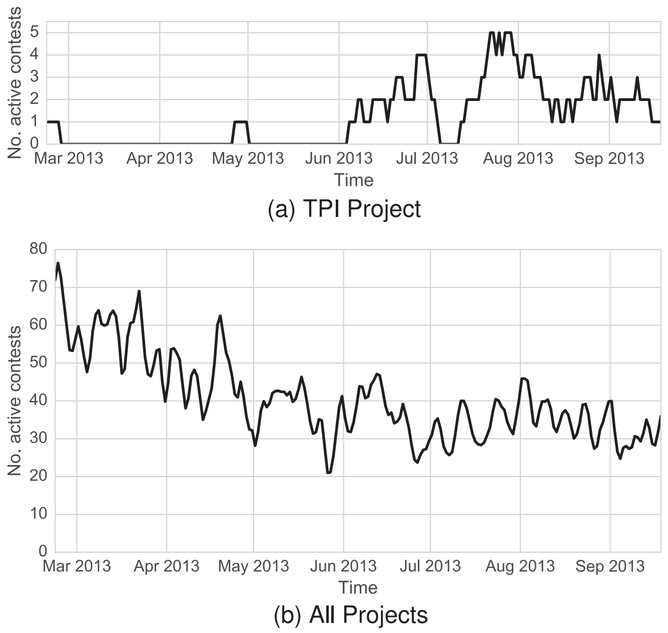


Fig. 4. Number of active (parallel) TPI challenges during TPI's project and all challenges during the same period.

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 crowd development community both in development and as a final test or demonstrator for code they developed. Prior to this, code testing was done with stubbed-out service calls to the back-end, but there was a concern within TPI that code delivered by crowd developers 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 the crowd, there were some quality issues, for example, 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 Topcoder 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 Topcoder 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.

Once a competition's submission deadline has passed, all submissions are reviewed and given a rating between zero and 100. Fig. 5 shows the distributions of the mean score (of all submissions) and the maximum score for (a) all TPI competitions, and (b) all competitions in our overall sample. Fig. 5a shows that the maximum score of most competitions is 100, with a limited tail down to approximately scores of 75. Only three of TPI's 44 competitions resulted in submissions that scored 100, with the remaining competitions resulting in submissions with scores between 78 and 100. When comparing the mean scores, TPI's competitions do not differ significantly from our overall sample (Mann-

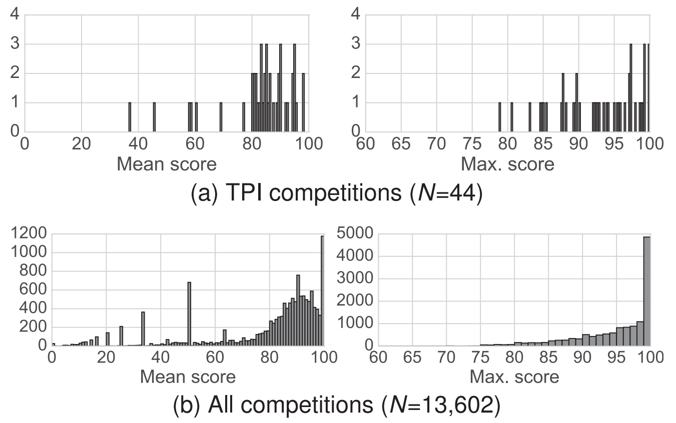


Fig. 5. Comparison of mean (left) and maximum (right) scores for TPI challenges and all challenges. The scale of figures on the right are trimmed to improve readability.

Whitney U test, $p = .2335$), though they do in terms of maximum score (Mann-Whitney U test, $p = .0005$), with the scores of TPI's competitions varying to a higher degree.

4.5 Knowledge and Intellectual Property

The "fleeting relationship" mentioned earlier also has consequences for knowledge management and IP. 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. Specifications are typically not available to Topcoder members unless they register for a competition. Table 7 shows the total number of registrants, and the total number of submissions per competition type. The table shows that there were considerable numbers of potential participants (each of whom would have access to the competition specifications), but that the number of submissions was significantly lower—almost 90 percent of those registered for a competition did not actually submit anything to that competition. 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 Topcoder platform. This was to obfuscate the fact that the work was for TPI, who are a major global player in the ICT sector. TPI suspected that developers from competing organizations might be working as crowd developers in their spare time. TPI took advantage of the standard Competition Confidentiality Agreement (CCA) which Topcoder 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 crowd developers. However, TPI were still concerned about the extent to which proprietary information may be exposed in competitions.

TABLE 7
Total Number of Competitions per Type, Registrations, Submissions, Unique Registrations and Unique Developers, and Average Submission Rates for TPI Contests

Type	No. comp.	Total reg.	Total subm.	Unique registr.	Unique devel.	Subm. rate
Co-pilot	1	13	6	13	6	46%
UI prototype	7	99	22	40	9	22.2%
Architecture	6	90	12	35	5	13.3%
Assembly	23	610	51	119	12	8.4%
Test Suite	7	92	15	44	5	16.3%
Total	44	904	106	182	37	11.7%

Fig. 6 lists the Top 10 countries based on Topcoder membership and submissions, that is, which countries host crowd developers that make most submissions. Most Topcoder members originate from India, China, and the USA, with most submissions coming from the same three countries, though in a different order with most submissions originating in China. These findings correspond to a study by Dubey et al. [122], though based on a smaller sample of competitions, they found that most crowd developers originate from India, with China in second place. Clearly, a ban on developers from some of these countries would limit the potential contribution that can be gained from the Topcoder platform.

4.6 Motivation and Remuneration

Given a potential development community of well over one million members, Topcoder would claim to have broad and deep enough expertise to ensure a healthy competition rate. However, TPI had to cancel some competitions due to a lack of submissions. Furthermore, 10 of the 44 competitions attracted only a single submission. The fact that TPI used a pseudonym may have been significant in that well known companies seem to attract crowd developers more readily and TPI would certainly be a very well known company globally. One motivation for crowd developers to participate is to learn new skills, but also to improve their track record—the ability to list working with blue chip companies is likely perceived to be more attractive than anonymous companies that use pseudonyms.

To investigate a potential cause of the lack of submissions, we also analyzed the number of active members in the crowd. Defining what an 'active' member is, is not

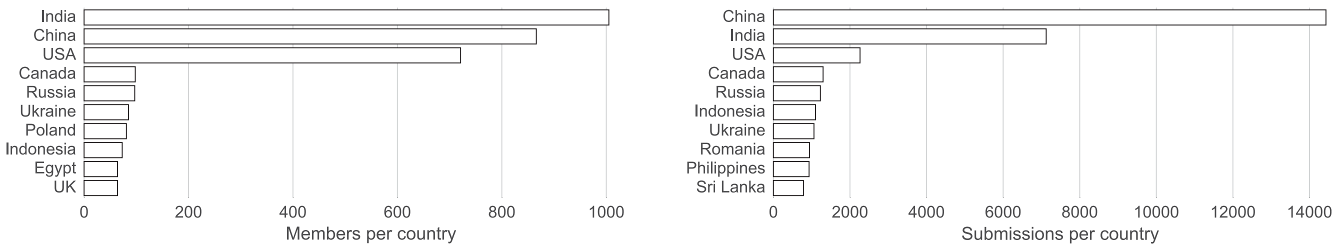


Fig. 6. Top 10 countries based on number of TC members and origin of submissions.

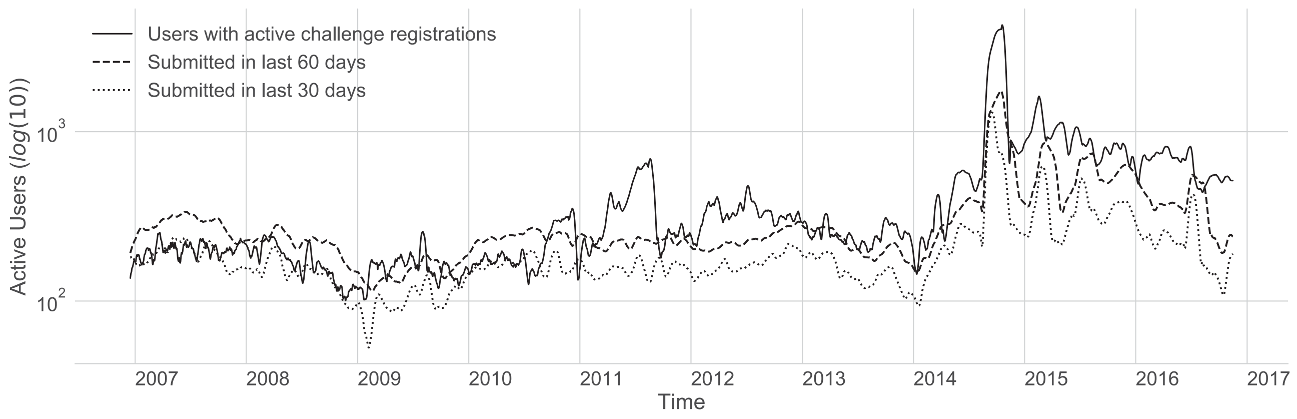


Fig. 7. Size of the active crowd based on competition registrations, or submitted in the last 30 and 60 days.

straightforward. Fig. 7 presents data using a number of threshold values. First, we defined it as all members that have registered for a competition at a given time. Using this definition the size of the active crowd varies between 150 and 700 members. A different definition is to count those members who have *submitted* in a given past period—we calculated this using two different time frames: 30 and 60 days. Using the most stringent definition, counting only those members who submitted in the last month, the size of the (active) crowd varies between approximately 20 and just over 1,000.

Fig. 8 plots the cumulative proportion of submissions against the most active developers (in decreasing order). This analysis only considers members that have submitted at least once—our data set includes 4,516 such members. The figure shows that approximately 80 percent of the submissions were submitted by the Top 15 percent most active developers, while 50 percent made approximately 95 percent of the submissions. These numbers are somewhat reminiscent of the results of a study of the open source Apache webserver by Mockus et al. [107]. Mockus et al. hypothesized that approximately 80 percent of development in open source projects was done by the Top 15 contributors. Those findings pertained to one specific open source project, which differs from our study as our data set comprises over 13,600 competitions. Thus, while Topcoder has over one million registered developers who potentially could perform work, our study suggests that many of them do actively participate.

