Research Note
RN/15/01

A Survey of the Use of Crowdsourcing in Software Engineering

May 10, 2015

Ke Mao, Licia Capra, Mark Harman and Yue Jia

University College London
{*k.mao, l.capra, m.harman, y.jia*}*@cs.ucl.ac.uk*

Abstract

Crowdsourcing can be used to support software engineering activities and research into these activities. In this paper we provide a comprehensive survey of the use of crowdsourcing to support software engineering activities (Crowdsourced Software Engineering), seeking to cover all literature on this topic. We describe the software engineering domains, tasks and applications for crowdsourcing and the platforms and stakeholders involved in realising Crowdsourced Software Engineering solutions. We also expose trends, issues and opportunities for Crowdsourced Software Engineering.

1 INTRODUCTION

Crowdsourcing is an emerging distributed problem-solving model based on the combination of human and machine computation. The term 'crowdsourcing' was jointly[1] coined by Howe and Robinson in 2006 [2]. According to the widely accepted definition presented in this article, crowdsourcing is the act of an organisation outsourcing their work to an undefined, networked labour using an open call for participation.

Crowdsourced Software Engineering derives from crowdsourcing more generally. It utilises an open call format to recruit global online software engineers, to work on various types of software engineering tasks, such as requirements extraction, design, coding and testing. This emerging development model has been claimed to reduce time-to-market by increasing parallelism [3]–[5], and to lower costs and defect rates with flexible development capability [3]. Crowdsourced Software Engineering is implemented by many successful crowdsourcing platforms, such as TopCoder, AppStori, uTest, Mob4Hire and TestFlight.

Crowdsourcing model has been applied to a wide range of creative and design-based activities [6]–[10]. Crowdsourced Software Engineering has also rapidly gained increasing interest in both industrial and academic communities. Our survey reveals a dramatic rise in recent work on Crowdsourced Software Engineering, yet many authors write that there is 'little work' on crowdsourcing for/in software engineering [5], [11]–[15]. These authors can easily be forgiven for this misconception, since the field is growing rapidly and touches many disparate aspects of software engineering, resulting in a literature that is spread over many different software engineering application areas and domains. This motivates the need for a comprehensive survey such as the one we present here.

The purpose of our survey was two-fold: First, to provide a comprehensive survey of current research progress in the field of Crowdsourced Software Engineering. Second, to summarise the challenges for Crowdsourced Software Engineering and to reveal to what extent these challenges were addressed by existing work.

The aim of conducting a comprehensive survey on all publications related to Crowdsourced Software Engineering necessitates a careful and thorough paper selection process. The process we adopted was borrowed from the conventions used for systematic literature reviews [16], [17]. Specifically, we followed the three steps described below:

To start with, we defined the inclusion criteria of the surveyed papers: the main criterion for including a paper in our survey is that the paper describes research on crowdsourcing that can serve for any activities involved in software engineering process. Also, the papers included in this survey must be presented in English as either conference papers, journal papers, theses, technical reports or books.

We performed three types of searches on related papers published before April 2015:

- Online library search using seven major search engines: *ACM Digital Library*, *IEEE Xplore Digital Library*, *Springer Link Online Library*, *Wiley Online Library*, *Elsevier ScienceDirect*, *ProQuest Research Library* and *Google Scholar*. A list of search queries we used are presented in Appendix A. We searched these queries in the fields of title, abstract and index terms of the papers.
- Issue-by-issue search of main conference proceedings and journals in software engineering from January 2006 to March 2015. This process was conducted manually to find those relevant papers that cannot be retrieved by the previous step. The specific conference proceedings and journals searched are listed in Table 1.
- Reference search for identifying missed papers by going through citations from included papers (snowballing).

Lastly, we conducted a screening process[2] to filter the collected papers by removing any that were not in the scope of Crowdsourced Software Engineering. We read the papers carefully and thoroughly, applying the inclusion criteria again and filtering out unrelated papers manually. After the screening process, 203 papers remained, and were selected for inclusion in this survey. The growth trend in publications is presented in Fig. 1. The distribution of these papers' publication types and a specific list of Master/PhD theses can be found in Fig. 2 and Table 2, respectively. As can be seen, there is a noticeable rise in publications on Crowdsourced Software Engineering, resulting in a significant body of literature which we study in depth in this survey.

We have built a repository which contains the meta-data of our collected papers. The meta-data includes the author, title, publication year and type as well as the conference proceeding/journal information of the paper. It was based on this repository that we conducted our analysis of the reviewed papers. This repository is publicly available online[3].

We also have employed the crowdsourcing model to perform a 'crowdsourced checking' process for this survey itself. We made an open call to the authors via email, for checking if there are any errors or missing

---

1. Jeff Howe attributes the creation of the term to Mark Robinson and himself [1].
2. The screening process is iterative, e.g., we also screened the papers suggested by authors in a later 'crowdsourced checking' process.
3. http://www.cs.ucl.ac.uk/staff/k.mao/cserep

TABLE 1
Selected Conference Proceedings and Journals for Manual Search

| Abbr.	Source
ICSE	International Conference on Software Engineering
ESEC/FSE	European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering
OOPSLA	Conference on Object-Oriented Programming Systems, Languages, and Applications
ASE	International Conference on Automated Software Engineering
ISSTA	International Symposium on Software Testing and Analysis
ICST	International Conference on Software Testing, Verification and Validation
RE	International Requirements Engineering Conference
CSCW	Conference on Computer-Supported Cooperative Work and Social Computing
ISSC	International Symposium on Software Crowdsourcing
CSI-SE	International Workshop on Crowdsourcing in Software Engineering
TSE	Transactions on Software Engineering
TOSEM	Transactions on Software Engineering Methodology
IEEE SW	IEEE Software
IET	IET Software
IST	Information and Software Technology
JSS	Journal of Systems and Software
SQJ	Software Quality Journal
SPE	Software: Practice and Experience
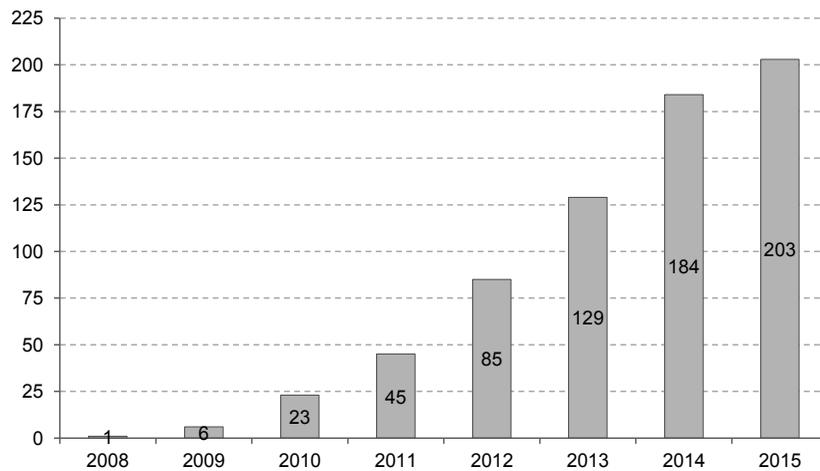
Fig. 1. Cumulative growth of Crowdsourced Software Engineering studies published before April 2015.
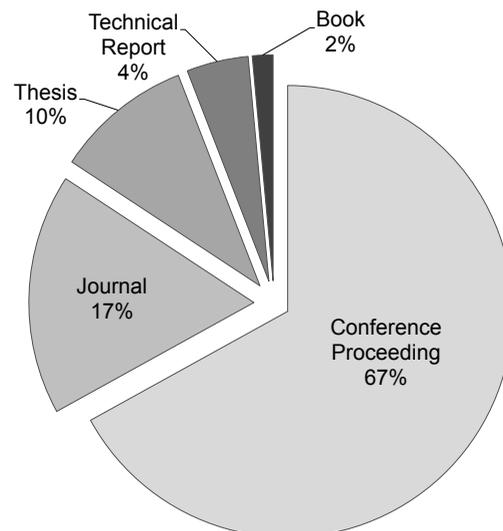

Fig. 2. Publication type of surveyed papers.

TABLE 2
A List of Master and PhD Theses on Crowdsourced Software Engineering

Year	Author	Degree	University	Title
2010	Lim [18]	PhD	University of New South Wales	Social Networks and Collaborative Filtering for Large-Scale Requirements Elicitation
2011	Manzoor [19]	Master	KTH - Royal Institute of Technology	A Crowdsourcing Framework for Software Localization
2011	Kallenbach [20]	Master	RWTH Aachen University	HelpMeOut - Crowdsourcing Suggestions to Programming Problems for Dynamic, Interpreted Languages
2011	Leone [21]	PhD	ETH Zurich - Swiss Federal Institute of Technology	Information Components as a Basis for Crowdsourced Information System Development
2012	Nag [22]	Master	Massachusetts Institute of Technology	Collabourative Competition for Crowdsourcing Spaceflight Software and STEM Education Using SPHERES Zero Robotics
2012	Saengkhattiya et al. [23]	Master	Lund University	Quality in Crowdsourcing - How Software Quality is Ensured in Software Crowdsourcing
2012	Gritti [24]	Master	Universitat Politécnica de Catalunya	Crowd Outsourcing for Software Localization
2012	Ponzanelli [25]	Master	University of Lugano	Exploiting Crowd Knowledge in the IDE
2012	Phair [26]	PhD	Colorado Technical University	Open Crowdsourcing: Leveraging Community Software Developers for IT Projects
2012	Bruch [27]	PhD	Technische Universität Darmstadt	IDE 2.0: Leveraging the Wisdom of the Software Engineering Crowds
2012	Goldman [28]	PhD	Massachusetts Institute of Technology	Software Development with Real-time Collabourative Editing
2013	Mijnhardt [29]	Master	Utrecht University	Crowdsourcing for Enterprise Software Localization
2013	Teinum [30]	Master	University of Agder	User Testing Tool: Towards a Tool for Crowdsource-Enabled Accessibility Evaluation of Websites
2013	Starov [31]	Master	East Carolina University	Cloud Platform for Research Crowdsourcing in Mobile Testing
2013	Chilana [32]	PhD	University of Washington	Supporting Users After Software Deployment through Selection-Based Crowdsourced Contextual Help
2013	Wightman [33]	PhD	Queen's University	Search Interfaces for Integrating Crowdsourced Code Snippets within Development Environments
2013	Xue [34]	PhD	University of Illinois at Urbana-Champaign	Using Redundancy to Improve Security and Testing
2013	Lin [35]	PhD	Carnegie Mellon University	Understanding and Capturing Peoples Mobile App Privacy Preferences
2014	Schiller [12]	PhD	University of Washington	Reducing the Usability Barrier to Specification and Verification
2015	Snijders [36]	Master	Utrecht University	Crowd-Centric Requirements Engineering: A Method based on Crowdsourcing and Gamification

information regarding our description of their work. For each cited paper, we distributed the copy of this survey to at least one author. In total we have contacted 303 authors and have received 83 replies. We then further revised this survey according to the authors' reviews.

The remaining parts of this paper are organised as follows. Section 2 presents background information on Crowdsourced Software Engineering. Section 3 describes practical platforms for Crowdsourced Software Engineering, together with their typical processes and relevant case studies. Section 4 provides a finer-grained view of Crowdsourced Software Engineering based on their application domains in software development lifecycle. Sections 5 and 6 describe current issues, open problems and opportunities. Finally Section 7 concludes.

2 BACKGROUND

We first review definitions of crowdsourcing (in general), before moving on to refine our focus to Crowdsourced Software Engineering.

2.1 Crowdsourcing

The term 'Crowdsourcing' was first published in the June 2006 *Wired* magazine article 'The Rise of Crowdsourcing', written by Jeff Howe [2]. In a companion blog post [1] to this article, the term was defined explicitly:

"Crowdsourcing represents the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call."

According to this definition, the undefined, large networked workforce and the open call format are the two prerequisites for crowdsourcing. Howe argues that crowdsourced work can be done by cooperation or by sole individuals [2].

This idea echoes the earlier book 'The Wisdom of the Crowds' [37] and also finds some resonance in the principles of Open Source Software (OSS) development [38]. Indeed, although the term 'crowdsourcing' has attracted significant recent attention, the underlying concepts can be found in many earlier attempts to recruit a large suitably-skilled yet undefined workforce in an open call for a specific task in hand. For example, we might trace the origins of crowdsourcing back to the Longitude competition in 1714, when the British government announced an open call (with monetary prizes), for developing a method to measure a ship's longitude precisely [39].

Turning to online crowdsourcing, early Internet-based crowdsourcing activities can be found in 2001, when 'InnoCentive' [40] was funded by Eli Lilly to attract a crowd-based workforce from outside the company to assist with drug development. In the same year, the TopCoder [41] platform was launched by Jack Hughes, as a marketplace using crowdsourcing for software development. To facilitate the online distributed software development activities, the TopCoder development method and system was proposed [42]. At the time of writing, TopCoder is the world's largest platform for Crowdsourced Software Engineering. By March 2015, its community of software engineers had numbered 750,000 and it had already awarded over $67,000,000 in monetary rewards for the Crowdsourced Software Engineering tasks it facilitated.

There are many other definitions of crowdsourcing with subtle differences and nuances, which we review here. In Brabham's 2008 article [43], crowdsourcing was viewed as an online model for distributed production and problem-solving. The Wikipedia page on crowdsourcing [44] cites the definition which appeared in the Merriam-Webster dictionary in 2011 [45]. It stresses the large group of workers and an online community, but drops any mention of 'undefined labour' and 'open call' format [46]. Estellés-Arolas et al. [47] collected 40 definitions from 32 articles published during 2006 to 2011, and proposed an integrated definition. This definition is compatible with the ones we have introduced and specifies the mutual benefits between the workers and the requesters. Based on these previous definitions we can identify four common features that pertain to crowdsourcing: the open access in production, the flexibility in workforce, the free will in participation and the mutual benefits among stakeholders.

The claimed benefits of crowdsourcing include easy access to a wide range of workers, diverse solutions, lower labour rates and reduced time-to-market. The granularity of crowdsourced tasks can be as fine grained as photo tagging or as coarse grained as software development [48], [49]. A list of more than 160 crowdsourcing projects has been compiled [50] (using crowdsourcing to compile the list).

Crowdsourcing has been used extensively in various disciplines, such as protein structure prediction [6], [51], drug discovery [7], [52], transportation planning [8], [53], weather forecasting [9], [54], information retrieval [10], [55], and software engineering [5], [11], [56]–[59], to which we now turn.

2.2   Crowdsourced Software Engineering

We use the term 'Crowdsourced Software Engineering' to denote the applications of crowdsourcing techniques to support software development (in its broadest sense). Some authors refer to this as 'Crowdsourced
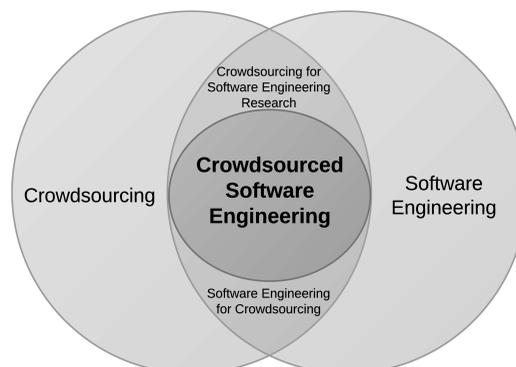

Fig. 3. Crowdsourcing and software engineering.

TABLE 3
Cited Crowdsourcing Definitions

Def.	None	Howe	Wiki	Own	Other
Count	138 (68%)	37 (18%)	4 (2%)	7 (3%)	22 (11%)

Software Development', 'Crowdsourcing Software Development' and 'Software Crowdsourcing' in previous studies [5], [60]–[66], but we prefer the term 'Crowdsourced Software Engineering' since it emphasises *any* software engineering activity included, thereby encompassing activities that do not necessarily yield software in themselves, such as requirements elicitation, test case refinement and project planning.

However, although our definition is inclusive of all software engineering activities, we wish to distinguish Crowdsourced Software Engineering from research activities on software engineering that happen to be supported by Crowdsourcing (see Fig. 3). Any research involving human subjects could potentially be supported by crowdsourcing, in the sense that the identification and recruitment of suitable human subjects for an experiment could be implemented using crowdsourcing techniques. In this application of crowdsourcing (to research studies), it is the identification of human subjects for experimentation that is important, rather than the particular research topic investigated.

If the research topic happens to be software engineering, then this work will be interesting to software engineers, but the principles and issues that arise will be more similar (and relevant) to those arising in other research involving human subjects. We call this application of crowdsourcing, 'crowdsourcing for software engineering research', to distinguish it from Crowdsourced Software Engineering. In this paper we comprehensively survey Crowdsourced Software Engineering. We do not claim to cover crowdsourcing for software engineering research as comprehensively, although we do also survey this topic for completeness. In addition, as shown in Fig. 3, software engineering techniques can also be used to support generic crowdsourcing [67], such kind of studies are out of scope for this survey.

Despite the wide usage of crowdsourcing in various software engineering tasks, the concept of Crowdsourced Software Engineering is seldom explicitly defined. According to our analysis (as shown in Table 3[4]), 68% of our surveyed papers use (or echo) the concept of crowdsourcing without citing any definition. Among the 34% that cite a definition, the most widely used definition is Howe's definition (18%), which we described earlier. Out of all the 203 publications we reviewed, only two give an explicit definition of what it means for crowdsourcing to be applied specifically to software engineering activities [5], [68].

Stol and Fitzgerald's definition [5] refines Howe's crowdsourcing definition to the software development domain, requiring the undefined labour force to have requisite specialist knowledge. The Dagstuhl seminar report's definition [68] was formalised as a Wikipedia page on software crowdsourcing [69]. It also specifies the tasks for software development, according to which the labour force can include anyone, but the characteristic of a large potential workforce is not mentioned.

Since the Howe's definition is the most widely accepted crowdsourcing definition in the papers we surveyed, we choose to define Crowdsourced Software Engineering simply as an instantiation of Howe's definition, as follows:

> *Crowdsourced Software Engineering is the act of undertaking any external software engineering tasks by an undefined, potentially large group of online workers in an open call format.*

Crowdsourced Software Engineering generally involves three types of actors (or stakeholders): Employers (aka requesters), who have software development work that needs to be done; Workers, who participate in developing software and Platforms, which provide an online marketplace within which requesters and workers can meet. Fig. 4 briefly depicts these three types of actors and the general process for Crowdsourced Software Engineering.

2.2.1 Claimed Advantages and Growth Trends in Crowdsourced Software Engineering

Crowdsourced Software Engineering has several potential opportunities compared to traditional software development methods. Crowdsourcing may help software development organisations integrate elastic, external human resources to reduce cost from internal employment, and exploit the distributed production model to speed up the development process.

For example, compared to conventional software development, the practice of TopCoder's crowdsourced software development has been claimed to exhibit the ability to deliver customer requested software assets with

---

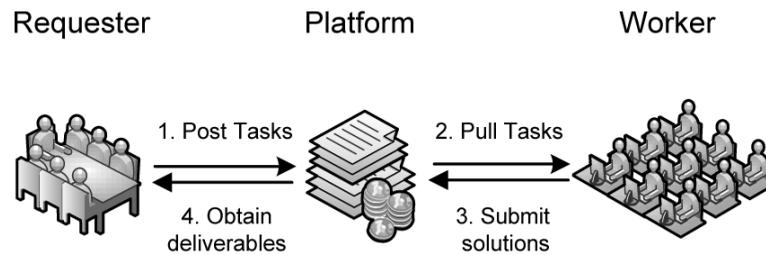4. One single paper may cite multiple definitions.

Fig. 4. Actors in Crowdsourced Software Engineering.

lower defect rate at lower cost in less time [3]. TopCoder claimed [70] that their crowdsourced development was capable of reducing costs by 30%-80% compared to in-house development or outsourcing. Furthermore, in the TopCoder American Online case study [3], the defect rate was reported to be 5 to 8 times lower compared with traditional software development practices.

In another study published in *Nature Biotechnology* [71], Harvard Medical School adopted Crowdsourced Software Engineering to improve DNA sequence gapped alignment search algorithms. With a development period of two weeks, the best crowd solution was able to achieve higher accuracy and 3 order of magnitude performance improvement in speed, compared to the US National Institutes of Health's MegaBLAST.

The increasing popularity of Crowdsourced Software Engineering revolves around its appeal to three different related stakeholders:

1) Requesters: Crowdsourced Software Engineering is becoming widely accepted by companies and organisations, from the military domain, academic institutions to large IT companies. DARPA created Crowdsourced Formal Verification (CSFV) program [72] for software formal verification and launched the Verigames website to facilitate the practice [73]. NASA and Harvard business school established the NASA Tournament Laboratory for crowdsourcing software solutions for NASA systems [74]. Microsoft crowdsourced partial software development activities in Office 2010 [75], Windows 8.1 [76] and Windows 10 [77].

2) Workers: Based on an industrial report from Masssolution [78], the number of workers engaged in software crowdsourcing increased by 151% in the year 2011.

3) Platforms: There is a growing number of crowdsourcing platforms built for software development domain, such as AppStori [79] and Mob4Hire [80]. These commercial platforms will be described in more detail in Section 3.

The flourishing Crowdsourced Software Engineering landscape is also revealed by the increasing number of relevant publications published in recent years, as shown in Fig. 1. Crowdsourced Software Engineering is also proving to provide an attractive topic for student dissertations. Specifically, 20 out of the total 203 publications are Master/PhD theses. A detailed list of these theses can be found in Table 2.

2.3 Research Topics

To classify the papers, we first carefully analysed the 203 papers we collected, revealing four top-level categories based on their study type: Study of Practice, Theories and Models, Applications to Software Engineering and Evaluations of Software Engineering Research. We referred to the ACM Computing Classification System [81], the IEEE Taxonomy of Software Engineering Standards [82] and the 2014 IEEE Keywords Taxonomy [83], to formulate sub-categories for each of these four top-level categories. Specifically, for applications to software engineering, we created sub-categories based on different stages of software development life-cycle addressed by the applications. The detailed classification scheme of the research topics is given in Fig. 5.

We manually classified the collected papers and assigned them to each of the categories. The classification results were cross-checked by three authors, reaching an agreement of 91.2% in classification accuracy. The distribution of the literature over the research topics is shown in Fig. 6. The most prominent class is Applications to Software Engineering (62%), followed by theoretical study (20%) and practical study (15%) categories. A few studies (3% in our collection of papers) employed crowdsourcing to evaluate software engineering research. This type of publications may not use crowdsourcing-related keywords in their meta information. We performed extra manual retrievals for related research. Nevertheless, there may be more papers which fall into this category yet which remain uncovered in our survey; this category is not the focus of our survey.

3 CROWDSOURCING PRACTICE IN SOFTWARE ENGINEERING

In this section we describe the most prevalent crowdsourcing platforms together with typical crowdsourced development processes for software engineering. Since most case studies in the papers we collected were based
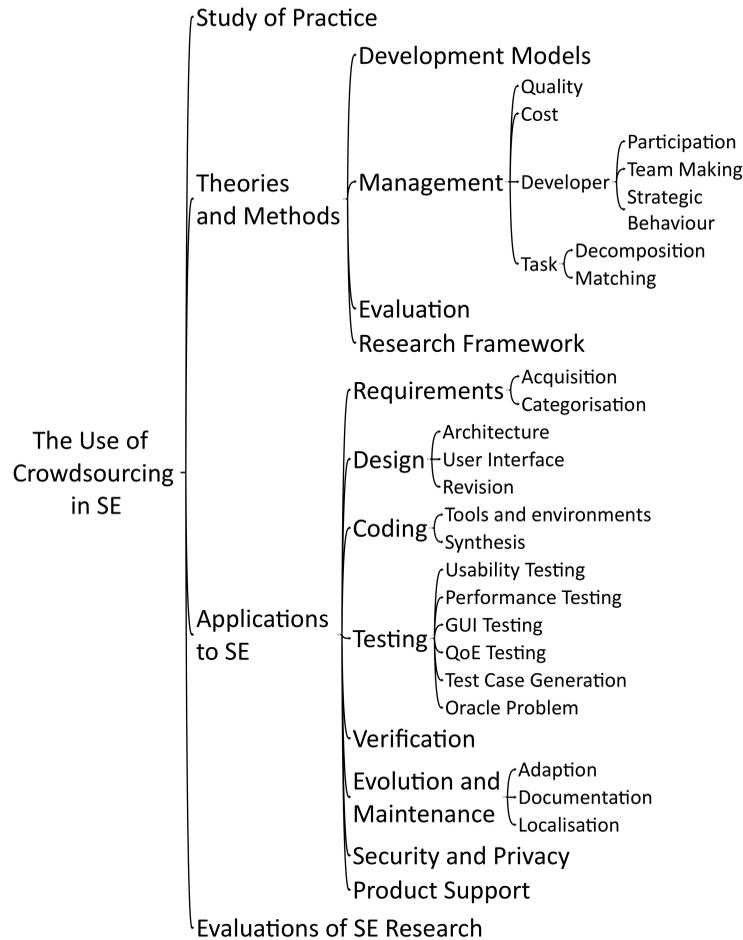
Fig. 5. Classification scheme of research on using crowdsourcing for software engineering (SE).
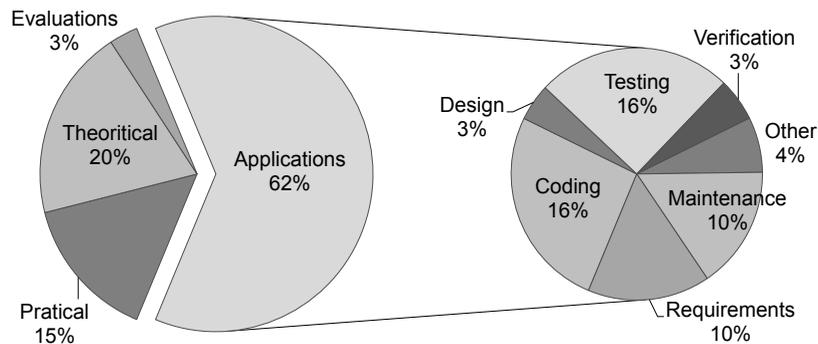
Fig. 6. Topic distribution.

on one (or several) of these commercial platforms, in the second part of this section, we present relevant case studies on the practice of Crowdsourced Software Engineering.