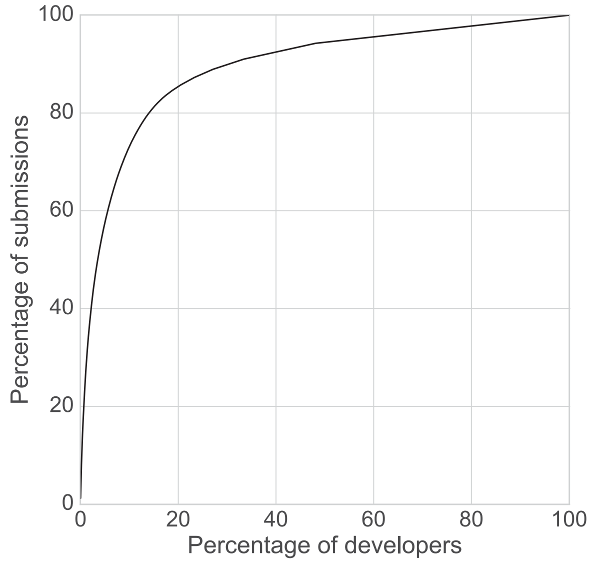


Fig. 8. Cumulative distribution of submissions by developers. Approximately 80 percent of submissions are made by the Top 15 percent most active developers.

The Topcoder pricing structure was quite elaborate. At the top level, there was a monthly platform fee to Topcoder. For TPI this was a monthly fee of $30,000. This allowed access to the Topcoder component catalog containing more than 1,500 software solutions. Topcoder estimates that approximately 60 percent 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 Topcoder and TPI typically cost $600 per competition. There was an initial specification review before the competition began, and this cost $50. The individual competition pricing was also quite complex. In the case of TPI, first prizes for competitions ranged from $200 up to $2,400, depending on the size and complexity of a competition. A second prize of 50 percent of the first prize was paid to the runner up in each competition, but this prize would only be paid if the quality rating of the submission was 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 percent of the first prize, depending on the past successful track record of the winning contestant (i.e., his/her reliability—in terms of whether a contestant actually submits to a competition having registered for it). In addition, there was a cost of 45 percent of the first prize to support the Topcoder Digital Run, an initiative whereby Topcoder share money with the Topcoder development community based on the monthly competition revenue and proportional to the number of points that crowd developers have amassed

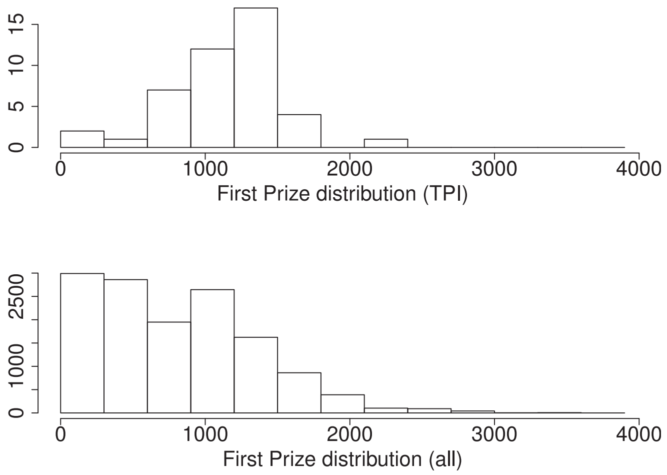


Fig. 9. Distribution of First Prize amount (bin size $300; x-axes trimmed to improve readability).

in competitions. 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 competitions, reviewers from the crowd community evaluated submissions at an average cost of approximately $800. Finally, Topcoder charged a 100 percent commission equal to the total development costs above. Overall, the total average cost per competition for TPI was approximately $7,200 (excluding the monthly platform fee).

Fig. 9 presents a distribution of the First Prize reward (to improve readability, we set the histogram's bin size to $300). (As indicated above, many other costs are derived as a percentage of the First Prize amount, e.g., the prize money for the Second Prize runner-up is 50 percent of the First Prize amount).

The remuneration offered does not follow a continuous distribution, but tends to be concentrated in "round" numbers. For example, we found peaks at the $500, $750, and $1,000 marks, with $1,000 being the most common amount being offered. The sum of First Prizes of all TPI's competitions was US $51,425. When including the additional expenses (to cover Second Prize, Digital Run, Reliability Bonus, etc.), the total cost to run these competitions was quite significant. A further significant additional cost arises in the extra senior development personnel that were allocated to the project. We conducted a χ^2 test to evaluate the similarity of the distribution of the First Prizes of TPI contests versus the Topcoder sample; in this case, we found that the distribution of the first prizes is not similar to the distribution of the larger Topcoder sample ($p < 0.001$).

4.7 Summary of the Case Study Findings

The goal of the case study was to gain greater insight into a customer's perspective on crowdsourcing software development, as this aspect had not featured in previous research. Table 8 summarizes the key findings.

In comparison with traditional development in-house, the TPI Program Manager was of the opinion that crowd 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 Topcoder 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. While it is not possible to quantify the mismatch between expectations and realization, overall, the TPI staff members involved in the Titan project were not convinced that crowdsourcing the project had been effective in terms of their initial goal, namely to speed up development and leverage external expertise.

TABLE 8
Summary of Key Findings of the Case Study

Theme	Key Findings
Task Decomposition	44 successful competitions primarily for front-end development, using HTML5 which is the 7th most popular technology in our sample of competitions.
Coordination & Communication	Several layers of communication, making communication very cumbersome. Answering questions was found to be very time-consuming and required senior (and thus costly) staff members. TPI's competitions attracted over 900 registrants in total.
Planning & Scheduling	Challenging to finish internal reviews of submissions in time. Tendency to accept submissions so as not to deter developers in future competitions. Up to 5 TPI competitions running in parallel, and 20-75 projects competitions in parallel in our sample during the same period.
Quality Assurance	Development process focuses strongly on requirements, leaving QA activities until later. Lack of continuity (fleeting relationship) prevents crowd developers from building up domain knowledge and experience in the project.
Knowledge & Intellectual Property	Competitions may attract considerable interest from the crowd, but this does not automatically lead to many submissions. However, 1,061 pages of detailed specification were available to all registrants.
Motivation & Remuneration	Rewards offered in TPI competitions are representative of platform sample. Despite considerable interest (number of registrations), the actual participation rate is limited. The size of the active crowd is very small in practice (37 unique developers).

In our presentation above, we included analyses of a large set of 13,602 competitions from the Topcoder platform in order to contextualize the case study findings. Based on this comparison, we conclude that the case study can be considered a typical and representative case—that is, it used a very popular technology (HTML5, ranking 7th most popular technology—see Fig. 4), offered rewards that seem inline with other competitions, and did not stand out in terms of the size of the crowd involved in TPI's project when compared to projects in general. It is important to understand these contextual factors, as perhaps they could help explain why TPI's experiences were disappointing. For example, if TPI offered rewards that were systematically lower than other competitions, one could suggest that as a potential explanation for the limited crowd participation in TPI's competitions. However, the TPI competitions did not seem to differ in significant ways from other competitions. In order to gain a better understanding of crowdsourcing software, we

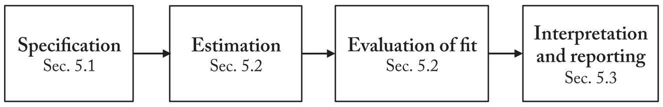


Fig. 10. Steps in implementing SEM (adapted from Hoyle [115, pp. 7]).

developed a theoretical model, drawing on the key insights generated through the case study. This is presented next.

5 Theory Development and Evaluation

Based on the case study findings and the extant literature on crowdsourcing, in this section we present our theory development and evaluation approach using Structural Equation Modeling [115]. The main steps of SEM are shown in Fig. 10. The first step is specification, referring to the derivation of a set of hypotheses, implemented as a structural equation model– - together they form our theory, and this is presented in Section 5.1. The second step is estimation of the model parameters using an estimation algorithm, detailed in Section 5.2. That section also reports on the third step: evaluation of fit of the model. The structural equation model is tested on a large quantitative data set from Topcoder. The last step is to interpret and report the findings (Section 5.3).

5.1 Theory Development and Model Specification

The first step in SEM is to *specify* a theoretical model, that is, to define a set of hypotheses. Drawing from both previous literature on crowdsourcing and our exploratory case study reported in Section 4, we formulate a number of interrelated hypotheses (see Fig. 11). Following SEM notation conventions, rectangular boxes represent observed variables, and arrows represent relations between variables [123]. In particular, we aim to develop a better understanding of some of the factors that might affect a crowd's interest and participation in competitions. Attracting sufficient people to register their interest and submit solutions in a competition is key to the success of crowdsourcing software development.