3.1 Commercial Platforms

Existing commercial crowdsourcing platforms that support software engineering are shown in Table 4. A timeline showing the year of their foundation is illustrated in Fig. 7. These platforms employ various types of open call formats, such as the most widely used online competition, on-demand matching, in which the workers are selected from the registrants, and online bidding, where the developers bid for tasks before starting their work. The platforms also focus on a broad range of task domains within software engineering. Platforms such as TopCoder and GetACoder support multiple types of software development tasks. Others are more specific. For example, uTest and BugCrowd are designed for software testing and security analysis

TABLE 4
A List of Commercial Platforms for Crowdsourced Software Engineering

Platform	URL	Task Domain	Open Call Form
TopCoder	www.topcoder.com	Software Development	Online Competition
GetACoder	www.getacoder.com	Software Development	Online Bidding
AppStori	www.appstori.com	Mobile App Development	Crowd Funding, Online Recruiting
Bountify	www.bountify.co	Small Coding Tasks	Online Competition
uTest	www.utest.com	Software Testing	On-demand Matching, Online Competition
Passbrains	www.passbrains.com	Software Testing	On-demand Matching
99Tests	www.99tests.com	Software Testing	On-demand Matching
TestBirds	www.testbirds.com	Software Testing	On-demand Matching
Testbats	www.testbats.com	Software Testing	On-demand Matching
Pay4Bugs	www.pay4bugs.com	Software Testing	On-demand Matching
CrowdTesters	www.crowdtesters.com.au	Software Testing	On-demand Matching
TestFlight	www.testflightapp.com	Mobile App Testing	On-demand Matching
Mob4hire	www.mob4hire.com	Mobile App Testing	Online Bidding
Testin	www.itestin.com	Mobile App Testing	On-demand Matching
Ce.WooYun	ce.wooyun.org	Software Security Testing	On-demand Matching
Bugcrowd	www.bugcrowd.com	Software Security Testing	Online Competition

respectively. There are also general crowdsourcing marketplaces such as Amazon Mechanical Turk, oDesk and Freelancer, which are not designed for software engineering specifically, but can nevertheless be used to support various software development tasks.

Different platforms may also use various process models. In the remainder of this subsection we introduce typical commercial platforms and their processes for Crowdsourced Software Engineering:

1) TopCoder: As a pioneer for practising Crowdsourced Software Engineering, TopCoder has its unique process and development model. It is known as the TopCoder Competition Methodology [84]. The platform can support independent graphic design, development, data science challenges, as well as the development of complex software (by decomposing into multiple sub-tasks). Viewed from the top level, the systematic process may resemble the waterfall model. However, each development phase is realised through a series of online competitions in which the crowd developers compete (with each other). Only qualified winning solutions are accepted. Qualified outputs are used as the inputs for the subsequent development phases. In this context, 'qualified' means passing a minimum acceptance score, rated through a review process.

The systematic development process starts from a requirements phase, in which the project goals, task plan and budget estimation are identified. This is achieved via communication between the project manager (who may come from the crowd or the platform) and the requester (who pays for the solutions offered by the crowd). This phase consists of several types of competitions such as Conceptualisation, Wireframe, Storyboards, UI Prototype and Specification. The outcome of this phase is a set of requirements specifications. These requirements are used as the inputs to the subsequent architecture phase, in which the application is decomposed into multiple components.

Each component further yields design and development tasks. The design challenges produce a set of documents such as UML diagrams and component specifications. Subsequent development challenges implement the corresponding design. The developed components are integrated in an assembly phase, certified by system level testing. In a subsequent deployment phase, the integrated solution is deployed into the requester's quality assurance environment. Finally, after passing a period of user acceptance testing, the solution is delivered to the requester. For further maintenance, new development challenges can be created to upgrade the current solution, while bug related challenges can be generated to identify and/or fix bugs.

2) AppStori: AppStori is a more recent platform for crowdsourcing mobile app development. Its development process is different from the TopCoder methodology. It uses a crowdfunding model to fund development and attracts app developers and consumers to work closely together. The crowd developers can post their projects to raise funds from the crowd or to recruit other developers for app implementation. Consumers can propose ideas for new app development, contribute money, act as beta testers and offer feedback on existing projects. The whole development process, from conception to release, is achieved through collaboration among crowd developers and consumers.

3) uTest: uTest is one of the several platforms for crowdsourced software testing. It currently claims to support the world's largest open community for software testing [85]. These crowd testers provide a wide
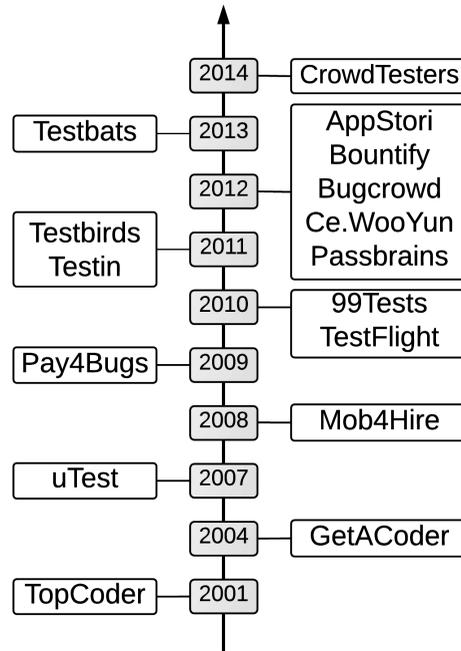
Fig. 7. The timeline depicting the foundation of major Crowdsourced Software Engineering platforms.

range of virtual on-demand testing services, such as functional testing, usability testing, localisation testing and load testing. The crowdsourced testing process commerces with a phase in which the clients can specify their testing needs. Flexible choices concerning testing device, operating system, geographic location and budget are provided by the platform. Appropriate testers are selected from the community: each project has an open call for testers and the crowd can apply to work on their desired testing tasks. However, only suitable testers (who are assessed based on their previous performance, skills, languages, testing devices, locations, etc.) will likely be favoured for selection. The selected testers report their testing work in real-time and submit their test report for approval. It is usually the clients' responsibility to review the submission and decide which workers are qualified to be paid for their work. The crowd testers are paid via the platform, based on such a per-project basis.

Other more general-purpose crowdsourcing platforms also have been widely used in software engineering research:

Amazon Mechanical Turk is a popular crowdsourcing marketplace for micro-tasks. By employing crowd workers on the platform to exploit human computation, small teams can avoid the challenges in developing complex software systems [86]. This platform has been employed to support program synthesis [57], graphical user interface (GUI) testing [87], oracle problem mitigation [88], and program verification [11] in software engineering.

StackOverflow [89] is a question and answer website which provides crowdsourced programming knowledge for software developers. Although such crowd knowledge does not contribute to the development of a software project directly, it has an impact on open source software development [90], [91] as well as conventional software development process. It has been used to improve integrated software development environments [25], [92]–[96] and software API documentation [97], [98].

Bountify is a platform similar to StackOverflow. However, it has more 'self-contained', micro programming tasks. Each will yield a payment of a certain amount of money, ranging from 1 to 100 US dollars. A study on program synthesis [57] used this platform to obtain initial seeds for their genetic programming algorithm.

Several studies provided further information on existing commercial platforms for software engineering. An introduction to software crowdsourcing platforms [99] briefly summarised several platforms for collaborative software development and compared crowdsourced software development with proprietary software development, outsourced software development and open source software development. Fried et al. [100] summarised three types of crowdsourcing platforms for the software industry: platforms such as Amazon Mechanical Turk [101] that support the use of human knowledge in an inexpensive way; platforms such as TopCoder that support contest-based software development; and platforms like MathWorks [102] that support

programming competitions with an unique 'competitive collaboration' feature. Wu et al. [61] proposed an evaluation framework for assessing software crowdsourcing processes from multiple objectives such as cost, quality, diversity of solutions and crowd competitions. The competition relationship was evaluated by a 'min-max' (defence-offence) mechanism adapted from game theory. Based on the proposed evaluation framework, the contrast between TopCoder and AppStori software crowdsourcing processes was illustrated.

3.2 Case Studies

Many Crowdsourced Software Engineering case studies have been reported in recent years. Most are based on one or several commercial platforms described above. Among them, the TopCoder platform has the most case studies reported upon in the literature [3], [5], [22], [62], [65], [103]–[105].

Stol et al. [5] presented an in-depth case study with a client company which has crowdsourced software development experience using TopCoder. A series of issues pertaining to the TopCoder development process were identified through interviews with the client company. For instance, the platform generally followed a waterfall model, which brought coordination issues to the client company as it adopted an agile development model. Also, quality issues were pushed to later stages in the TopCoder development process, which was not regarded as best practice. The research protocol [14] contains details of the design of this case study which can be used for replicating the study. Based on the lessons learned from this case study, the authors further enunciated their own advice for crowdsourced software development [106].

Tajedin and Nevo [65] also conducted an in-depth case study in the form of interviews, but from the perspective of TopCoder's management team, rather than the client. The case study revealed two types of value-adding actions that exist in the crowdsourcing platform, i.e., the macro, market level and the micro, transaction level actions.

Wu et al. [62] highlighted the lessons learned from their collected software crowdsourcing data. Two crowd-sourced software development processes employed by TopCoder and AppStori were examined. The paper argues that the 'min-max' competition behaviour contributes to the quality and creativity of crowdsourced software development.

Nag et al. [104] reported their collaboration with TopCoder to crowdsource spaceflight software development for the SPHERES Zero Robotics Program [107], supported by NASA, DARPA and Aurora Flight Sciences. The program was also used as a platform to crowdsource navigation software for satellite control [108]. The winners' solutions from the crowd were verified and validated on the SPHERES satellites within the International Space Station. More detailed description on this case can be found in Nag's master thesis [22].

Lakhani et al. [3] described the development of TopCoder from the year 2001 to 2009, including the evolution of the platform and the community, the benefits and concerns from the client's perspective, and the management roles and challenges of the TopCoder development process.

Archak [103] conducted an empirical analysis of developers' strategic behaviour on TopCoder. The cheap talk [109] phenomenon during the registration phase of the contest was identified, i.e., in order to soften competition, highly rated developers tend to register for the competition early thereby seeking to deter their opponents from seeking to participate in the market place. Archak argued that the cheap talk phenomenon and the reputation mechanisms used by TopCoder contribute to the efficiency of simultaneous online contests. In addition, a regression analysis was performed to study the factors that affect the quality of the contest outputs. The payment and the number of requirements factors were identified as significant predictors for final submission quality. Li et al. [105] also conducted a case study on TopCoder to identify important quality factors.

Regarding the case studies that are based on the platforms other than TopCoder: Zogaj et al. [15], [110] conducted a case study on a German start-up crowd testing platform called *testCloud*. Three types of challenges were highlighted in the case study: managing the crowd, managing the process and managing the techniques. Bergvall-Kareborn and Howcroft [111] reviewed Apple's business model for crowdsourcing mobile applications. By reporting fieldwork among Apple mobile app developers in three countries, they showed how the company benefited from crowdsourcing.

Some case studies have focused on region-specific practices in crowdsourced software development. For example, one case study [66], [112] presented the preliminary results of a multi-year study on crowdsourcing in the Brazilian IT industry. This study reported interviews that highlighted the generally low awareness of software crowdsourcing and concerns about the crowdsourced software quality. Phair's doctoral thesis [26] reported a qualitative case study on using crowdsourced software development to implement a web application for a non-profit organisation. Benefits such as measurable cost savings and an increased ability to work on multiple projects were identified. A few other case studies have reported the practice of software crowdsourcing in specific domains, such as crowdsourced proteomics software development [113] and crowdsourced e-government software development [114], [115].
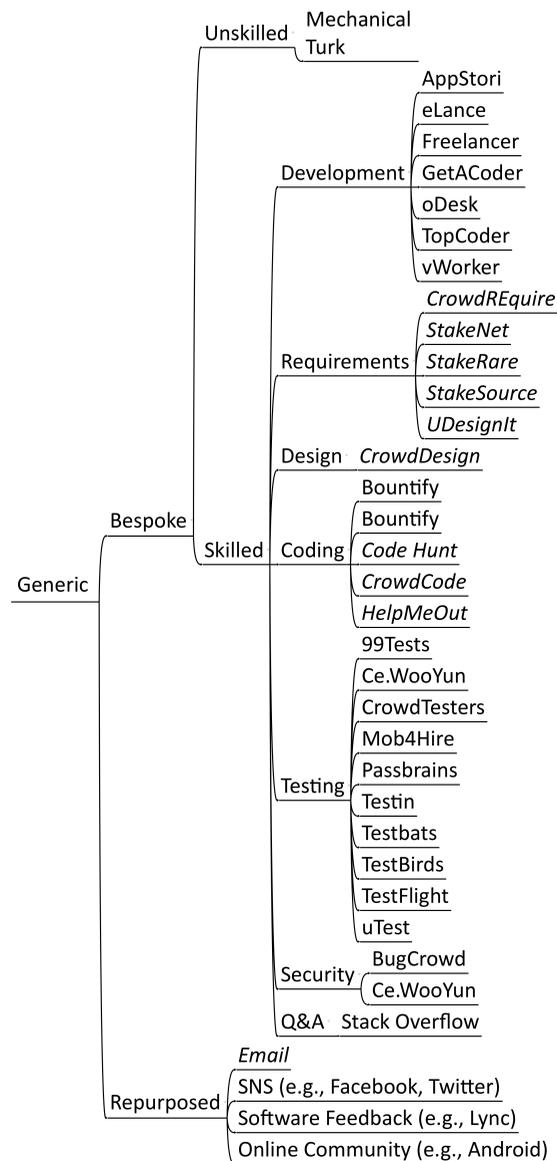
Fig. 8. Scheme of crowdsourced software engineering platforms. (The *italic text* indicates an experimental/non-commercial platform.)

4 CROWDSOURCING APPLICATIONS TO SOFTWARE ENGINEERING

Crowdsourcing applications to software engineering are presented as multiple subsections, according to the software development life-cycle activities that pertain to them. The following major stages are addressed: software requirements, software design, software coding, software testing and verification, software evolution and maintenance. An overview of the research on Crowdsourced Software Engineering is shown in Table 5. The commercial and experimental crowdsourcing platforms in these studies follow the classification scheme in Fig. 8.

A timeline of the introduction of vary ideas and concepts is illustrated in Fig. 9. For example, starting from 2009, crowdsourcing was employed to help evolve software and its localisation. Most recently, the crowdsourcing model was used for program synthesis. Other important events and theoretical/practical studies that can reflect the development of Crowdsourced Software Engineering are also illustrated in the timeline.

For the Crowdsourced Software Engineering studies with empirical evaluations, we summarised the conducted experiments in Table 6, to reveal the detailed experimental settings and results. With the summary, we calculated the distributions of the crowd size, cost and the platforms used in Crowdsourced Software Engineering experiments, as shown in Fig. 10, and Fig. 11 respectively.

TABLE 5
An Overview of the Research on Applying Crowdsourcing to Software Engineering (SE)

| SE Phase	SE Task	Why	Bespoke Tool	Stakeholder			Ref.
				Requester	Platform	Worker	
Requirements	Requirements Acquisition	Cost, User needs, Domain knowledge, Automation, Quality	StakeSource, StakeSource2.0, StakeNet, StakeRare,iRequire	Requirements engineers, Designers, Software teams, Researchers	Email, StakeSource, StakeSource2.0, StakeNet, StakeRare, CrowdREquire, UDesignIt, Bespoke, AOI, AMT	All stakeholders, Users, Undefined crowd	[56], [116]–[131]
	Requirements Categorisation	User needs	None	Requirements engineers, Designers	Unspecified	Users	[123], [124]
Design	User Interface Design	User needs, Quality, Diversity	None	Designers, Non-technical end users	Bespoke, AMT, CrowdDesign, Email	Users	[132]–[135]
	Architecture Design	Quality, Diversity	None	Researchers	Email	Designers	[134]
	Design Revision	Quality, Diversity	None	Researchers	Email	Designers	[134]
Coding	IDE Ehancement	Debugging, API aid	BlueFix, Calcite, Example Overflow, Seahawk, Prompter, SnipMatch	Developers	HelpMeOut, Stack Overflow, oDesk	Developers	[20], [25], [27], [33], [92]–[96], [136]–[143]
	Program Optimisation	Human solutions	None	Developers, Researchers	Bountify, AMT	Developers, Undefined crowd	[57]
	Crowd Programming Support	Automation, Human solutions	Jabberwocky, AutoMan, TurKit, CrowdLang, CIDRE, Collabode	Developers, Teachers	Bespoke, AMT, Code Hunt, CrowdCode	Users, Developers	[28], [144]–[153]
Testing	Usability Testing	Cost, Time	CrowdStudy	Testers	CrowdStudy, Bespoke, AMT, CrowdFlower	Users	[30], [154]–[159]
	Performance Testing	Real-world measure	None	Client companies	Lync	Users	[160]
	GUI Testing	Cost, Scalebility	None	Testers	AMT	Undefined crowd	[87], [161]
	QoE Testing	Cost, Diversity	Quadrant of Euphoria	Researchers	Quadrant of Euphoria, Bespoke, AMT, Microworkers	Undefined crowd	[162]–[165]
	Test Generation	Human inputs	PAT	Testers, Researchers	Twitter	Undefined crowd	[166], [167]
	Oracle Problem Mitigation	Human solutions, Automation	None	Testers, Researchers	AMT	Qualified / Unqualified crowd	[88]
	Crowd Testing Support	Human inputs	CrowdBlaze	Testers, Researchers	Bespoke, AMT, Mobileworks, Email	Undefined crowd	[34], [168], [169]
	General Evaluation	User needs, Diversity	None	Researchers	Bespoke, AMT	Users	[170]–[172]
Verification	Non-expert Verification	Cost, Speed	Verification Games, VeriWeb	Developers, Researchers	Bespoke, AMT, vWorker	Undefined crowd	[11], [12], [173], [174]
Evolution	Software Adaptation	User needs, Cost, Diversity, Speed	MoWA, CrowdAdapt	Developers, Designers, Users, Researcher	Bespoke, Facebook, Online community	Users	[120], [175]–[185]
Maintenance	Software Documentation	Domain knowledge	COFAQ	Developers, Researchers	Q&A, Stack Overflow, SciPy Community	Developers, Researchers	[97], [98], [140], [186], [187]
	Software Localisation	Domain knowledge, Cost, Speed	None	Developers, Researchers	AMT	Undefined crowd	[19], [24], [29], [188]
Other	Security and Privacy Augmentation	Diversity, Domain knowledge, User needs	Crowdroid, Modding-Interface	Developers, Researchers	Android User Community	Users	[35], [189], [190]
	End User Support	Domain knowledge	LemonAid	Developers, Researchers	AMT	Users	[32], [191], [192]
	Software Ideation	User needs, Open innovation, Recruitment	SAPiens, IdeaMax	Client Companies	Repurposed, Bespoke	Users	[193]–[196]
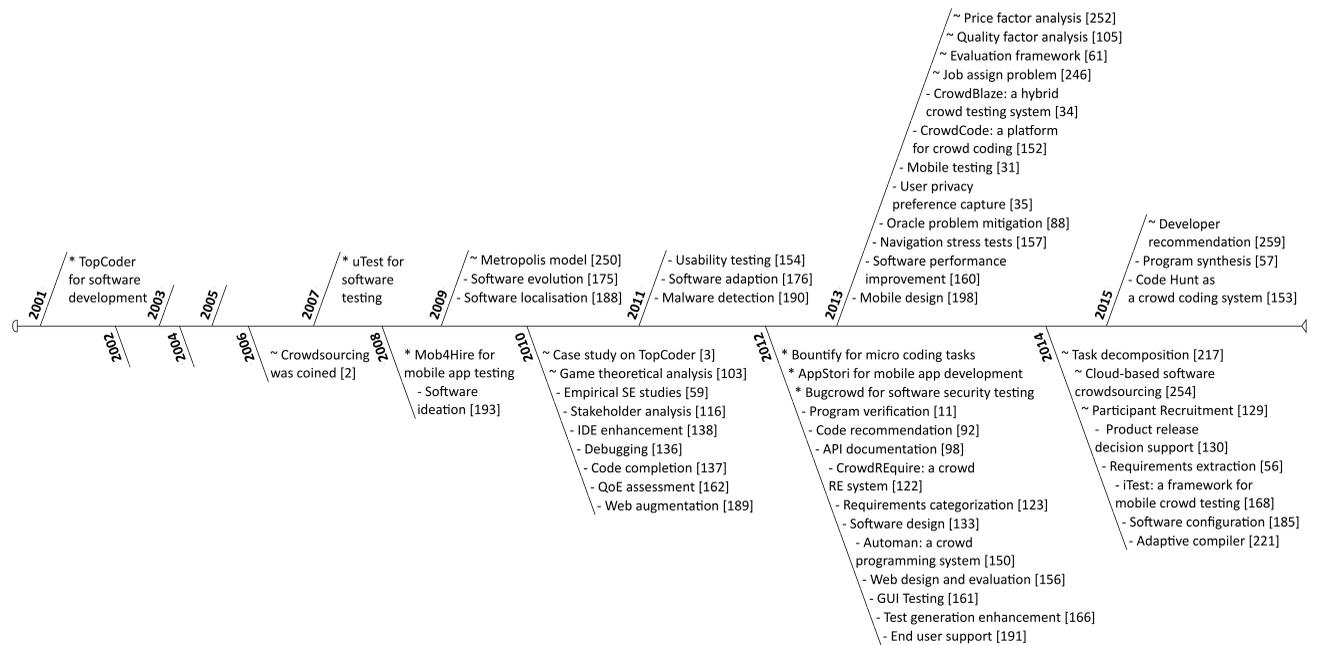
Fig. 9. Timeline on the development of Crowdsourced Software Engineering ('*' indicates the establishment of a platform. '~' shows the first practical/theoretical study and '-' stands for the first application work).

4.1 Crowdsourcing for Software Requirements Analysis

Requirements analysis is a widely accepted critical step that impacts the success of software projects [197]. A series of studies [56], [116]–[131] have investigated crowdsourcing to support this process.

Traditional stakeholder analysis tools require experts' manual effort to extract stakeholders' information. Lim et al. [116] proposed *StakeSource* to identify crowdsourced stakeholders involved in a stakeholder analysis process. This tool was designed to reduce the cost of reliance on experts to approach stakeholders. It was complementary to their previously proposed *StakeNet* [117], which recommends stakeholders via social networking. The authors further improved this tool and proposed *StakeSource2.0* [119]. The new version integrates support for identifying stakeholders and prioritising their requirements. *StakeSource2.0* was used to automate the stakeholder identification and prioritisation step of the *StakeRare* [121] method, an approach for large-scale requirements elicitation based on social network analysis and collaborative filtering techniques. Lim and Ncube [127] subsequently showed the application of the tool in system of systems projects. The tool is publicly available online[5].

Hosseini et al. [126] focused on employing crowdsourcing for requirements elicitation. They summarised the main features of the crowd and crowdsourcer in crowdsourced requirements engineering by reviewing existing literature. A preliminary result of a survey conducted on two focus groups was reported to reveal the relationship between these features and the quality of the elicited requirements. Wang et al. [129] also used crowdsourcing to acquire requirements, but with a focus on overcoming the problem of recruiting stakeholders with specific domain knowledge. They proposed a participant recruitment framework, based on spatio-temporal availability. Their theoretical analysis and simulation experiments demonstrated the feasibility of the proposed framework.

The crowd stakeholders are not only a source of requirements, but also can help with requirements prioritisation and release planning. Nascimento et al. [123] investigated the use of crowdsourcing for requirements categorisation based on Kano's model. The model uses questionnaire to help classify requirements into five categories. The value of each requirement for a given user is identified in their approach. A framework was proposed for finding stakeholders involved in the process. Nayebi and Ruhe [130] presented the Analytical Open Innovation (AOI) approach to help developers make release decisions. The crowdsourcing model enables the AOI approach to systematically gather information from customers and other stakeholders. An illustrative case study was presented as a proof-of-concept to demonstrate the key ideas of the AOI approach.

Non-professional crowd workers have been used to process requirements documents. It is a laborious task to manually extract requirements from large natural language text source. However such data are frequently

5. http://www.cs.ucl.ac.uk/research/StakeSource

needed as the ground truth for evaluation. This limits the generalisation of evaluations to automatic requirements extraction methods. Breaux and Schaub [56] conducted three experiments concerned with employing untrained crowd workers to manually extract requirements from privacy policy documents. Experimental results indicated a 16% increase in coverage and a 60% decrease in cost of manual requirements extraction, with the help of their task decomposition workflow.

To support crowdsourced requirements engineering activities, Adepetu et al. [122] proposed a conceptualised crowdsourcing platform named *CrowdREquire*. The platform employs a contest model to let the crowd compete with each other to submit requirements specification solutions to the client defined tasks. The business model, market strategy and potential challenges such as quality assurance and intellectual property issues of the platform were also discussed.

4.2   Crowdsourcing for Software Design

Among existing commercial crowdsourcing marketplaces, there are many platforms supporting software interface design, such as 99designs, DesignCrowd and crowdSPING. However, few research studies have been reported on the performance of using crowdsourcing for software design.

In order to provide software designers inspiring examples during the wireframing stage, Huang et al. [198] leveraged the crowd to map between mobile app wireframes and design examples over the Internet. Lasecki et al. [135] proposed a crowdsourcing system named *Apparition* to help designers prototype interactive systems in real-time based on sketching and function description. Experimental results showed that *Apparition* was able to achieve an accuracy higher than 90% regarding user's intent, and to respond in only a few seconds.