When customers advertise competitions on the Topcoder platform, a competition is part of an overall project that is owned by that customer. For example, all of TPI's competitions belonged to the same project which is indicated by a project identifier in the data set. Customers may choose to run several competitions in parallel—for example, the case study company, TPI, ran several competitions in parallel (see Fig. 4). Decomposition of a project into many smaller tasks may reduce its complexity [124]. However, when doing so, the available workforce may be limited, as crowd workers may not be able to take on several competitions at the same time. This is exacerbated by the fact that the majority of competitions have a relatively short duration (see Fig. 2): with a short deadline for a competition, a developer already working on one competition may not have time to work on another competition in parallel. Therefore, when a customer runs several competitions in parallel, we expect a reduction in the crowd's interest. Thus, we hypothesize the following:

Hypothesis 1 (H1). *Running competitions in parallel within a project is negatively associated with the interest from the crowd for that competition.*

In open source contexts, financial incentives have been found to "crowd out" intrinsic motivations [125]. Although

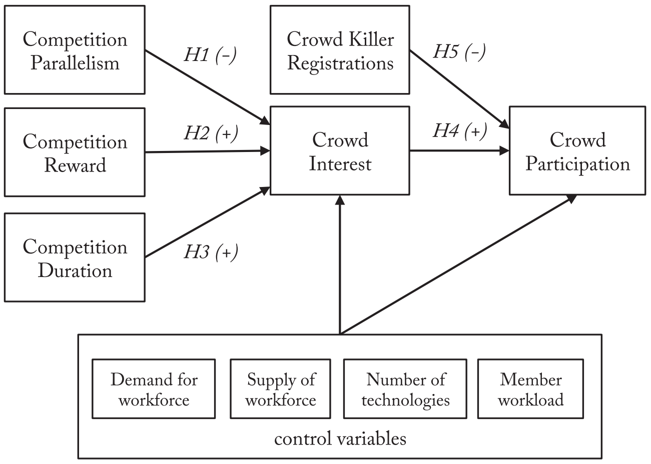


Fig. 11. Proposed theoretical model.

crowdsourcing shares some similarities with open source, it is quite different in that the fundamental premise of crowdsourcing is that an incentive or prize can elicit potential solutions to specific problems. This can range from micropayments for performing fairly mundane and trivial tasks to millions of dollars for solving intractable problems in the bio-medical and pharmaceutical sector [11]. Not surprisingly, a significant amount of research has been conducted on setting optimal prize levels, and their relationship to the effort required [60], [126], [127]. Brabham [72] reports the desire to make money as one of the strongest motivators in the iStockphoto community. Lakhani and Panetta [12] also report the significant extrinsic motivator that the cash prize represents in software development contexts, which is confirmed in the specific case of Topcoder [34]. This relationship was also borne out in our case study as one of the key tasks of the co-pilots was to estimate the prize level necessary to attract submissions to a particular challenge. This leads to the following hypothesis:

Hypothesis 2 (H2). *The reward for a competition is positively associated with an increase in the interest from the crowd for the competition.*

The literature on software development project management generally suggests that a longer duration can lead to greater software development productivity [128]. A number of research studies have identified competition duration as an important factor in crowdsourcing [124], [127], [129], [130], [131]. The allotted duration for a competition on Topcoder defines the maximum possible time for software development. Clearly, a competition which has a very short duration may not attract developers as they will not have time to complete the task. Developers may also work in weekends. Thus, they will need a certain amount of time to become aware of competitions and to assess whether they have the required skills to accomplish the task. This leads to the following hypothesis:

Hypothesis 3 (H3). *Competition duration is positively associated with crowd interest in the competition*

In numerous situations, intention to act has been found to precede actual behavior, an issue that has been thoroughly investigated in the social psychology and behavior science literature [132]. The *Theory of Reasoned Action* [133], [134] suggests *intention* as a mediating construct between attitude on the one hand and performance on the other.

We therefore suggest that registering for a competition indicates an *intention*. Therefore, we posit that registering is a good predictor of submitting to a competition. Previous research into the volunteering process in crowdsourcing also suggests a series of commitment stages which include registering interest prior to submitting [135]. This is also borne out in our case study, as co-pilots would suggest an estimate of the number of potential submissions based on the number of registrants. This leads to the following hypothesis:

HYPOTHESIS 4 (H4). *Interest from the crowd is positively associated with participation in the competition.*

Clearly, there will not be a one-to-one correspondence between the number of registrants and the number of submissions. Some registrants may be unable to accomplish the given task, for example. However, the literature suggests that the reputation of certain competitors, those who have achieved a high ranking on Topcoder due to their previous successes, can deter other registrants who do not actually submit for competitions in which they have registered because they believe they have no chance of winning [34]. We label the former as "crowd killers" in our model as they deter other competitors in the crowd from registering. This phenomenon is also evident in our case study. We see that runners-up receive 50 percent of the first prize, but even more tellingly, the Topcoder Reliability Bonus is designed to reward registrants who have a past history of submitting an entry that passes the minimum quality review threshold for competitions in which they have initially registered. This serves to act as an incentive to actually submit in competitions where contestants have registered, even if they think they have no chance of winning the overall prize. This leads to the following hypothesis:

HYPOTHESIS 5 (H5). *"Crowd killers" will be negatively associated with participation in the competition.*

We also include a number of control variables in Fig. 11. We include a demand type factor ("demand for developers") as the number of other active competitions in parallel during a competition, as this will dilute the developer pool—if there are more competitions to join, this has the potential to reduce the overall number of registrations and submissions in individual competitions. Developers who initially had intended to participate in a given competition might be distracted by other active competitions.

The growth of the pool of available developers appears to have dramatically increased over the history of Topcoder, from 50,000 members in 2004 to over one million members in 2016. We model this as "supply of workforce" as it represents the *available* developer pool, which can have an impact on the number of registrations and submissions. A larger pool of potential developers may lead to a significant increase in registrations and participation in competitions. While most members of the community have never registered for any contest, these members have been referred to as the " *'latent pool': people who were interested enough in the Topcoder platform to register and had the potential to provide Topcoder with increased development under the right conditions*" [21, p. 4]. Including this as a control variable ensures that this effect is considered.

The "number of technologies" might be a factor which would influence the number of registrants and submissions. While it is clear that we cannot state anything about the complexity of competitions without inspection of the specifications of each competition, the number of technologies that is involved is a crude indicator of the knowledge that is required from developers.

Finally, we considered the "workload" of members, which is the number of contests that registrants are already working on.[2] When a new contest is advertised (i.e., its registration is opened), members may already be submitting in other contests, which is to say, they already have a certain workload. This may reduce their interest in the newly advertised contest, but it might also shift their attention from a current contest to the new contest, for example when the new contest is more attractive.

Table 9 lists the definitions of the variables that are used to operationalize the constructs in our hypotheses. In the remainder of this section, we evaluate the fit of our model to the data (Section 5.2), after which we evaluate the hypotheses (Section 5.3).

TABLE 9
Variable Definitions

Construct variables	Description
Competition parallelism	The number of competitions that are run simultaneously within the same project (where all competitions within a given project belong to the same customer).
Competition reward	First Prize money offered for a competition.
Competition duration	Number of days between the registration deadline and the submission deadline (included).
Crowd Killer registrations	Developers whose previous win count is $>$ (Mean no. of wins + 3 SD).
Crowd interest	Number of registrations for a competition.
Crowd participation	Number of submissions. Only registered members are able to submit
Control variables	**Description**
Demand for workforce	At a given time, the number of competitions that are running at the time of a competition being advertised.
Supply of workforce	The number of platform members at the time of a competition's advertisement; most members are not active (the "latent pool").
Number of technologies	The number of technologies that are specified for a competition.
Member workload	For a given contest c, the average number of submissions that developers registered for c make to other contests.

5.2 Estimation and Evaluating the Model Fit

After specifying the structural equation model, the next step is to *estimate* the model and *evaluate its fit*; that is, the parameters of the model are estimated by a SEM software program, and the generated model fit indexes can be evaluated (see Fig. 10). Estimation is done using an estimation algorithm, with Maximum Likelihood being the most common estimator by far. However, ML assumes multivariate normality of the data [119], [136]. Violating this assumption may lead to incorrect results [137, p. 68]. Therefore, it is important to examine the distribution of the data. Kitchenham et al. [138] recommend the use of kernel density plots instead of box plots—we included kernel density plots for all variables used in our model in Appendix B, which can be found on

2. We thank an anonymous reviewer for this suggestion.

TABLE 10
Model Fit Indexes

Model fit index	Value	Interpretation
χ^2	131.063	n/a
χ^2 Satorra-Bentler corrected	5.918	The Satorra-Bentler correction adjusts the χ^2 test statistic to account for non-normally distributed data
p value χ^2	0.205	Non-significant when $p > 0.05$, indicating the theoretical model supports the data
Degrees of freedom (df)	4	n/a
Corrected χ^2 / df	1.4795	Ratio of ≤ 2 suggests a good fit
Root Mean Square Error of Approximation (RMSEA) (90% CI)	0.048 (0.041, 0.056)	Values ≤ 0.05 suggests close fit; ≤ 0.08 indicates acceptable fit; ≥ 0.10 suggests poor fit
Robust RMSEA (90% CI)	0.028 (NA, 0.072)	
p value RMSEA $\leq .05$	0.635	Probability that RMSEA $\leq .05$; higher is better
Comparative Fit Index (CFI)	0.996	Values over 0.90 or 0.95 indicate a good model fit
Nonnormed Fit Index (NNFI) a.k.a. Tucker-Lewis	0.983	Values ≥ 0.90 indicate a good model fit
Standardized Root Mean Square Residual (SRMR)	0.007	Values ≤ 0.05 indicate a good model fit

the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2017.2774297. As the density plots show, none of the variables have a normal distribution, which can lead to an inflated Chi-square (χ^2) statistic, from which many other fit indexes are derived. There are several alternative techniques to work with non-normally distributed data in SEM—in our study we have used two such techniques. First, we used Robust ML estimation [139] which leads to the same parameter estimates as ML, but the χ^2 estimates and standard errors are robust to non-normality [140]. We used the Satorra-Bentler correction [141] which adjusts the value of χ^2 (see Appendix C for details, available in the online supplemental material). The data set we used was complete, hence we did not have to consider this in selecting an appropriate estimator. In addition to using the Robust ML estimator, we also used an alternative technique to deal with non-normally distributed data. We used the default ML estimator, but rather than relying on the default generated standard errors (which would be incorrect given the non-normality of the data), we used the Bollen-Stine bootstrap procedure to calculate standard error values [142]. Instead of the Satorra-Bentler correction, an alternative p value is calculated, based on the Bollen-Stine bootstrap procedure. This p value was 0.911, which is well over the cut-off value of 0.05, and thus the results of this approach also supported our model. (Appendix C provides further details, available in the online supplemental material.)

Numerous indexes of fit have been proposed to evaluate structural equation models. One critique of many studies is that they report only a single fit index [118]. Following guidelines on reporting SEM [118], [119], [143], we discuss several fit indexes that are commonly reported, and also discuss how our model scores on these indexes. It is important to note that there is no general consensus regarding the cut-off points for most of these indexes, a point we address in more detail below. Table 10 summarizes the fit indexes.

A common method of evaluating goodness-of-fit is χ^2, with low values suggesting a good fit. Alternatively, the ratio of the χ^2 and the degrees of freedom (df) was suggested by Wheaton [144], with a ratio of smaller than 2 suggesting a good fit. For our model, this ratio is 1.480.

There is growing consensus that one of the most useful indexes is the *Root Mean Square Error of Approximation* [145], proposed by Steiger and Lind [146]. An RMSEA value of 0.05 indicates a “close fit” [147], a value between 0.05 and 0.08 an “acceptable” [147], [148] fit, and a value of over 0.10 indicates a poor fit. The RMSEA of our model is 0.048, which suggests a good fit, with a 90 percent confidence interval (CI) between 0.041 and 0.056. The robust RMSEA scores better at 0.028; while its 90 percent confidence interval is slightly wider, the upper limit is still acceptable as it is below the cut-off of 0.10 beyond which a model would be considered a poor fit.

A major issue with the χ^2 measure is its sensitivity to the size of the sample [149]. The *Comparative Fit Index* (CFI) was developed by Bentler to overcome such limitations [148], [150]. Recommended cut-off values vary from 0.90 to 0.95 [147], [148]. The CFI for our model is over 0.99 indicating a very good fit.

The Nonnormed Fit Index (NNFI), also known as the Tucker-Lewis Index (TLI) is another fit index. Hoe [148, p. 77] recommends a cut-off value of 0.90—our model has an NNFI of 0.98 suggesting a close fit. The Standardized Root Mean Square Residual (SRMR) is another recommended index of fit to report [151]. The SRMR for our model is 0.007, which indicates a good model fit.

Hu and Bentler suggest the use of combinations of indexes [147] because no single index can represent all fit dimensions of a model. In particular, they suggest that a CFI of 0.96 (or higher) combined with an SRMR of 0.09 (or smaller) indicates a good fit. Another combination is a TLI of close to 0.95 in combination with an SRMR cut-off close to 0.09. Our model complies with both combinations.

As recommended in the technical literature on structural equation modeling, we reported multiple indexes of fit [119]. West et al. suggest that the theoretical model is supported by the data when *“a majority of the fit indices”* indicate an acceptable model [151]. Thus, based on the fit indexes reported above, we conclude that our theoretical model is supported by the data. However, it is important to note that our model is not the only viable model, and that alternative models may exist; this is a point often overlooked in literature presenting SEM studies. Also, we wish to remind the reader that

TABLE 11
Means, Standard Deviations, and Correlations (Spearman)

	Mean	SD	1	2	3	4	5
1. Parallelism	4.06	16.50					
2. Reward	886.30	1,794.34	-0.283*				
3. Duration	9.29	9.70	0.246*	-0.005			
4. Interest	18.37	31.36	-0.258*	0.374*	-0.003		
5. Participation	2.99	14.66	-0.076*	-0.187*	-0.109*	0.240*	
6. Crowd killer	1.54	1.47	-0.195*	0.220*	-0.177*	0.230*	0.002

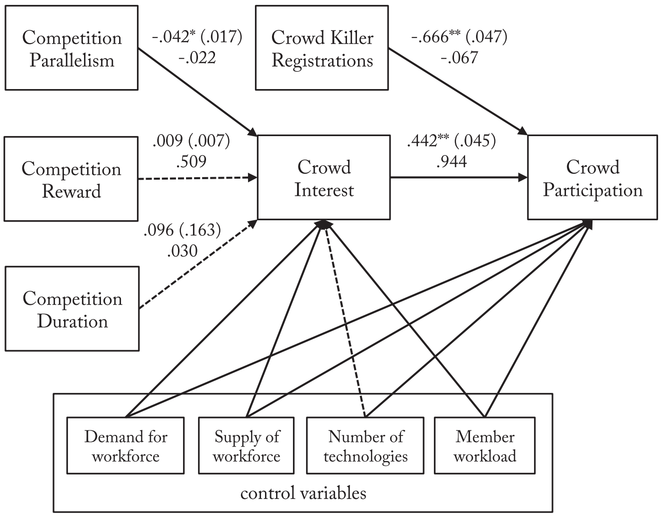
$^* p < .001$

although a good model fit is important, a prerequisite theoretical grounding of the model is equally important.

5.3 Hypothesis Testing and Interpretation

On the basis of the various model fit indexes, we have considerable confidence that our proposed model is a plausible one. It is important to note that such support does not imply our model is optimal—it is simply *a* model. The theoretical model is a good fit with the data, which is a prerequisite for being able to interpret the parameter estimates. Following reporting guidelines for SEM [119], [152], we list the constructs of our model, correlations, means, and standard deviations in Table 11 (as the data were not normally distributed, we used the non-parametric Spearman's ρ correlation coefficient). Fig. 12 presents the research model with the estimated coefficients and standard errors, and Table 12 provides the full set of parameters, including those for the control variables, as well as confidence intervals. Ideally, modeling variables would be uncorrelated, yet the table shows that several of them exhibit statistically significant correlation (e.g., parallelism and reward). This is normal in real domains because many domain phenomena influence several variables at once. Kenny pointed out such relationships *"can vanish"* after controlling for other causal variables, and the belief that *"if X does not cause Y, they should be uncorrelated"* is naive [153, p. 80]. A summary of the results of the hypotheses is presented in Table 12.

We found statistically significant evidence to support three out of five hypotheses—we discuss the statistical and practical significance below. The level of parallelism within a project is statistically significantly negatively correlated with the crowd's interest (*H1*). The standardized coefficient is -0.022, which suggests a very limited effect. There is no statistically significant correlation between the offered reward and the crowd's interest in a competition (*H2*). The confidence interval is quite wide and includes zero, which is why no statistical significance was found. This is surprising, but we note that the standardized coefficient is 0.509, representing a medium effect size. We observe that the sign of the coefficient is as we hypothesized. Furthermore, we found no evidence that a competition's duration is significantly positively correlated with the crowd's interest (*H3*). In this case, we found that the standardized coefficient is only 0.030 which is very modest. Again, we observe that the sign is positive as we hypothesized. The data do support *H4*: a crowd's interest in a competition is significantly positively correlated with the level of participation (i.e., number of submissions). The standardized coefficient is very large at 0.944. Finally, we also found evidence for the crowd killer



Coefficients for control variables are listed in Table 12

Fig. 12. Results of the research model, with unstandardized parameter estimates, standard errors (in parentheses), and standardized estimates ($^*p < 0.05$, $^{**}p < 0.001$; dotted lines indicate not statistically significant relationships).

concept (*H5*): there is a significantly negative correlation between the top-winning members' interest in a competition and the level of participation as a whole in that competition. While statistically significant, the effect size is quite limited, with a standardized coefficient of -0.067.

Table 12 also shows the standardized path coefficients of the correlations of the control variables. In terms of crowd interest (as reflected by number of registrations) and workforce demand, we found a significant negative correlation, that is, the number of other active, "competing" competitions was likely to decrease registrations. However, the effect size was quite small at -0.051. In relation to workforce supply, we found a significant positive correlation between the number of platform members and registrations, with a moderate effect size at 0.135. However, we found no significant correlation between the number of technologies specified in a competition and the crowd's interest in that competition; the effect size was also very limited at -0.011. We also observed a significant correlation between member workload and crowd interest; that is, as members were already working on other submissions, this may have reduced their interest in other competitions, although this effect was very small at -0.043.

In terms of crowd participation (that is, actual submissions), we investigated the same control variables. The demand for workforce (i.e., the number of concurrent active challenges that "compete" for the available workforce) has a significant positive correlation with participation in a competition, though the effect size was very small at 0.075. The data suggest a significant negative correlation between the supply of workforce and actual participation (with a small effect size, -0.106). Likewise, we found a significant negative correlation between the number of technologies and participation, though with a very small effect size (-0.042). Finally, we also found a significant correlation between member workload and submissions (though with a very small effect size, 0.044). This, in combination with the negative correlation between member workload and interest, could suggest that some developers are very good at fast delivery or

TABLE 12
Parameter Estimations, Confidence Intervals, Standard Errors, and Standardized Coefficients

Paths	Unstandardized	95% CI	SE	*p*	Standardized
***H1*:Project parallelism → less interest**	-0.042*	(-0.076, -0.008)	0.017	0.016	-0.022
H2: Reward → interest	0.009 (n.s.)	(-0.005, 0.023)	0.007	0.225	0.509
H3: Duration → interest	0.096 (n.s.)	(-0.224, 0.416)	0.163	0.557	0.030
***H4:* Interest → participation**	0.442**	(0.353, 0.531)	0.045	0.000	0.944
***H5:* Crowd killers → less participation**	-0.666**	(-0.758, -0.574)	0.047	0.000	-0.067
Control variables					
Demand for workforce → interest	-0.026*	(-0.052, -0.001)	0.013	0.041	-0.051
Supply of workforce → interest	0.001**	(0.000, 0.001)	0.000	0.000	0.135
No. technologies → interest	-0.193 (n.s.)	(-1.070, 0.684)	0.448	0.666	-0.011
Member workload → interest	-0.219**	(-0.317, -0.120)	0.050	0.000	-0.043
Demand for workforce → participation	0.018**	(0.014, 0.023)	0.002	0.000	0.075
Supply of workforce → participation	-0.000**	(0.000, 0.000)	0.000	0.000	-0.106
No. technologies → participation	-0.348**	(-0.415, -0.282)	0.000	0.000	-0.042
Member workload → participation	0.104**	(0.073, 0.136)	0.016	0.000	0.044

* $p < 0.05$, ** $p < 0.001$, *n.s: not significant; statistically significant hypotheses are set in boldface.*

perhaps multitasking (these are highly productive developers), whereas others might focus on whatever contest they have registered for (these developers might be less productive). However, no further conclusions can be drawn about these correlations without further research.

Some of these findings are surprising, as one would expect that a larger available workforce leads to a higher level of participation. However, this only seems to lead to a greater number of registrations which do not follow through to actual submissions. Likewise, the 'dilution' of the available workforce as there are other challenges concurrently active would suggest a lower level of participation. Thus, it is not altogether intuitive to see this associated with a greater degree of participation. We discuss the results of our hypotheses as well as the limitations of our study in Section 6.

6 Discussion and Conclusion

6.1 Implications for Practice and Research

This article sheds more light on our understanding of participation in crowdsourcing software development from a customer's perspective, drawing on both qualitative and quantitative data.