Fewer crowdsourcing platforms support software architecture design. TopCoder is one of the widely used platforms. However, industrial crowdsourcing platforms such as TopCoder have limitations in evolving designs from multiple designers' solutions [134]. LaToza et al. [134] let designers produce initial designs and evolve their solutions based on others' solutions. Their study demonstrated the usefulness of recombination in crowdsourced software designs. A few suggestions on improving software design competitions were also highlighted based on their findings.

Nebeling et al. [133] also proposed to evolve software designs based data and functionality contributed by the crowd. However, the designs are specifically website components within the web engineering domain. Two preliminary experiments were conducted to show the capability of the proposed approach. Crowd motivation, quality assurance, security and intellectual property issues were also briefly discussed.

4.3   Crowdsourcing for Software Coding

Using crowdsourcing for software coding has focused on three sub-areas: integrated development environment (IDE) enhancement, crowd programming environment, and program optimisation:

1) IDE enhancement: Using crowd knowledge to support coding activities in integrated development environments has been extensively studied since 2010. Several tools and methods have been proposed to help the developers with coding and debugging [20], [25], [27], [33], [92]–[96], [136]–[143], each of which we describe below:

HelpMeOut [136] is a social recommender system that assists debugging with crowdsourced suggestions. The system has a database that stores fixes for coding errors constructed by crowd developers. For collecting the fixes, the system automatically tracks code changes over time and records actions that make the error code become error-free. The evaluation was performed with novice developers through two three-hour workshops. The results showed the proposed approach was able to recommend useful fixes for 47% of the errors. However, *HelpMeOut* only supports static, compiled programming languages such as Java. To further support dynamic, interpreted web programming languages, another tool named *Crowd::Debug* [199] was proposed.

The idea that a crowd of developers may be able to provide recommendations of patches for software systems finds a strong resonance in recent work on genetic improvement [200]–[203], and in particular work on automated bug fixing (aka 'patching' or 'automated program repair') [204]. Genetic improvement seeks to automatically improve software systems by suggesting modifications that improve functional and non-functional properties. Genetic improvement regards program code as genetic material to be manipulated in the automated search for improvements. Recent results have demonstrated the potential for this technique improve real work program's speed [200]–[203], [205], energy [206]–[208] and dynamic memory [209] consumption and functionality, both by fixing bugs [210] and by adding new features [211]. Work on automated repair has also harvested human developed patches in order to improve the automated reparation process [212]. It therefore seems reasonable to conclude that hybridised versions of automated repair and social recommender systems (like *HelpMeOut*) could be extremely successful, a topic to which we return in Section 6.3.

TABLE 6
An Overview of the Crowdsourcing Experiments Conducted in the Application Papers

| Phase	SE Task	Platform	Crowd	Size	Application	Effort	Reward	Result	Ref.
Requirements	Requirements Elicitation	StakeNet	Stakeholders	68	RALIC	-	-	StakeNet can identify stakeholders and their roles with high recall, and can prioritise them accurately.	[117]
	Requirements Elicitation	StakeRare	Stakeholders	87	RALIC	-	-	StakeRare can predict and prioritise stakeholder needs accurately.	[121]
	Requirements Extraction	AMT	Unskilled	76 / 38 / -	-	448 classifications / 135 classifications / -	$0.15 per task / $0.15 per task / $0.08-$0.10 per task	The approach can reduce 60% cost and increase 16% coverage in manual extraction.	[56]
Design	Architecture Design / User Experience Design	Email	Students	20 / 20	An educational traffic flow simulation program	avg. 12.9 h / avg. 21.3 h	$100 each person + 4*$1000 prizes	All participants borrowed others' design ideas and most improved the design quality.	[34]
Coding	IDE Enhancement	StackOverflow	SO community	-	35 development tasks: 12 Swing, 12 Boost, 11 LINQ	-	-	For 77.14% of the assessed tasks at least one useful Q&A pair can be recommended.	[96]
	IDE Enhancement - Code Annotation	oDesk	Developers	-	Codex	annotated 500 code snippets	-	Among the annotated snippets, 86% correspond to a useful programming task, 96% can be embedded into a standalone function, and 91% do not have another more common form.	[143]
	Program Synthesis	Bountify	Developers	5	Regular expressions	wrote 14 regular expression	$10	Consistent program boosts in accuracy can be achieved with modest monetary cost	[57]
Testing	System evaluation	AMT	Unskilled	65 / 69	Semantic search systems	579 HITs / 421 HITs	$0.20 per HIT, $347.16 in total	Crowdsourced evaluation tasks can be repeated over time and still maintain reliable results.	[170]
	Usability Testing	CrowdStudy + AMT	Unskilled	84	A news article page	33 different type tasks	-	The usefulness and the ability to be configured for different scenarios were demonstrated.	[158]
	Usability Testing	AMT	Unskilled	93 / 28	A subpage of a university website	28 custom layouts, 143 ratings and 32 answers / avg. 4-5m to answer all 8 questions	-	The crowdsourced usability testing shared similar results with a laboratory setting.	[157]
	Usability Testing	AMT + CrowdFlower	Unskilled	11 / 44	A graduate school's website	11 HITs / 44 HITs	$0.15 per HIT, $2.92 in total / $0.20 per HIT, $347.16 in total	Reduced cost and time. However quality was worse compared to the testing in a lab. setting.	[155]
	GUI Testing	AMT	Unskilled	398	Tribler, KDE login, KDE mount, Xfce	700 assignments	$0.10-$0.15 per HIT	The approach is feasible and reliable, although there was a quality issue in continuous testing.	[87]
	GUI Testing	AMT	Unskilled	100 / 100	Tribler	100 assignments, 28h58m / 100 assignments, 28h38m	$25 in total	The approach was able to evaluate an experimental user interface feature within a few days at low costs.	[161]
	Performance Testing	Lync	End users	48,000	Lync	Usage behaviours	0	The approach had been successfully deployed and had improved development decisions at Microsoft.	[160]
	QoE Testing	Quadrant of Euphoria + AMT	Unskilled	-	MP3 bit rate, VoIP quality, Video codec, Loss concealment	2130 runs of experiments	$21.3 in total	With consistency assurance crowdsourcing can yield results as well as laboratory experiments.	[162]
	QoE Testing	Microworkers	Unskilled	10,737 / 1,593	Videos	10,737 ratings / 1593 ratings	$0.2625 per rating / $0.0834 per rating	The proposed methods represented a step in making crowd-testing sufficiently mature for wide adoption.	[163]
	Test Generation	Twitter	Unskilled	120	PAT	avg. 1m per puzzle	0	84 of the top 100 constraint solving puzzles were successfully solved. 100 object mutation puzzles were successfully solved.	[166]
	Oracle Problem Mitigation	AMT	Unqualified	-	java.util.Stack, Trove4j, ISO8583 / java.util.Stack	200 assignments	-	CrowdOracles is a promising solution to mitigate the oracle problem; however getting useful results from an untrained crowd is difficult.	[88]
	Crowd Testing Support	Mobileworks + Email	End users	75	Video Hot Mix, Craigslist, Cnet, Amazon, Weather, Twitter, Evernote, Yelp	500 assignments	$0.15-$0.20 per assignment	Statically Aided Interactive Dynamic Analysis consistently obtained greater coverage than other techniques.	[34]
Verification	Non-expert Verification	vWorker / AMT	Developers / Unskilled	14 / < 10	StackAr / -	avg. 3 h / -	$6-$22 per hour / >$0.25 per HIT	VeriWeb can save time and money with contracted workers. Current ad-hoc labours are not well-suited for verification.	[11]
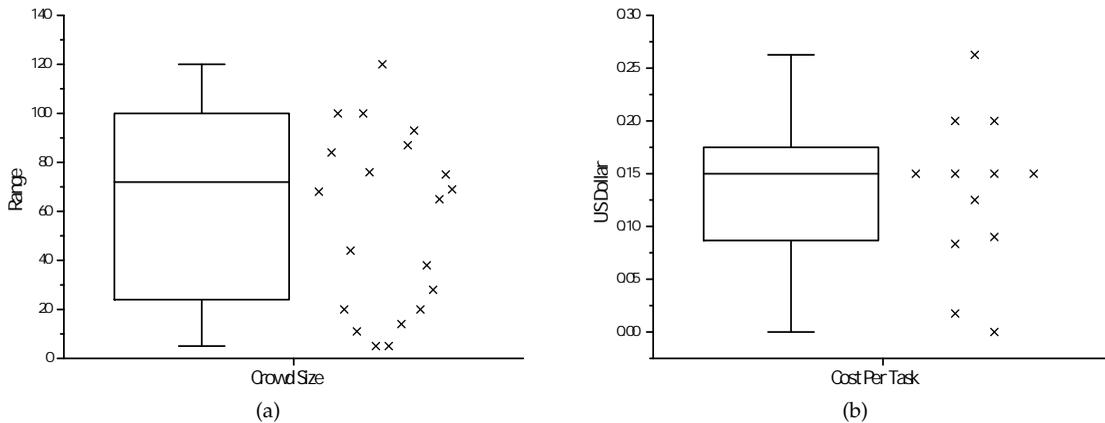
Fig. 10. Crowd size (a) and cost per task (b) in the experiments.

BlueFix [139] is an online tool concerned with the problem of compiler error message interpretation and understanding for novice programmers. An evaluation was performed based on an audience of 11 novice student programmers. The results indicated that the tool was able to help the students fix compile-time errors faster, and when compared with *HelpMeOut*, *BlueFix*'s suggestions were 19.52% higher in precision.

Calcite [137] is an Eclipse plugin that specifically focuses on constructor API comprehension and correct usage. The plugin uses a database that contains common object construction examples by collecting code from the web. According to a reported user study, this plugin can help developers to increase their completion rate by 40%.

Example Overflow [92], [140] is a code search system which utilises crowd knowledge from question and answer (Q&A) websites for suggesting embeddable code with high quality. The code snippets were collected from Stack Overflow via its public API. The search function is based on Apache Lucene. A preliminary evaluation on a subset of coding tasks indicated that the system suggested results were better than other existing tools studied in the experiments.

Seahawk [25], [93]–[95] is an Eclipse plugin, the aim of which has some resonance with *Example Overflow*. It seeks to utilise crowd knowledge in Q&A websites such as *StackOverflow* for documentation and programming support. Compared to *Example Overflow*, *Seahawk* integrated Q&A services into IDEs and provided more friendly user interface features. For example, it was found to be better at formulating queries automatically, based on code entities and providing interactive search results. It also addresses the limitation of Q&A websites that they do not offer support for exploiting their data in a team-working context [93]. By enabling developers to link imported code snippets to their documents via language-independent annotations, *Seahawk* can help developers share documents with their teammates [95]. The evaluation experiments were performed on 35 exercises from Java training courses [94]. The results were generally promising. Although the tool might not always suggest useful documents, it sometimes aided developers with surprising insights.

Souza et al. [96] also aimed to use crowd knowledge from *StackOverflow*, but focused on proposing a ranking approach for potential solutions. The ranking strategy is based on two factors, including the quality of question-answer pairs and the textual similarity of the pairs regarding the developer's query. Experiments were performed on three programming topics. The results demonstrated that at least one suggested question-answer pair is helpful for 77.14% of the evaluated activities.

WordMatch and *SnipMatch* [33] are two search tools for helping developers integrate crowdsourced code snippets. *WordMatch* provides an end-user programming environment that enables users (without programming experience) to generate direct answers to search queries. *SnipMatch* is an Eclipse plugin built on *WordMatch* that retrieves customised, ranked source code snippets, based on current code context and the developer's search query. A similar plugin for Microsoft Visual Studio that has been used in practice is the Bing Code Search Addin [213], which can retrieve code snippets from the Internet.

Amann et al. [214] investigated on using crowd knowledge for method-call recommendations. Crowd knowledge was collected from multiple developers' implicit feedback on their context-sensitive usage of the APIs. Collaborative filtering techniques were employed for recommending method calls based on such feedback knowledge.

Bruch [27] proposed the idea of *IDE 2.0* (based on the concept of *Web 2.0*). Bruch showed how crowd knowledge can help improve multiple functions such as API documentation, code completion, bug detection and code search. Evaluations were performed on each of the proposed tools, revealing that the concept of
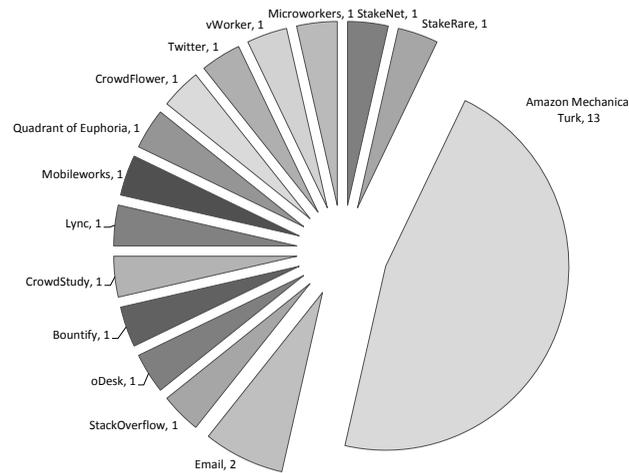
Fig. 11. Platforms used in the experiments.