The first finding is that the number of competitions run in parallel within a project has a significant negative effect on the crowd's interest in a competition (*H1*). This is not surprising as developers may simply not be able to work on different competitions in parallel, all the more telling since the size of the participating crowd of active developers is lower than possibly anticipated. However, this issue does have implications for customers who are seeking increased speed of software development through crowdsourcing. Due to the perception that the crowd with the requisite expertise is very large, companies will invariably choose to run several competitions in parallel. This was the case in our case study. However, this strategy will not be effective if the crowd cannot scale for parallel competitions. There is also the downside that the coordination of parallel competitions through the very narrow chat communication forum is quite frustrating and time-consuming, all the more problematic given that this communication tends to require a senior resource within participating companies.

The second finding is the reward offered for a competition did not have a *statistically* significant positive effect on the interest of the crowd in a competition (not supporting *H2*). This is particularly interesting as it seems to fly counter to the fundamental premise of crowdsourcing, namely that the crowd does it for a reward. The amount of reward is not a significant motivator, in so much as developers are not waiting for the reward to increase before they participate. Also, the reward amount is suggested by the co-pilot, who is an experienced member of the crowd community, and who therefore has a good intuition for the amount the reward should be set at, so as to be perceived as equitable and attractive to the crowd.

We found that a competition's duration did not appear to significantly affect the crowd's interest in a competition (counter to *H3*). Again, we can argue that this is somewhat similar to the case for *H2*. The average duration for competitions on Topcoder is 9 days and 7 hours, with peaks around 5 days and 30 days. In our data set, we found a fairly uniform distribution of submissions across all seven days of the week. The competition duration is decided by the experienced Topcoder co-pilot and given that Topcoder developers can also be in full-time employment, the average competition duration (9 days and 7 hours) allows for weekend work to be a possibility. One lesson for customers would be to partition work so as to fit these duration trends as they appear to be natural to the crowd community.

The crowd's indication of interest has a significant positive effect on the crowd's participation (*H4*). Obviously this is in keeping with what one would intuitively expect, and is also supported by the Theory of Reasoned Action—the higher the number of registrations, the higher the number of submissions. However, there is some attrition between the two. This could be explained by the fact that the developer only sees the full specification for the competition after registering, and that may surface some required capability that the developer does not actually possess. Also, after registration, developers may become aware of competition in the form of crowd killers, discussed next.

H5 suggested that what we have termed *crowd killers* reduce the participation in a competition. A number of high performing individuals earn a very good livelihood from

Topcoder—the highest paid crowd developer has received more than $1m [154]. However, this is not necessarily a bad thing for customers as they are likely to receive high quality contributions from such developers. Furthermore, experimental research on microtasks (not software development) suggests that a high number of participants in a competition might lead to lower quality. Whether or not this applies to a software development context remains to be seen and is an opportunity for further research.

6.2 Limitations

We are aware of a number of limitations of our study which we discuss below. Phase I of our study comprised an exploratory qualitative case study, which are inherently limited in the generalization of the findings, when considering the traditional, positivist meaning of the term generalization because this usually refers to *statistical* generalization [155], [156]. Our case study considered one specific project from a single customer organization, using a specific crowdsourcing platform—as such, the findings are not statistically generalizable. Walsham suggests a set of alternative forms of generalization that are more applicable to naturalistic case studies that are more appropriate to our case study [157]:

- *Development of concepts:* For example, our study identified the concept of "fleeting relationship," which is a characterization of the nature of the relationship between crowd developers and a customer organization.
- *Generation of theory:* Our study develops six key concerns for crowdsourcing software development, which together form a theoretical framework to study crowdsourcing.
- *Drawing of specific implications:* Our study identified a number of specific implications for TPI; for example, TPI's in-house development approach is agile, whereas the Topcoder process resembles a waterfall model. For TPI, this led to considerable rework, both in-house and in contests. Also, given TPI's specific existing technology stack, the company has been unable to benefit Topcoder's catalog of existing software components. Both findings have direct implications for TPI as discussed in Section 4.
- *Contribution of rich insight:* The case study presented in Section 4 presents a detailed account through "thick description" [103] of a customer organization engaging in crowdsourcing, which helps to brings to life a real-world phenomenon in a contemporary software development context.

Phase II of our study adopted a theory development approach. We derived a set of interrelated hypotheses, which we evaluated using structural equation modeling. The structural equation model was implemented in *R* using the open source SEM package "lavaan," which performs the estimation of the model parameters and generates indexes of model fit. We used a data set from Topcoder. SEM is a technique that helps in establishing relationships between constructs, but not necessarily causal relationships [119].

For our analysis, our data set only includes successful competitions because the success and failure of competitions was not the focus of our study. We suggest that future work can focus on the relation between parallelism, rewards, and duration on the success (or fail) rate of competitions.

In discussing the external validity or generalizability of our study results, we distinguish between the theoretical model and the data set. The theoretical model was developed by drawing from the crowdsourcing literature and the case study findings. While the case study was conducted by two of the authors [38], all three authors were closely involved in all stages of Phase II of the study. The theoretical model was discussed in great detail by all three authors. Two of the authors discussed which data to collect and store from the Topcoder platform. The data analysis and evaluation of the structural equation model was also conducted by all three authors and discussed in several meetings.

The generalizability of our findings is limited to the Topcoder platform, because our theory has not been tested using data from other crowdsourcing platforms. However, the theoretical model could also be evaluated with data from other competition-based crowdsourcing platforms that require registration prior to participation. We have focused specifically on competition-based crowdsourcing, as indicated by the "crowd killers" concept. Crowd killers are highly successful members of the crowd in terms of their win rate. These members may have established a reputation for their high success rate within the crowdsourcing community [154]. Crowd killers' interest in a given contest may discourage others from participating if they believe they no longer have a reasonable chance of winning. This construct only applies to competition-based crowdsourcing, and not, for example, to other platforms that act as open marketplaces such as oDesk (whose participation model is not based on competitions).

As reported, many indexes of fit have been proposed for structural equation models. While the fit indexes suggest that the model fits well with the data, it is important to note that these fit indexes should not be seen as absolute indicators of a good model as each fit index has limitations. Fit indexes may identify issues with a model, but they should not be seen as evidence of an optimal model. SEM is what Kline has called a *"disconfirmatory procedure that can help us to reject false models"*; we can only conclude that the model is consistent with the data [119, p. 21]. From this, it follows that our model is simply *a* model, and there may be many equivalent or better models that we have not identified.

6.3 Conclusion and Future Research

Crowdsourcing is an emerging alternative strategy to outsourcing software development, which has attracted considerable attention in recent years. Most studies thus far have focused on analyses of the "crowd" (i.e., developers) and crowdsourcing platforms (e.g., [35], [37]), but very little attention has been paid to the customer's perspective. To address this gap, this article presents a multi-method study on competition-based crowdsourcing software development from a customer perspective.

The first phase of our study comprised an exploratory industry case study with one company (TPI) that used the Topcoder platform for a non-trivial software development project. In this case study, TPI faced a number of significant challenges with the crowdsourcing process. First, the company had to deal with several layers of communication. The company felt the process of answering the many questions about their competitions quite cumbersome, and the high level of involvement of senior staff made this quite costly.

Another key issue related to planning and scheduling; the company found it quite challenging to deal with internal reviews in a timely manner. Furthermore, in order not to deter developers from future participation, the company tended to accept submissions even if their quality was less than acceptable. Although TPI used agile methods internally, from the company's perspective the crowdsourcing process reflected a waterfall process because there was a very strong focus on establishing and documenting requirements at the front-end of the process, while leaving QA activities until much later in the process. Finally, while Topcoder reports a very large number of registered developers, the active participation in TPI's set of competitions was quite limited, with only 37 unique developers participating. Thus, TPI's experience was that the potential of engaging a "crowd" was not achieved.

In Phase II of our study, we developed and evaluated a theoretical model of competition-based crowdsourcing which consists of a set of five hypotheses that link a number of factors to crowd developers' interest (i.e., registration for a competition that signals an intention to participate in that competition) and participation in crowdsourcing competitions. We evaluated these hypotheses using structural equation modeling with a large sample of competitions that we retrieved from Topcoder's public API. In this study, we found that the level of parallelism within a given project (representing a customer) has a significantly negative correlation with the level of interest in competitions. Furthermore, we found a statistically significant correlation between the interest in a competition and actual participation. We also found statistical support for the concept of "crowd killers": these are top-performing developers in the Topcoder community who, when they have registered for a competition, tend to deter other potential developers (hence the term "crowd killer"). In our study we did not find a statistical support for the relationship between the reward offered and the level of interest for a given competition. Nor did we find support for the relationship between a competition's duration and the level of interest for a given competition. While these findings suggest a number of actionable implications (see Section 6.1) for other companies engaging with competition-based crowdsourcing, we cautiously remind the reader that more studies are needed to confirm these findings before we could make clear recommendations with confidence.

In this article we position competition-based crowdsourcing as a distinct alternative form of outsourcing to an unknown workforce. We believe crowdsourcing has great potential, although as we demonstrated in this article there are considerable challenges that crowdsourcing customers may need to overcome. We suggest a number of avenues for future research:

- With "fleeting relationships" characterizing the interaction between crowdsourcing customers and crowd developers, what and how can customers effectively crowdsource in a software development context?
- Given the widespread adoption of agile approaches to software development (in particular Scrum) that emphasize regular face-to-face communication, how can the crowdsourcing approach (which resembles a waterfall-style approach to software development with an emphasis on documented requirements) be effectively combined and coordinated?
- How effective is the competition-based approach to crowdsourcing compared to alternative approaches to crowdsourcing software development?
- What factors make competitions attractive to the crowd? A clear understanding of this can prevent organizations from advertising competitions that are not attractive and thus may fail due to a lack of submissions.
- How can the "long tail" of the crowd be mobilized to participate in crowdsourcing approaches (either competition-based or otherwise)? That is, while Topcoder boasts more than 1.2 million members, only a fraction of its members seem to be actively participating. How can the "democratization of participation" [52] in crowd-based software development be truly achieved?

Answering these questions will require further research from all three key perspectives in crowdsourcing systems, namely crowdsourcing platforms, the crowds, and crowdsourcing customers. We believe the answers will imply new ways to work with unknown workforces such as the crowd.

Acknowledgments

We thank Lutz Prechelt and the anonymous reviewers for very thorough feedback which has led to a better article. We thank TPI for granting us access to the case study data. This work was supported, in part, by Science Foundation Ireland grant 15/SIRG/3293 and 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero—the Irish Software Research Centre (www.lero.ie).