Web 2.0 can be leveraged to improve the developer's IDE.

Fast et al. [143] conducted a study that echoes the idea of *IDE 2.0*. However, it focused on codifying emergent programming behaviour. By building a knowledge-based named *Codex*, which contained more than three million lines of popular Ruby code, novel data driven interfaces were constructed. For example, *Codex* was used for detecting unusual code that may contain a bug, annotating popular programming idioms identified by the system and generating utility libraries that capture emerging programming practice. According to Fast et al. [143], limitations of the current version of the proposed tool may include the adoption of GitHub, the only source of training data, which may introduce open sourced code with low quality.

Using the crowd knowledge to find common examples from the web, shares similarities with work on automatic harvesting of realistic test cases from the web-based systems [215], [216]. As with the potential for combination of genetic improvement and social recommenders, this similarity also points to the possibility of hybridise versions that harvest such information from a combination of crowd and web for testing purposes.

2) Crowd programming environment: Instead of adopting crowd knowledge over the Internet to help conventional coding activities, some other studies have focused on providing systems to support crowd-based coding work [28], [144]–[148], [152], [153], [217].

Goldman [145] proposed role-specific interfaces for coordinating collaborative crowd coding work. By building *Collabode*, a real-time web-based IDE, the author aimed to enable emerging highly-collaborative programming models such as crowd programming. Ball et al. [148] demonstrated the design of the Cloud-based Integrated Development and Runtime Environment (CIDRE), and its implementation *TouchDevelop* [218]. CIDRE consists of three components: a crowd developer community, an online IDE and an app store. These components link the IDE designers, application developers and users together and promote the mutual feedback among them during the development process.

Xie et al. [153] proposed to use *Code Hunt* [219] from Microsoft Research as a platform for crowd programming. The platform provides coding duel games with various difficulty levels to attract online developers' participation. By carefully designing the coding duels, the platform can serve for software construction purpose by leveraging the best solutions from the crowd. Also, by recording the crowd developers' duel solving process, the multiple attempts with evolving code versions can serve for education purpose.

In crowdsourcing both human and machine can be viewed as programmable units. There has been work on integrating the crowd as a part of the program code itself [146], [147], [149]–[151], [220]. We also discuss these support and integration studies:

Jabberwocky [149] is a programming framework for crowd computation. It consists of three components: *Dog*, *ManReduce* and *Domouse*. Bernstein et al. [220] proposed the view of programming the 'global brain'. The study pointed out some emerging research directions for better utilising the emerging human-computer network. *AutoMan* [150] was designed to help programmers focus on their programming logic while utilising human computation to solve tasks that are difficult for computers. It employs crowd workers for computation tasks and makes them callable as ordinary functions in a standard programming language. It is the first crowd programming system to be fully automated in managing budget, schedule and quality. Although the idea of adopting human intelligence as computational power dates back to the pre-electronic-computing era, in which the noun 'computer' typically referred to a human who preformed computation; the *AutoMan* framework, supported by modern electronic-computation-based networking and infrastructure, which dynamically

manages anonymous 'crowds' as a computational device via an API is novel.

3) Program optimisation: More recently, crowdsourcing has been used to support compilation optimisation [221] and program synthesis [57].

Auler et al. [221] presented a crowdsourced adaptive complier for JavaScript code optimisation. A compiler flag recommendation system was built in the cloud, based on the application performance data gathered from web clients. The system was used to guide the compiler to perform optimisation for a certain platform. Experiments were conducted on three optimisation implementations by JavaScript code emission for eight platforms. One of the best optimisation performance showed an average of five fold increase in execution speed.

Cochran et al. [57] proposed an approach called *Program Boosting*, which uses crowd knowledge and genetic programming techniques to help tackle hard programming tasks such as writing robust regular expressions for URLs. *Program Boosting* relies on two different types of crowds for 'boosting' a program: one 'expert' crowd for generating initial candidate programs and the other 'user' crowd for evaluating the outputs (e.g., the validity of URLs) generated from the candidate programs being evolved. The solutions obtained from the expert are used as the first population, which is subsequently evolved (by genetic programming) to yield improved solutions. Evaluation from the user crowd contributes to the evolution process. Experimental evaluation was performed on four regular expression writing tasks (to represent URLs, emails, phone numbers and dates). Experimental results showed that an average improvement of 16.25% in accuracy could be achieved on the initial human solutions.

4.4   Crowdsourcing for Software Testing and Verification

Software testing and verification have received considerable attention in the software engineering research community. It is therefore unsurprising that we found the number of related crowdsourcing studies dominate those of other categories.

4.4.1   Crowdsourcing for Software Testing

Crowdsourcing for software testing is often termed 'Crowdsourced Testing' or 'Crowd Testing'. Compared with traditional software testing, crowdsourced software testing has the advantage of recruiting, not only professional testers, but also end users to support the testing tasks.

Different from conventional beta testing which focused user acceptance, crowdsourced testing can be used to support various types of testing activities with different testing levels, in both black-box and white-box ways. It has been applied to usability testing [30], [154]–[159], performance testing [160], GUI testing [87], [161], test case generation [166], [167], and the oracle problem [88]. We discuss each of these below:

1) Usability Testing: Traditional usability testing is labour-intensive and can be expensive and time consuming [155]. Recruiting online ad-hoc crowd labour may be a way to ameliorate these issues, by exploiting a large potential user pool and providing lower labour rates with extended incentives to the end users. Crowdsourced usability testing has demonstrated its capability for detecting usability problems as good as the testing done by 'experts' [154]. However, Liu et al. [155] showed that the quality of crowdsourced usability testing was worse than that of the face-to-face usability testing in a laboratory setting. Nebeling et al. [158] further discussed this issue and suggested that the advantages outweigh disadvantages according to their results. Nevertheless these existing studies agree on the benefits of cost saving, fast delivery as well as easy access of crowdsourced usability testing.

Schneider and Cheung [154] first demonstrated the viability of employing on-demand crowd users for usability testing. They also proposed methods to help observe the testers during the process. Liu et al. [155] conducted a comparative study on crowdsourced and traditional laboratory usability testing. Their experimental results highlighted quality issues and the challenge of detecting 'cheating behaviour'. Nebeling et al. [156], [158] proposed a framework with a toolkit implementation named *CrowdStudy* for crowdsourced web site usability testing. For identifying outliers in the crowdsourced usability testing results, Gomide et al. [159] proposed an approach that employs a deterministic automata for automatic hesitation detection. The idea is to capture users' biofeedback from mouse movements and a skin sensor, for revealing their hesitation behaviours. This can be useful in filtering non-confirming usability testing results.

2) Performance Testing: Software performance in a real-world setting can be hard to test due to the various user behaviours and execution environments. Musson et al. [160] proposed an approach, in which the crowd was used to measure real-world performance of software products. The work was presented with a case study of the Lync [222] communication tool at Microsoft. The study indicated the usefulness of the approach for identifying performance issues and assisting development team with decision making. In this case the Lync software itself is repurposed as the crowdsourcing platform, and there is an implicit open call (i.e., permission

grant request) for providing performance data from the crowd users. Other similar cases for such crowdsourced performance testing include the Chrome's and Firefox's built-in telemetries (performance testing frameworks) [223]–[225] .

3) GUI Testing: Automated GUI test case generation is difficult, while manual GUI testing is too slow for many applications [226]. It is a challenging task to test a GUI continuously. Crowdsourcing is considered as a promising approach for continuous GUI testing [87].

Vliegendhart et al. [161] first proposed GUI testing for multimedia applications. Crowd testers were recruited from Amazon Mechanical Turk. They were asked to carry out A/B tests of user interfaces via remote virtual machines. Their experimental results indicated that it took less than three days and 50 US dollars to complete two featured GUI testing tasks with 100 assignments each. Based on this crowd performance, it was concluded that user connection speed was not an issue in their study. However the quality of the testing results was not reported in this study.

Dolstra et al. [87] also demonstrated the possibility of crowdsourcing GUI tests by offering remote virtual machines to testers, recruited from Amazon Mechanical Turk. The experimental results showed feasibility and reliability of the proposed approach.

4) Test Case Generation: Test cases are essential to ensure software quality. Although a number of automatic test case generation methods have been proposed, their test coverage is not ideal [227], due to several non-trivial tasks that are difficult for programs but may not be so hard for humans [166]. Chen and Kim [166] investigated object mutation and constraint solving issues, underlying existing test generation tools such as jCUTE [228], Randoop [229] and Pex [230]. A Puzzle-based Automatic Testing (PAT) environment was presented for decomposing and translating the object mutation and constraint solving problems into human-solvable games (gamification). Experimental results from two open source projects showed 7.0% and 5.8% coverage improvement, compared to the the coverage of two state-of-art test case generation methods.

Pham et al. [231] conducted a study on the testing culture of the social coding site — GitHub, and found that capable developers sometimes solve issues in others' repositories in a fast and easy manner, which is called the *drive-by commit* phenomenon. This phenomenon has the potential to be leveraged for generating test cases in social coding sites [167]. However it is still a conceptual idea which remains to be realised in future work.

5) Oracle Problem: An oracle is typically needed to determine the required output of a program for a given input [232], [233]. Such oracles may need to rely on human input [234], which makes it hard to fully automate software testing. Pastore et al. [88] investigated crowdsourcing to mitigate the oracle problem. They crowdsourced automatically generated test assertions to a qualified group of workers (with programming skills) and an unqualified group of workers on Amazon Mechanical Turk. Workers were asked to judge the correctness of the assertions and further fix false assertions. The experimental results suggested that crowdsourcing can be a viable way to mitigate the oracle problem, although the approach requires skilled workers provided with well-designed and documented tasks.

To support the application of crowdsourcing for software testing, especially for mobile application testing, several frameworks have been proposed [34], [168], [169]:

CrowdBlaze [34] is a crowd mobile application testing system which combines automatic testing and human-directed interactive testing. This study aimed to use redundant resources to help improve software systems. *CrowdBlaze* initially explores the app with static analysis and automatic testing, and then recruits crowd users to provide input for complex cases which enable automatic testing to further explore the app. By switching between automatic testing and crowdsourced testing, it aims at a higher testing coverage. Compared to employing automatic testing alone, the proposed system was demonstrated to cover 66.6% more user interfaces according to the evaluation results.

iTest [168] is a framework for mobile applications with more automation features than existing industrial mobile application testing service platforms such as uTest and Mob4Hire: the crowd testers are selected via a greedy algorithm, and the generated test results and logs in the framework are submitted automatically.

Caiipa [169] is a cloud service for scalable mobile application testing. The service framework is equipped with a unique contextual fuzzing approach to extend the mobile app running context space. It uses both crowdsourced human inputs and crowdsourced measurements, such as various network conditions, with multiple operator networks and different geographic locations. Experimental results suggested that *Caiipa* has the capability to uncover more bugs compared to existing tools with none or partial mobile contexts.

Xie [235] summarised three types of cooperative testing and analysis: human-tool, tool-tool and human-human cooperation. Crowd supported software testing and analysis falls into the human-human type of cooperation according to this study.

Besides, crowdsourcing has also been applied to general software evaluation [170]–[172] and more specific evaluation of Quality of Experience (QoE) [162]–[165].

4.4.2 Crowdsourcing for Software Verification

Current software verification techniques generally require skilled workers, thereby raising cost issues. Crowd-sourcing may reduce the skill barriers and costs for software verification [11], [12], [173], [174], [180].

DARPA published a solicitation for game-based large scale software verification in 2011, which is named the Crowd Sourced Formal Verification (CSFV) program [72]. A series of research and practice [73], [173], [236], [237] were conducted under this program. Dietl et al. [173] proposed to use gamification to attract a general crowd as a verification workforce. The 'verification games' approach transforms a verification task into a visual game that can be solved by people without software engineering knowledge.

Li et al. [174] presented a system called *CrowdMine* for recruiting non-expert humans to assist with the verification process. The system represents simulation or execution traces as images and asks the crowd of humans to find patterns that fail to match any pre-defined templates.

Schiller and Ernst [11] developed a web-based IDE called *VeriWeb* for reducing the barriers to verified specification writing. The IDE was designed to break down a verified specification writing task into manageable sub-problems. The experimental results suggested time and cost benefits. However, the workforce needs to be contracted workers rather than ad-hoc labours provided by crowdsourcing markets such as Amazon Mechanical Turk. A more detailed version of this study can be found in Schiller's doctoral thesis [12].

4.5 Crowdsourcing for Software Evolution and Maintenance

Software evolution and maintenance are among the earliest areas that have benefited from the application of crowdsourcing. A series of studies have investigated the potential of crowdsourced software evolution and maintenance [19], [24], [29], [97], [98], [175], [178], [179], [183]–[188].

4.5.1 Crowdsourced Software Evolution

Formal or automated verification methods may fail to scale to large software systems [175]. To help scalability, a market-based software evolution mechanism was proposed by Bacon et al. [175]. The goal of the mechanism is not to guarantee the absolute 'correctness' of software, but rather to economically fix bugs that users care about most. The proposed mechanism lets users bid for bug fixes (or new features) and rewards the bug reporters, testers and developers who respond. It was designed to apply to both crowdsourced settings and closed source systems.

Software adaptation aims to satisfy users' dynamic requirements. However, context is difficult to capture during the software design phase, and it is a challenging task to monitor context changes at runtime. Ali et al. [120] proposed *Social Sensing* to leverage the wisdom of the end users and used them as monitors for software runtime adaptation. This technique may help software designers (and their systems) to capture adaptation drivers and define new requirement and contextual attributes through users' feedback. A follow-up work of *Social Sensing* is *Social Adaptation* [179], in which several techniques (such as the goal model) for realising social sensing were further discussed. Also, evaluation of the proposed framework was performed on a socially adaptive messenger system. He et al. [183] proposed a 'suggestion model' to encourage crowd users to become more closely involved in commercial software runtime adaptation. A prototype and several adaptation strategies were introduced in this study. Challiol et al. [181] proposed a crowdsourcing approach for adapting mobile web applications based on client-side adaptation.

Nebeling and Norrie [176], [177] presented an architecture and visual supporting tools for facilitating crowdsourced web interface adaptation. Design and technical challenges when applying the crowdsourcing model, especially for quality control, were discussed. A tool named *CrowdAdapt* [182] was further implemented and evaluated. Experimental results showed the tool's capability in leveraging crowd users for generating flexible web interfaces.

In order to tackle the 'bloat' issue in enterprise applications, Akiki et al. [180] focused on utilising crowd-sourcing for user interface adaptations. Their proposed approach is based on model-driven user interface construction which enables the crowd to adapt the interfaces via an online editing tool. A preliminary online user study pointed to promising findings for usability, efficiency and effectiveness.

Users may become overwhelmed by the number of choices offered by software systems. In order to provide customised configuration dialogs to users, Hamidi et al. [185] proposed to extract configuration preferences from a crowd dataset. The optimised configuration dialogs were formed using a Markov Decision Process. When constructing customised dialogs, configuration decisions can be automatically inferred from knowledge elicited in previous dialogs. The evaluation of the method was performed on a Facebook dataset collected from 45 student users. Experimental results indicated that the proposed method could help users to reduce configuration steps by 27.7%, with a configuration prediction precision of 75%.

4.5.2 Crowdsourcing for Software Documentation

Software documentation plays a crucial role in program understanding. Previous studies have pointed out that inaccurate or insufficient documentation is a major cause of defects in software development and maintenance [238]–[240]. Several researchers have investigated crowdsourcing models to enhance software documentation [97], [98], [140], [186], [187].

Jiau and Yang [97] conducted an empirical study based on StackOverflow to reveal the severe uneven distribution of crowdsourced API documentation. To deal with the inequality, a reuse method based on object inheritance was proposed. An empirical evaluation was performed on three Java APIs: GWT, SWT and Swing. The results confirmed the feasibility of the documentation reuse methods with improved documentation quality and coverage.

Parnin et al. [98] conducted a similar empirical study, but with a focus on investigating the coverage and dynamics of API documentation supported by StackOverflow. Three APIs including the Java programming language, GWT and Android, were studied. The results showed that the crowd was able to generate rich content with API usage examples and suggestions. For example, for Android, 87% of its classes were covered by 35,000 developer contributed questions and answers. However, since the study is based on a single Q&A platform, there may exist issues in generalising the findings. Chen and Zhang [186] also studied crowd knowledge for API documentation. Documentation reading and searching behaviours were recorded for extracting question and answer pairs. Frequently asked questions were maintained for generating expanded API documentation automatically. In addition, *Seahawk* [94] described in Section 4.3 can also retrieving additional software documentation from Q&A services to help developers with program comprehension.

Pawlik et al. [187] conducted a case study on crowdsourced software documentation for *NumPy* (a Python library for scientific computing). The case study highlighted aspects that need to be considered when applying crowdsourcing for software documentation, e.g., technical infrastructure, stylistic instruction and incentive mechanism.

4.5.3 Crowdsourcing for Software Localisation

Software localisation is also relevant to 'software internationalisation' or 'globalisation' [19], such as tailoring the natural language output from systems for each country in which they are deployed. Localisation may be an important factor for the adoption and success of international products [241]. Research on utilising crowdsourcing for software localisation [19], [24], [29], [188] aim to reduce the cost and time-to-market periods of the traditional developer-based localisation process.

Exton et al. [188] first proposed the idea to use crowdsourcing for software localisation. Manzoor [19] developed a prototype for crowdsourced software localisation. An Action-Verification Unit method, together with a quality-oriented rewarding system, was proposed for quality control. The preliminary evaluation results showed that outcomes with acceptable quality can be delivered by the crowd. Gritti [24] also worked on a similar project and established a prototype system for crowdsourced translation and software localisation.

4.6 Crowdsourcing for Other Software Engineering Activities

Crowdsourcing has also been applied to software security and privacy analysis [35], [189], [190], software end user support [32], [191], [192] and software ideation [193]–[196].

The increasing number of mobile apps make malware analysis an urgent problem. Burguera et al. [190] amied to suggest an environment for submitting and analysing apps. Specifically, they presented a novel crowdsourced framework named *Crowdroid* for detecting malware in Android platforms. App behaviour traces were collected from real users in the crowd, and were subsequently used for differentiating malicious or benign apps. The experimental results showed a 100% detection rate in identifying malware executions in 3 self-written apps. In another real-world app experiment, the detection accuracies were 85% and 100% for two real malware specimens. Arellano [189] proposed crowdsourced web augmentation, based on the idea that end users are not only beneficiaries of web augmentation scripts, but can also contribute to them.

Users frequently struggle with reviewing permissions requested by mobile apps. Lin [35] collected the the permissions granted to mobile apps from a crowd consisting of over 700 mobile phone users. The collected privacy preferences were analysed using clustering algorithms, and the privacy profiles identified to be important were used for providing default permission settings for mitigating user burden. Evaluation, based on three fake apps and the crowd recruited from Amazon Mechanical Turk, indicated the resulting preference models were able to relieve users' burden in choosing privacy settings.

Regarding crowdsourced end user support, Chilana et al. [32], [191], [192] proposed *LemonAid*, a tool for providing contextual help for web applications, enhanced by crowd knowledge. The tool retrieves users' previously asked questions and answers in response to their user interface selections on screen. The evaluation

TABLE 7
Crowd Evaluation for Software Engineering Research

Ref.	SE Task	Size	Crowd	Platform	Effort	Application	Cost
[242]	Fault localisation	65	Developers	AMT	1,830 judgements	45 Java files collected from 5 textbooks	-
[59]	Evaluate the impact of code smells	50	End users programmers	AMT	160 HIT responses	Yahoo!Pipes	$0.2 per task
[136]	IDE enhancement	13	Students	Workshop	39 person-hours	HelpMeOut	-
[243]	Patch maintainability	157	Developers	Campus, AMT	2,100 judgements (filtered)	32 defects and 40 patches for 6 real word applications	$4 per task for AMT participants
[215]	Evaluate the readability of string test inputs	-	Developers	CrowdFlower	8 questions per task, 250 responses	Java methods from 17 open source projects	-
[247]	Evaluate the impact of code smells	61	End users programmers	AMT	366 task responses	Yahoo!Pipes	$0.25 per task
[248]	Survey on code search habits	99	Developers	Campus, AMT	10 questions per survey	-	-
[143]	Code annotation	-	Developers	oDesk	500 code snippets' evaluation	Codex	-

performed on Amazon Mechanical Turk showed that *LemonAid* was able retrieve at least one user support answer for 90% of the selection behaviours studied, and a relevant answer was likely to be in the top two results. Results from a field deployments across multiple sites suggested that over 70% of the end users were likely to find a helpful answer from *LemonAid* and might reuse the support system.

Software engineering research can also benefit from crowdsourcing. It can be used to conduct human studies [59], [136], [215], [242], [243]. We summarised a few studies on using crowd evaluation for software engineering research in Table 7. Note that we do not claim to have surveyed such crowdsourced human studies in software engineering research comprehensively, as this is not the focus of this study but it can be a direction for future work. The model can also be employed in organising broadly accessible software engineering contests [244] such as Predictive Models in Software Engineering (PROMISE), Mining of Software Repositories (MSR) and Search Based Software Engineering [245] (SBSE) challenges.

Several authors have anticipated that crowdsourcing will be applied to address more challenges in software engineering research [126], [166], [246].

5 ISSUES AND OPEN PROBLEMS

Despite the extensive applications for Crowdsourced Software Engineering, the emerging model itself faces a series of issues that raise open problems for future work. These issues and open problems have been identified by previous studies. But only a few research have discussed the solutions.

According to an in-depth industrial case study on TopCoder [5], key concerns including task decomposition, planning and scheduling, coordination and communication, intellectual property, motivation and quality challenges were highlighted as interesting and important challenges.

Several studies are concerned with suggesting potential research topics. Stol and Fitzgerald [13] presented a research framework inspired by the issues identified in the TopCoder case study [5]. It took the perspective of three key stakeholders, i.e., the requester, the platform and the worker. Research questions were proposed for issues identified from the view of each of the three stakeholders. LaToza et al. [4] briefly outlined a series of research questions concerning the division of crowd labour, task assignment, quality assurance and the motivation of the crowd's participation. A follow-up research agenda can be found in the recent paper [249].

In the remainders of this section, we discuss Crowdsourced Software Engineering issues together with relevant work in more detail:

5.1 Theory and Model Foundations

The use of undefined external workforce differentiates Crowdsourced Software Engineering from conventional software engineering. Existing software development theories and models may no longer apply to this emerging model [250]–[252].

In order to better facilitate Crowdsourced Software Engineering, a series of theories and models have been proposed. The first published theoretical model for Crowdsourced Software Engineering is the Metropolis Model proposed by Kazman and Chen [250], [251], who argued that classical software development models such as the waterfall model, the spiral model and the more recent agile models are not suitable for Crowdsourced Software Engineering.

The Metropolis Model distinguishes three types of roles, i.e., the platform (referred to as *kernel*), applications built on the *kernel* (referred to as *periphery*), and the end users (referred to as *masses*). Seven principles of the model were introduced for managing crowdsourced development.

Saxton et al. [253] subsequently analysed 103 crowdsourcing websites and provided a taxonomy of nine crowdsourcing models. Among them, the Intermediary Model and the Collaborative Software Development Model support Crowdsourced Software Engineering.

Tsai et al. [254] summarised the commonalities in different Crowdsourced Software Engineering processes and proposed an architecture for cloud-based software crowdsourcing. The architecture specifies a management web interface for the requesters, a series of development tools for online workers, worker ranking and recommendation tools provided by the platform, collaboration tools for multiple stakeholders, a repository for software assets and a cloud-based payment system.

A few studies have also considered game theoretic crowd formulations to understand competition among crowd developers [62], [255], [256]. Wu et al. identified the 'min-max' (defence-offence) nature of crowdsourced software development competitions and argued that the nature contributes to the quality and creativity of the produced software [62]. Hu and Wu [255] proposed a game theoretic model for analysing the competition behaviours among TopCoder developers. The conclusions of this paper were drawn based on theoretical analysis, e.g., Nash equilibria computation, without empirical evaluation, so the applicability of the model remains to be analysed in future work.

5.2 Task Decomposition

Crowdsourced complex tasks lead to heavy workloads and require dedicated resources. With the high skill barrier, it limits the number of potential workers. In order to increase parallelism and to expand the qualified labour pool, it is essential to decompose software engineering tasks into smaller pieces. However, software engineering tasks are often concerned with specific contexts, for which decomposition may be non-trivial. Several studies have focused on this decomposition problem.

LaToza et al. [217] developed an approach for decomposing programming work into microtasks. The method breaks down a single higher level task into multiple lower level tasks iteratively, and coordinates work by tracking changes linked to artifacts. A platform called *CrowdCode* [152] was implemented to support their proposed method. The evaluation was performed on a crowd of 12 developers and the results indicated that the approach had an 'overhead issue' which led to a potentially lower productivity compared to the traditional development methods. LaToza et al. [257] also proposed to decontextualise software development work as part of decomposition. Three types of development work including programming, debugging and design were discussed regarding their decontextualisation.

As discussed in the crowdsourcing applications for software testing and verification (Section 4), two previous studies also offered decomposition approaches: Chen and Kim [166] decomposed the test generators' complex constraint solving and object mutation problems into small puzzles, which can be solved by crowd labours. Schiller and Ernst [11] proposed an online IDE for verification named *VeriWeb*, which can decompose the verifiable specifications task into manageable sub-problems.

5.3 Planning and Scheduling

The highly heterogeneous nature of crowd labour necessitates careful planning and scheduling.

Tran-Thanh et al. [258] proposed a bounded multi-armed bandit model for expert crowdsourcing. Specifically, the proposed ε-first algorithm works in two stages: First, it explores the estimation of workers' quality by using part of the total budget; Second, it exploits the estimates of workers' quality to maximise the overall utility with the remaining budget. The evaluation of the proposed algorithm was based on empirical data collected from oDesk. The results indicated that the algorithm was able to outperform related state-of-the-art crowdsourcing algorithms by up to 300%.