References

[1] A. Begel, J. Bosch, and M. A. Storey, "Social networking meets software development: Perspectives from GitHub, MSDN, Stack-Exchange, and TopCoder," *IEEE Softw.*, vol. 30, no. 1, pp. 52–66, Jan./Feb. 2013.

[2] D. Tamburri, P. Lago, and H. van Vliet, "Organizational social structures for software engineering," *ACM Comput. Surveys*, vol. 46, no. 1, 2013, Art. no. 3.

[3] M.-A. Storey, A. Zagalsky, F. Filho, L. Singer, and D. German, "How social and communication channels shape and challenge a participatory culture in software development," *IEEE Trans. Softw. Eng.*, vol. 43, no. 2, pp. 185–204, Feb. 2017.

[4] B. Boehm, "A view of 20th and 21st century software engineering," in *Proc. Int. Conf. Softw. Eng.*, 2006, pp. 12–29.

[5] J. D. Herbsleb and A. Mockus, "An empirical study of speed and communication in globally distributed software development," *IEEE Trans. Softw. Eng.*, vol. 29, no. 6, pp. 481–494, Jun. 2003.

[6] S. Greengard, "Following the crowd," *Commun. ACM*, vol. 54, no. 2, pp. 20–22, 2011.

[7] J. Howe, "The rise of crowdsourcing," *Wired*, vol. 14, pp. 1–4, 2006.

[8] D. C. Brabham, *Crowdsourcing*. Cambridge, MA, USA: MIT Press, 2013.

[9] E. Schenk and C. Guittard, "Towards a characterization of crowdsourcing practices," *J. Innovation Economics*, vol. 1, no. 7, pp. 93–107, 2011.

[10] L. Hoffmann, "Crowd control," *Commun. ACM*, vol. 52, no. 3, pp. 16–17, 2009.

[11] J. Howe, *Crowdsourcing: Why the Power of the Crowd Is Driving the Future of Business*. New York, NY, USA: Crown Business, 2008.

[12] K. R. Lakhani and J. A. Panetta, "The principles of distributed innovation," *Innovations: Technol. Governance Globalization*, vol. 2, no. 3, pp. 97–112, 2007.

[13] A. Doan, R. Ramakrishnan, and A. Y. Halevy, "Crowdsourcing systems on the world-wide web," *Commun. ACM*, vol. 54, no. 4, pp. 86–96, 2011.

[14] K. Mao, L. Capra, M. Harman, and Y. Jia, "A survey of the use of crowdsourcing in software engineering," *J. Syst. Softw.*, vol. 126, pp. 57–84, 2016.

[15] P. G. Ipeirotis, "Analyzing the Amazon mechanical turk marketplace," *Crossroads ACM Mag. Students*, vol. 17, no. 2, pp. 16–21, 2010.

[16] A. Kittur, "Crowdsourcing, collaboration and creativity," *Crossroads ACM Mag. Students*, vol. 17, no. 2, pp. 22–26, 2010.

[17] A. Kittur, B. Smus, S. Khamkar, and R. E. Kraut, "CrowdForge: Crowdsourcing complex work," in *Proc. ACM Symp. User Interface Softw. Technol.*, 2011, pp. 43–52.

[18] A. Kittur, et al., "The future of crowd work," in *Proc. ACM Conf. Comput.-Supported Cooperative Work*, 2013, pp. 1301–1318.

[19] N. Kaufmann, T. Schulze, and D. Veit, "More than fun and money. Worker motivation in crowdsourcing—A study on mechanical turk," in *Proc. 17th Americas Conf. Inf. Syst.*, 2011, pp. 1–11.

[20] E. Schenk and C. Guittard, "Crowdsourcing: What can be outsourced to the crowd, and why?" Available: http://halshs.archives-ouvertes.fr/halshs-00439256/, 2009.

[21] K. R. Lakhani, D. A. Garvin, and E. Lonstein, "TopCoder (A): Developing software through crowdsourcing," *Harvard Business School 610–032*, 2010.

[22] N. Savage, "Gaining wisdom from crowds," *Commun. ACM*, vol. 55, no. 3, pp. 13–15, 2012.

[23] R. L. Saremi, Y. Yang, G. Ruhe, and D. Messinger, "Leveraging crowdsourcing for team elasticity: An empirical evaluation at TopCoder," in *Proc. 39th Int. Conf. Softw. Eng.*, 2017, pp. 103–112.

[24] E. Bonabeau, "Decisions 2.0: The power of collective intelligence," *MIT Sloan Manage. Rev.*, vol. 50, no. 2, pp. 45–52, 2009.

[25] C. Bonner, *10 Burning Questions on Crowdsourcing: Your Starting Guide to Open Innovation and Crowdsourcing Success*, I. Heffan, Ed. Indianapolis, IN, USA: TopCoder Inc., 2013.

[26] K. J. Boudreau, N. Lacetera, and K. R. Lakhani, "Incentives and problem uncertainty in innovation contests: An empirical analysis," *Manage. Sci.*, vol. 57, no. 5, pp. 843–863, 2011.

[27] W. Ebner, M. Leimeister, U. Bretschneider, and H. Krcmar, "Leveraging the wisdom of crowds: Designing an IT-supported ideas competition for an ERP software company," in *Proc. 41st Hawaii Int. Conf. Syst. Sci.*, 2008, pp. 417–417.

[28] L. B. Erickson, "Leveraging the crowd as a source of innovation: Does crowdsourcing represent a new model for product and service innovation?" in *Proc. SIGMIS Comput. People Res.*, 2012, pp. 91–96.

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

[30] B. Fitzgerald, "Software crisis 2.0," *IEEE Comput.*, vol. 45, no. 4, pp. 89–91, Apr. 2012.

[31] A. Begel, J. Herbsleb, and M.-A. Storey, "The future of collaborative software development," in *Proc. ACM Conf. Comput. Supported Cooperative Work Companion*, 2012, pp. 17–18.

[32] R. Kazman and H.-M. Chen, "The metropolis model: A new logic for development of crowdsourced systems," *Commun. ACM*, vol. 52, no. 7, pp. 76–84, 2009.

[33] K. Stol, T. LaToza, and C. Bird, "Crowdsourcing for software engineering," *IEEE Softw.*, vol. 34, no. 2, pp. 30–36, Mar./Apr. 2017.

[34] N. Archak, "Money, glory and cheap talk: Analyzing strategic behavior of contestants in simultaneous crowdsourcing contests on TopCoder.com," in *Proc. 19th Int. Conf. World Wide Web*, 2010, pp. 21–30.

[35] Y. Yang and R. Saremi, "Award vs. worker behaviors in competitive crowdsourcing tasks," in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2015, pp. 1–10.

[36] Y. Yang, M. Karim, R. Saremi, and G. Ruhe, "Who should take this task?—Dynamic decision support for crowd workers," in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2016, Art. no. 8.

[37] H. Zhang, Y. Wu, and W. Wu, "Analyzing developer behavior and community structure in software crowdsourcing," in *Information Science and Applications*, K. Kim, Ed. Berlin, Germany: Springer, 2015.

[38] K. Stol and B. Fitzgerald, "Two's company, three's a crowd: A case study of crowdsourcing software development," in *Proc. 36th Int. Conf. Softw. Eng.*, 2014, pp. 187–198.

[39] K. Eisenhardt and M. Graebner, "Theory building from cases: Opportunities and challenges," *Academy Manage. J.*, vol. 50, no. 4, pp. 25–32, 2007.

[40] J. Feller, P. Finnegan, B. Fitzgerald, and J. Hayes, "From peer production to productization: A study of socially enabled business exchanges in open source service networks," *Inf. Syst. Res.*, vol. 19, no. 4, pp. 475–493, 2008.

[41] P. Ågerfalk and B. Fitzgerald, "Outsourcing to an unknown workforce: Exploring opensourcing as a global sourcing strategy," *MIS Quart.*, vol. 32, no. 2, pp. 385–409, 2008.

[42] R. Musson, J. Richards, D. Fisher, C. Bird, B. Bussone, and S. Ganguly, "Leveraging the crowd: How 48,000 users helped improve Lync performance," *IEEE Softw.*, vol. 30, no. 4, pp. 38–45, Jul./Aug. 2013.

[43] Y.-H. Tung and S.-S. Tsenga, "A novel aproach to collaborative testing in a crowdsourcing environment," *J. Syst. Softw.*, vol. 86, no. 8, pp. 2143–2153, 2013.

[44] A. L. Zanatta, L. S. Machado, G. B. Pereira, R. Prikladnicki, and E. Carmel, "Software crowdsourcing platforms," *IEEE Softw.*, vol. 33, no. 6, pp. 112–116, Nov./Dec. 2016.

[45] Topcoder, "Topcoder website." [Online]. Available: http://www.topcoder.com

[46] P. Ågerfalk, B. Fitzgerald, and K. Stol, *Software Sourcing in the Age of Open: Leveraging the Unknown Workforce*. Berlin, Germany: Springer, 2015.

[47] G. von Krogh, S. Haefliger, S. Spaeth, and M. W. Wallin, "Carrots and rainbows: Motivation and social practice in open source software development," *MIS Quart.*, vol. 36, no. 2, pp. 649–676, 2012.

[48] E. Estellés-Arolas and F. González-Ladrón-de-Guevara, "Towards an integrated crowdsourcing definition," *J. Inf. Sci.*, vol. 38, no. 2, pp. 189–200, 2012.

[49] L. Hetmank, "Components and functions of crowdsourcing systemsa systematic literature review," in *Proc. 11th Int. Conf. Wirtschaftsinformatik*, 2013, pp. 55–69.

[50] J. Howe, [Online]. Available: http://www.crowdsourcing.com

[51] D. Naparat and P. Finnegan, "Crowdsourcing software requirements and development: A mechanism-based exploration of 'opensourcing'," in *Proc. 19th Americas Conf. Inf. Syst.*, 2013.

[52] T. D. LaToza and A. van der Hoek, "Crowdsourcing in software engineering: Models, motivations, and challenges," *IEEE Softw.*, vol. 33, no. 1, pp. 74–80, Jan./Feb. 2016.

[53] J. Feller and B. Fitzgerald, *Understanding Open Source Software Development*. London, U.K.: Pearson Education, 2002.

[54] B. Fitzgerald, "The transformation of open source software," *MIS Quart.*, vol. 30, no. 3, pp. 587–598, 2006.

[55] M. Germonprez, J. Kendall, K. Kendall, L. Mathiassen, B. Young, and B. Warner, "A theory of responsive design: A field study of corporate engagement with open source communities," *Inf. Syst. Res.*, vol. 28, pp. 64–83, 2016.

[56] L. B. Erickson, I. Petrick, and E. M. Trauth, "Organizational uses of the crowd: Developing a framework for the study of crowdsourcing," in *Proc. Annu. Conf. Comput. People Res.*, 2012, pp. 155–158.

[57] H. Simula, "The rise and fall of crowdsourcing?" in *Proc. 46th Hawaii Int. Conf. Syst. Sci.*, 2013, pp. 2783–2791.

[58] A. Schwarz, M. Mehta, N. Johnson, and W. Chin, "Understanding frameworks and reviews: A commentary to assist us in moving our field forward by analyzing our past," *Database Adv. Inf. Syst.*, vol. 38, no. 3, pp. 29–50, 2007.

[59] P. G. Ipeirotis and P. K. Paritosh, "Managing crowdsourced human computation," in *Proc. 20th Int. Conf. World Wide Web*, 2011, pp. 287–288.

[60] A. Kulkarni, M. Can, and B. Hartmann, "Collaboratively crowdsourcing workflows with Turkomatic," in *Proc. ACM Comput.-Supported Cooperative Work*, 2012, pp. 1003–1012.

[61] J. D. Herbsleb and R. E. Grinter, "Splitting the organization and integrating the code: Conway's Law revisited," in *Proc. 21st Int. Conf. Softw. Eng.*, 1999, pp. 85–95.

[62] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Commun. ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

[63] T. D. LaToza, W. B. Towne, A. van der Hoek, and J. D. Herbsleb, "Crowd development," in *Proc. 6th Int. Workshop Cooperative Human Aspects Softw. Eng.*, 2013, pp. 85–88.

[64] T. D. LaToza, W. Towne, C. Adriano, and A. van der Hoek, "Microtask programming: Building software with a crowd," in *Proc. 27th ACM Symp. User Interface Softw. Technol.*, 2014, pp. 43–54.

[65] H. Tajedin and D. Nevo, "Determinants of success in crowdsourcing software development," in *Proc. SIGMIS Comput. People Res.*, 2013, pp. 173–178.

[66] T. W. Malone and K. Crowston, "The interdisciplinary study of coordination," *ACM Comput. Surveys*, vol. 26, no. 1, pp. 87–119, 1994.

[67] R. E. Kraut and L. A. Streeter, "Coordination in software development," *Commun. ACM*, vol. 38, no. 3, pp. 69–81, 1995.

[68] M. Cataldo and J. Herbsleb, "Coordination breakdowns and their impact on development productivity and software failures," *IEEE Trans. Softw. Eng.*, vol. 39, no. 3, pp. 343–360, Mar. 2013.

[69] K. Mao, Y. Yang, Q. Wang, Y. Jia, and M. Harman, "Developer recommendation for crowdsourced software development tasks," in *Proc. IEEE Symp. Service-Oriented Syst. Eng.*, 2015, pp. 347–356.

[70] F. P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*. Reading, MA, USA: Addison-Wesley, 1995.

[71] Y. Zhao and Q. Zhu, "Evaluation on crowdsourcing research: Current status and future direction," *Inf. Syst. Frontiers*, vol. 16, no. 3, pp. 417–434, Jul. 2014.

[72] D. C. Brabham, "Crowdsourcing as a model for problem solving: An introduction and cases," *Convergence*, vol. 14, no. 1, pp. 75–90, 2008.

[73] M. Vukovic, "Crowdsourcing for enterprises," in *Proc. Congr. Services I*, 2009, pp. 686–692.

[74] S. P. Dow, A. Kulkarni, S. R. Klemmer, and B. Hartmann, "Shepherding the crowd yields better work," in *Proc. ACM Conf. Comput. Supported Cooperative Work*, 2012, pp. 1013–1022.

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

[76] P. Kinnaird, L. Dabbish, S. Kiesler, and H. Faste, "Co-worker transparency in a microtask marketplace," in *Proc. ACM Comput. Supported Coordination Work*, 2013, pp. 1285–1290.

[77] D. C. Brabham, "The myth of amateur crowds: A critical discourse analysis of crowdsourcing coverage," *Inf., Commun. Soc.*, vol. 15, no. 3, pp. 394–410, 2012.

[78] T. LaToza, M. Chen, L. Jiang, M. Zhao, and A. van der Hoek, "Borrowing from the crowd: A study of recombination in software design competitions," in *Proc. 37th Int. Conf. Softw. Eng.*, 2015, pp. 551–562.

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

[80] F. Bjørnson and T. Dingsøyr, "Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used," *Inf. Softw. Technol.*, vol. 50, no. 11, pp. 1055–1068, 2008.

[81] K. C. Desouza and J. R. Evaristo, "Managing knowledge in distributed projects," *Commun. ACM*, vol. 47, no. 4, pp. 87–91, 2004.

[82] L. Dabbish, R. Farzan, R. Kraut, and T. Postmes, "Fresh faces in the crowd: Turnover, identity, and commitment in online groups," in *Proc. ACM Conf. Comput. Supported Cooperative Work*, 2012, pp. 245–248.

[83] T. K. Abdel-Hamid, "A study of staff turnover, acquisition, and assimilation and their impact on software development cost and schedule," *J. Manage. Inf. Syst.*, vol. 6, no. 1, pp. 21–40, 1989.

[84] S. Wolfson and M. Lease, "Look before you leap: Legal pitfalls of crowdsourcing," in *Proc. ASIST Annu. Meet.*, 2011, pp. 1–10.

[85] V. Chanal and M. L. Caron-Fasan, "The difficulties involved in developing business models open to innovation communities: The case of a crowdsourcing platform," *M@n@gement*, vol. 13, no. 4, pp. 318–341, 2010.

[86] G. Jouret, "Inside Cisco's search for the next big idea," *Harvard Bus. Rev.*, vol. 87, no. 9, pp. 43–45, 2009.

[87] D. Chandler and A. Kapelner, "Breaking monotony with meaning: Motivation in crowdsourcing markets," *J. Econ. Behavior Org.*, vol. 90, pp. 123–133, 2013.

[88] Y. C. Zhao and Q. Zhu, "Effects of extrinsic and intrinsic motivation on participation in crowdsourcing contest: A perspective of self-determination theory," *Online Inf. Rev.*, vol. 38, no. 7, pp. 896–917, 2014.

[89] D. DiPalantino and M. Vojnovic, "Crowdsourcing and all-pay auctions," in *Proc. 10th ACM Conf. Electron. Commerce*, 2009, pp. 119–128.

[90] S. Faridani, B. Hartmann, and P. G. Ipeirotis, "What's the right price? pricing tasks for finishing on time," in *Proc. AAAI Workshop Human Comput.*, 2011, pp. 26–31.

[91] J. J. Horton and L. B. Chilton, "The labor economics of paid crowdsourcing," in *Proc. ACM 11th Conf. Electron. Commerce*, 2010, pp. 209–218.

[92] K. Mao, Y. Yang, M. Li, and M. Harman, "Pricing crowdsourcing-based software development tasks," in *Proc. 35th Int. Conf. Softw. Eng.*, 2013, pp. 1205–1208.

[93] W. Mason and D. J. Watts, "Financial incentives and the 'performance of crowds'," in *Proc. ACM SIGKDD Workshop Human Comput.*, 2009, pp. 77–85.

[94] D. Pilz and H. Gewald, "Does money matter? Motivational factors for participation in paid- and non-profit-crowdsourcing communities," in *Proc. 11th Int. Conf. Wirtschaftsinformatik*, 2013, Art. no. 37.

[95] S. Beecham, N. Baddoo, T. Hall, H. Robinson, and H. Sharp, "Motivation in software engineering: A systematic literature review," *Inf. Softw. Technol.*, vol. 50, no. 9/10, pp. 860–878, 2008.

[96] B. W. Boehm, *Software Engineering Economics*. London, U.K.: Pearson Education, 1981.

[97] N. Baddoo and T. Hall, "Motivators of software process improvement: An analysis of practitioners' views," *J. Syst. Softw.*, vol. 62, no. 2, pp. 85–96, 2002.

[98] Y. Singer and M. Mittal, "Pricing mechanisms for crowdsourcing markets," in *Proc 22nd Int. Conf. World Wide Web*, 2013, pp. 1157–1166.

[99] G. G. Gable, "Integrating case study and survey research methods: An example in information systems," *Eur. J. Inf. Syst.*, vol. 3, no. 2, pp. 112–126, 1994.

[100] P. Ågerfalk, "Embracing diversity through mixed methods research," *Eur. J. Inf. Syst.*, vol. 22, no. 3, pp. 251–256, 2013.

[101] J. Mingers, "Combining is research methods: Towards a pluralist methodology," *Inf. Syst. Res.*, vol. 12, no. 3, pp. 240–259, 2001.

[102] P. Clarke, R. O'Connor, B. Leavy, and M. Yilmaz, "Exploring the relationship between software process adaptive capability and organisational performance," *IEEE Trans. Softw. Eng.*, vol. 41, no. 12, pp. 1169–1183, Dec. 2015.

[103] R. Yin, *Case Study Research*, 3rd ed. Thousand Oaks, CA, USA: Sage, 2003.

[104] B. Kaplan and D. Duchon, "Combining qualitative and quantitative methods in information systems research: A case study," *MIS Quart.*, vol. 12, no. 4, pp. 571–586, 1988.

[105] J. Danziger and K. Kraemer, *Survey Research and Multiple Operationism: The URBIS Project Methodology*. Boston, MA, USA: Harvard Bus. School Press, 1991, pp. 351–371.

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

[107] A. Mockus, R. Fielding, and J. D. Herbsleb, "A case study of open source software development: The Apache server," in *Proc. Int. Conf. Softw. Eng.*, 2000, pp. 263–272.

[108] Topcoder, "The 3 pillars of digital creation at Topcoder - enterprise open innovation." [Online]. Available: https://www.youtube.com/watch?v=4QVVQdaXnYo, Accessed on: May. 25, 2017.

[109] Topcoder, "How it works: Community driven design, development, & data science." [Online]. Available: https://www.topcoder.com/community/how-it-works/

[110] K. Stol and B. Fitzgerald, "Research protocol for a case study of crowdsourcing software development," Univ. Limerick, Tech. Rep., Limerick, Ireland.

[111] M. Leininger, *Criteria and Critique*. Thousand Oaks, CA, USA: Sage Publications, 1994.

[112] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Trans. Softw. Eng.*, vol. 25, no. 4, pp. 557–572, Jul./Aug. 1999.

[113] M. Miles and A. Huberman, *Qualitative Data Analysis: An Expanded Sourcebook*, 2nd ed. Thousand Oaks, CA, USA: Sage Publications, 1994.

[114] K. Stol, P. Ralph, and B. Fitzgerald, "Grounded theory in software engineering research: A critical review and guidelines," in *Proc. 38th Int. Conf. Softw. Eng.*, 2016, pp. 120–131.

[115] R. Hoyle, Ed., *Handbook of Structural Equation Modeling*. New York, NY, USA: Guildford Press, 2012.

[116] E. Capra, C. Francalanci, and F. Merlo, "An empirical study on the relationship among software design quality, development effort, and governance in open source projects," *IEEE Trans. Softw. Eng.*, vol. 34, no. 6, pp. 765–782, Nov./Dec. 2008.

[117] Y. Lindsjørn, D. I. Sjøberg, T. Dingsøyr, G. Bergersen, and T. Dybå, "Teamwork quality and project success in software development: A survey of agile development teams," *J. Syst. Softw.*, vol. 122, pp. 274–286, 2016.

[118] R. E. Schumacker and R. Lomax, *A Beginner's Guide to Structural Equation Modeling*, 4th ed. Evanston, IL, USA: Routledge, 2016.

[119] R. Kline, *Principles and Practice of Structural Equation Modeling*, 4th ed. New York, NY, USA: Guilford Press, 2016.

[120] K. Stol and B. Fitzgerald, "Theory-oriented software engineering," *Sci. Comput. Program.*, vol. 101, pp. 79–98, 2015.

[121] Y. Rosseel, "lavaan: An R package for structural equation modeling," *J. Statistical Softw.*, vol. 48, no. 2, pp. 1–36, 2012.

[122] A. Dubey, et al., "Dynamics of software development crowdsourcing," in *Proc. IEEE 11th Int. Conf. Global Softw. Eng.*, 2016, pp. 49–58.

[123] M.-H. R. Ho, S. Stark, and O. Chernyshenko, "Graphical representation of structural equation models using path diagrams," in *Handbook of Structural Equation Modeling*, R. H. Hoyle, Ed. New York, NY, USA: Guildford Press, 2012.

[124] R. Saremi and Y. Yang, "Empirical analysis on parallel tasks in crowdsourcing software development," in *Proc. 30th IEEE/ACM Int. Conf. Automated Softw. Eng. Workshop*, 2015, pp. 28–34.

[125] J. Roberts, I.-H. Hann, and S. Slaughter, "Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the Apache projects," *Manage. Sci.*, vol. 52, no. 7, pp. 984–999, 2006.

[126] N. Archak and A. Sundararajan, "Optimal design of crowdsourcing contests," in *Proc. 30th Int. Conf. Inf. Syst.*, 2009, pp. 1–16.

[127] J. Leimeister, M. Huber, U. Bretschneider, and H. Krcmar, "Leveraging crowdsourcing: Activation-supporting components for it-based ideas competition," *J. Manage. Inf. Syst.*, vol. 26, no. 1, pp. 197–224, 2009.

[128] B. Boehm, C. Abst, and S. Chulani, "Software development cost estimation approaches—A survey," *Ann. Softw. Eng.*, vol. 10, no. 1–4, pp. 177–205, 2005.

[129] T. Walter and A. Back, "Towards measuring crowdsourcing success: An empirical study on effects of external factors in online idea contest," in *Proc. Mediterranean Conf. Inf. Syst.*, 2011.

[130] Y. Yang, P.-Y. Chen, and P. Pavlou, "Open innovation: An empirical study of online contests," in *Proc. Int. Conf. Inf. Syst.*, 2009.

[131] D. Gefen, G. Gefen, and E. Carmel, "How project description length and expected duration affect bidding and project success in crowdsourcing software development," *J. Syst. Softw.*, vol. 116, pp. 75–84, 2016.

[132] R. Sorrentino and E. Higgins, *Handbook of Motivation and Cognition: Foundations of Social Behaviour*, vol. 1. Hoboken, NJ, USA: Wiley, 1986.

[133] I. Ajzen and M. Fishbein, "Factors influencing intentions and the intention-behavior relation," *Human Relations*, vol. 27, no. 1, p. 1–15, 1974.

[134] I. Ajzen and M. Fishbein, *Understanding Attitudes and Predicting Social Behavior*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1980.

[135] W. Lei, M. N. Huhns, W.-T. Tsai, and W. Wu, eds., *Crowdsourcing: Cloud-Based Software Development*. Berlin, Germany: Springer, 2015.

[136] S. Finney and C. DiStefano, *Non-Normal and Categorical Data in Structural Equation Modeling*. Charlotte, NC, USA: Information Age Publishing, 2006.

[137] R. E. Schumacker and R. G. Lomax, *A Beginner's Guide to Structural Equation Modeling*, 2nd ed. Mahwah, NJ, USA: Lawrence Erlbaum Associates, 2004.

[138] B. Kitchenham, et al., "Robust statistical methods for empirical software engineering," *Empirical Softw. Eng.*, vol. 22, no. 2, pp. 579–630, 2017.

[139] O. V. Berkout, A. M. Gross, and J. Young, "Why so many arrows? Introduction to structural equation modeling for the novitiate user," *Clinical Child Family Psychology Rev.*, vol. 17, pp. 217–229, 2014.

[140] J. J. Hox, C. J. Maas, and M. J. Brinkhuis, "The effect of estimation method and sample size in multilevel structural equation modeling," *Statistica Neerlandica*, vol. 64, no. 2, pp. 157–170, 2010.

[141] A. Satorra and P. Bentler, "Corrections to test statistics and standard errors in covariance structure analysis," in *Latent Variables Analysis: Applications for Developmental Research*, A. von Eye and C. Clogg, Eds. Thousand Oaks, CA, USA: Sage, 1994, pp. 399–419.

[142] K. Bollen and R. Stine, "Bootstrapping goodness-of-fit measures in structural equation models," in *Testing Structural Equation Models*, K. Bollen and J. Long, Eds. Thousand Oaks, CA, USA: Sage Publications, 1993, pp. 111–135.

[143] R. P. Bagozzi and Y. Yi, "Specification, evaluation, and interpretation of structural equation models," *J. Academy Marketing Sci.*, vol. 40, no. 1, pp. 8–34, 2012.

[144] B. Wheaton, B. Muthen, D. Alwin, and G. Summers, "Assessment reliability and stability in panel models," in *Sociological Methodology*, D. Heise, Ed. San Francisco, CA, USA: Jossey-Bass, 1977, pp. 84–136.

[145] R. L. Matsueda, *Key Advances in the History of Structural Equation Modeling*. New York, NY, USA: Guilford Press, 2012.

[146] J. H. Steiger and J. C. Lind, "Statistically-based tests for the number of common factors," in *Proc. Handout Talk Annu. Meet. Psychometric Soc.*, 1980.

[147] L. Hu and P. M. Bentler, "Cutoff criteria for fit indexes in covariance structure analysis: Conventional criteria versus new alternatives," *Structural Equation Model.: A Multidisciplinary J.*, vol. 6, no. 1, pp. 1–55, 1999.

[148] S. L. Hoe, "Issues and procedures in adopting structural equation modeling technique," *J. Appl. Quantitative Methods*, vol. 3, no. 1, pp. 76–83, 2008.

[149] B. Wheaton, "Assessment of fit in overidentified models with latent variables," *Sociol. Methods Res.*, vol. 16, no. 1, pp. 118–154, 1987.

[150] P. Bentler, "Comparative fit indexes in structural models," *Psychological Bulletin*, vol. 107, no. 2, pp. 238–246, 1990.

[151] R. L. Matsueda, "Model fit and model selection in structural equation modeling," in *Handbook of Structural Equation Modeling*, R. H. Hoyle, Ed. New York, NY, USA: Guilford Press, 2012.

[152] R. McDonald and R. Moon-Ho, "Principles and practice in reporting structural equation analyses," *Psychological Methods*, vol. 7, no. 1, pp. 64–82, 2002.

[153] D. A. Kenny, *Correlation and Causality*, revised ed., 2004.

[154] Topcoder, "Interview with first TC millionaire argolite." [Online]. Available: https://community.topcoder.com/tco11/2011/07/25/interview-with-first-tc-millionaire-argolite/, Accessed on: May. 25, 2017.

[155] B. Fitzgerald and D. Howcroft, "Towards dissolution of the is research debate: From polarization to polarity," *J. Inf. Technol.*, vol. 13, pp. 313–326, 1998.

[156] J. W. Creswell and D. L. Miller, "Determining validity in qualitative inquiry," *Theory Practice*, vol. 39, no. 3, pp. 124–130, 2000.

[157] G. Walsham, "Interpretive case studies in IS research: Nature and method," *Eur. J. Inf. Syst.*, vol. 4, pp. 74–81, 1995.

Klaas-Jan Stol is a lecturer in the Department of Computer Science, University College Cork, and a Science Foundation Ireland principal investigator. He is a member of Lero—the Irish Software Research Centre, where he was a research fellow prior to his position with UCC. His research interests include research methodology, and contemporary software development approaches, specifically open source software, inner source, and crowdsourcing.

Bora Caglayan is a researcher with IBM Ireland. Previously, he was a post-doctoral researcher with Lero—the Irish Software Research Centre, University of Limerick. His research interests include empirical software engineering and recommender systems in software engineering.

Brian Fitzgerald is director of Lero—the Irish Software Research Centre. He holds an endowed chair, the Frederick Krehbiel II Chair in Innovation in Business and Technology, University of Limerick. His research interests include open source software, inner source, crowdsourcing, and lean and agile methods.