Tung and Tseng [246] focused on using crowd resources effectively to support collaborative testing and treated the problem as an (NP-Complete) job assignment problem. They proposed a greedy approach with four heuristic strategies. To evaluate the proposed model, a Collaborative Testing System (COTS) was implemented. Experimental results showed the system was able to generate the average objective solution within

approximately 90% of the optimal solutions. When applied to a real-time crowd testing environment, the system was able to save 53% of the test effort.

In some open call formats such as online competition, the tasks are given to unknown developers rather than assigned to specific crowd participants. In such cases the developers cannot be directly scheduled but may be optimised using recommendation techniques to guide them to work on their most suitable tasks. Mao et al. [259] employed content-based technique to recommend developers for crowdsourced software development tasks. The approach learns from historical task registration and winner records to automatically match tasks and developers. Experimental results on TopCoder datasets indicated the recommendation performance was promising in both accuracy (50%-71%) and diversity (40%-52%).

Estimating the appropriate number of crowd developers and delivery time for Crowdsourced Software Engineering tasks is an important yet challenging problem. To date, very limited work has been done in this research area. Mäntylä and Itkonen [260] studied how the crowd size and allocated time can affect the performance of software testing. Their results, conduced on 130 students, indicated that multiple crowd workers under time pressure had 71% higher effectiveness (measured by number of detected bugs) than single worker without time pressure. The authors suggested that the number of crowd workers for manual testing tasks should be adjusted according to the effectiveness of the mechanisms and tools for detecting invalid and duplicate bug reports.

To guarantee the participation level in Crowdsourced Software Engineering tasks, Wang et al. [129] proposed a framework to support crowdsourcing systems in their recruitment of participants with domain knowledge for requirements acquisition. The framework was established based on the observation that crowd workers with similar domain knowledge tend to cluster in particular spatio-temporal regions. The feasibility of this framework was demonstrated by a theoretical study and a simulation experiment.

5.4   Motivation and Remuneration

Motivation is viewed as a critical factor for the success of a software project [261]–[263]. For crowdsourced software projects, developers without proper motivation may not be able to make consistent contributions, while inappropriate remuneration may lead to low capital efficiency or task starvation. Varshney [264] demonstrated that player motivation is essential for driving participation and ensuring a reliable delivery platform. Based on a study from IBM's internal crowdsourced software development system — *Liquid*, several intrinsic, extrinsic, and social motivation factors were identified. Developer participation was found to follow a power-law distribution. A momentum-based generative model and a thermodynamic interpretation were used to describe the observed participation phenomena.

Mao et al. [252] proposed 16 cost drivers for training empirical pricing models to meet crowd developers' monetary remuneration. Specifically, the development type (upgrade or new development) of the task, the number of component specifications, the number of sequence diagrams of the design and the estimated size of the task were considered as significant factors that impact the remuneration. Based on the identified cost drivers, nine predictive pricing models were trained using popular machine learning algorithms. Evaluation on 490 TopCoder projects indicated that high prediction quality was achievable.

Leimeister et al. [194] investigated the motivation of participants for IT-based idea competitions. Incentives such as organiser's appreciation, prizes and expert knowledge were highlighted in this study. Olson and Rosacker [265] discussed the motivation for participating in crowdsourcing and open source software (OSS) development. The element of altruism was considered to be important in motivating participation in both OSS and crowdsourced software development. Ramakrishnan and Srinivasaraghavan [266] presented intrinsic and extrinsic motivational factors among students in a crowdsourced programming task context. A controlled experiment was performed to show the viability of employing a captive university crowd for software development.

5.5   Quality Assurance

Crowd labour is transient and workers vary in expertise and background. The use of such an undefined work force inherently raises quality questions for crowdsourcing in general [267]–[269] as well as Crowdsourced Software Engineering [4], [5], [23], [105].

Li et al. [105] identified 23 quality factors for crowdsourced software development from the perspective of platform and project, based on an empirical study of TopCoder. Four important aspects were identified in order to improve crowdsourced software quality, including the prosperity level of the platform, the scale of the task, the participants' skill levels and the design quality of the task.

Saengkhattiya et al. [23] investigated how crowdsourcing companies deal with the quality assurance challenge by conducting interviews with four companies: Microworkers, Clickchores, Microtask and TopCoder.

Ten diverse methods for managing quality were identified, such as ranking/rating, reporting spam, reporting unfair treatment, task pre-approval, and skill filtering.

Tajedin and Nevo [63] built a 'success model' of crowdsourced software development, which contains three high-level determinants, namely the project characteristics, the crowd composition and the stakeholder relationship. The model was proposed based on the analysis of related studies on the success of information systems, OSS development and general software development.

Much of the work on quality assurance remains to be fully evaluated, leaving rigorous evaluation of Crowdsourced Software Engineering quality assurance as a pressing topic for future work.

5.6 Unexplored Issues

Unexplored issues in Crowdsourced Software Engineering include coordination and communication, intellectual property and data security problems. These issues also exist in general crowdsourcing and have relevant studies [253], [270], [271]. However, according to our analysis of the papers we were able to find for this study, they have not been explored under the specific Crowdsourced Software Engineering context.

Regarding the coordination and communication issue, both the resources and development process need to be coordinated. For example, geographically distributed and transient crowd workers need to reach a consistent understanding of the tasks required of them. Without coordination, it may be quite problematic, for example, when the crowdsourced developers and the requester use different development methods.

Intellectual property and data security is another important issue. Since crowdsourcing uses an open call format, the general public can access task information. Task requesters may find it difficult to describe the task as they can only provide limited information (for security reasons), while the crowdsourcing task needs to be as clear as possible. Intellectual property issues may arise when transferring the task deliverables. For example, it is possible that the crowd developers include pre-existing or third-party code intended for non-commercial use, but the client company actually requests the task for commercial purposes.

6 OPPORTUNITIES

This section outlines five ways in which the authors believe Crowdsourced Software Engineering may develop as it matures, widens and deepens its penetration into software engineering methods, concepts and practices.

6.1 Who is the Crowd?

Except few studies [180], [196], [264], [272], almost all previous work on Crowdsourced Software Engineering has assumed that the crowd will be *external* to the requester's organisation, recruited by an open call. Indeed, this external, open call format is part of the current definition of crowdsourcing. However, requesters could also identify specific crowds from their own organisation's employees, thereby extending the definition of what it means to be a crowd.

Crowdsourcing technology has provided platforms that support allocation of 'micro tasks'. Hitherto, the micro tasks have been a necessary part of the decomposition for distribution to a large external crowd. These micro-task allocation and collaboration platforms could be repurposed to support various forms of crowd-like software engineering *within* organisations, in which the crowd is formed, partly or wholly, of employees (or other stakeholders).

For example, an organisation could use crowdsourcing platforms to throw open acceptance testing of a newly procured system to a wider group of internal stakeholders than traditionally possible. Organisations already undertake such 'crowd like' acceptance testing activities, informally (and without infrastructural support), by inviting internal stakeholders to try out new products and provide comments. Crowdsourcing platforms could provide a technology and infrastructure to systematise, support and extend this existing informal activity.

Crowdsourcing technology could also be used to support internal training and dissemination of best practice. It could be used to harvest workarounds (currently used by employees to overcome software system limitations), or to elicit new requirements from the organisation. In this way, crowdsourcing infrastructure can be repurposed to help an organisation achieve greater involvement of its workforce in software system procurement and deployment.

More radically, perhaps *all* software systems that involve multiple users should, in future, be regarded as crowd-based software systems. This is not merely a philosophical perspective, but could have practical ramifications for enhanced adaptivity; by overlaying crowdsourcing technology a system could harvest and respond to it users. As the technologies develop, we may (hope to) witness a merging of crowdsourcing with adaptive software engineering [273]–[275].

6.2 Speculative Crowdsourced Software Engineering

Currently, Crowdsourced Software Engineering is envisaged as a way to replace existing software engineering activities with alternative versions implemented using the crowd. We anticipate that Crowdsourced Software Engineering will increasingly also open up new possibilities for software engineering activities that are *not* currently possible.

Specifically, the low-cost, flexibility and rapid response available through Crowdsourced Software Engineering may create possibilities for speculative software engineering, in which the software development process can become much more experimental. Crowdsourcing technology may provide a mechanism through which organisations can achieve even more rapid prototyping, with the crowd being used to simulate the functionality of a putative software system.

6.3 Hybrid Crowdsourced Software Engineering

The Crowdsourced Software Engineering solutions surveyed in this paper typically concern the substitution of a crowdsourced activity for an existing (non-crowdsourced) activity. In this regard, the solution is either crowdsourced or not crowdsourced, with a sharp 'binary divide' between the two kinds of activity. We envisage this binary divide becoming blurred as Crowdsourced Software Engineering achieves greater penetration into the research and practitioner communities.

This blurring of the distinction between traditional and crowd sourced activities will lead to the further development of Hybrid Crowdsourced Software Engineering. Tools such as *CrowdBlaze* (Section 4.4) already offer a form of hybridisation between crowdsourcing and automated software testing, while bug fix recommendation tools such as *HelpMeOut* could be augmented with genetic improvement (as mentioned in Section 4.3)
.

Hybrid Crowdsourced Software Engineering will require new processes and methodologies that feedback crowdsourced knowledge into software development process (as it proceeds) and that feed software development information back to the crowd. The growth in the use of app stores as a platform for software deployment and review [276]–[279], is already providing a kind of Hybrid Crowdsourced Software Engineering. The review mechanisms implemented by app stores already resemble a channel of communication between the users (a crowd) and an app's developers. We envisage greater deployment, extension and development of such crowdsourced software deployment, review and feedback infrastructures.

6.4 Multi-Crowdsourced Software Engineering

Current work on crowdsourcing typically involves a single crowd, which is given a well-defined, single task. We propose that this model can be generalised to Multi-Crowdsourced Software Engineering, in which multiple distinct (but communicating) crowds work on distinct (but related) problems. In Multi-Crowdsourced Software Engineering, the crowds communicate with each other, such that the behaviour of each is dependent upon the behaviour of the others. This interaction between distinct crowds distinguishes Multi-Crowdsourced Software Engineering from existing (single) Crowdsourced Software Engineering.

For example, the problem of testing and debugging could be formulated as a Multi-Crowdsourced problem, in which one crowd works on generating test cases to find bugs, while the other crowd works on finding patches to fix the bugs. This would lead to a crowd-based implementations of co-evolutionary mutation testing [280] and co-evolutionary patching [281].

Many other software engineering problems offer natural formulations for Multi-Crowdsourced Software Engineering. For example, requirements elicitation and rapid prototyping could proceed in tandem, with one crowd gathering requirements while the other develops prototypes for these requirements. Architectural evolution and software testing could also proceed in tandem using two crowds, one targeting performance test case generation and the other targeting architectural improvements to avoid performance bottlenecks. In a previous study on using crowdsourcing for program synthesis [57], one professional crowd was employed for generating regular expression candidates and another non-professional crowd was recruited for evaluating the instances generated from these expressions, for further evolving better programs.

Multi-Crowdsourced Software Engineering is not limited to two crowds. We could envisage a three-crowd software engineering problem involving requirements elicitation, rapid prototyping and software test case generation, each with their own dedicated crowd. Each of the three crowds will depend on the activities of the other, and use the outputs of the tasks undertaken by the other. The prototyping crowd implements some of the requirements emerging from the requirements elicitation crowd. The test case generation crowd generates tests, some of which will uncover issues in the prototypes. These issues, in turn, may suggest new features to the requirements elicitation crowd.

6.5 Iterative Crowdsourced Software Engineering

Most existing approaches to Crowdsourced Software Engineering consist of a single, batch mode, application of crowdsourcing to solve a well-defined single task. It is quite striking to see the waterfall-like model, making such a strong resurgence as a practical methodology underpinning much crowdsourced development work (see Section 3.1). This phenomenon maybe transitory; as it matures, Crowdsourced Software Engineering will likely become adaptive and iterative to better model the underlying software engineering processes it supports (e.g., a recent study shows the iterative recombination can help improve crowdsourced software design [134]). Indeed, our proposed Multi-Crowdsourced Software Engineering is a naturally iterative process, in which each crowd responds to and affects the results of tasks performed by other crowds.

7   CONCLUSIONS

This study has surveyed the use of crowdsourcing in software engineering activities and research into these activities. It has revealed an increasing trend in the publication numbers in recent years and has presented a snapshot of the research progress of this area from the perspectives of theories, practice and applications.

The paper has also highlighted potential issues in Crowdsourced Software Engineering, together with related analysis and solutions conducted in existing studies. The issues/opportunities that were insufficiently studied or explored have been pointed out in this survey, which may provide directions for future research.

APPENDIX A
SEARCH TERMS

TABLE 8
Terms for Online Library Search

Category	Terms
General	software crowdsourcing crowd software engineering crowdsourcing software engineering crowdsourced software engineering crowd software development crowdsourcing software development crowdsourced software development crowd development crowdsourcing development crowdsourced development
Domain	crowd requirements crowdsourcing requirements crowdsourced requirements crowd design crowdsourcing design crowdsourced design crowd coding crowdsourcing coding crowdsourced coding crowd testing crowdsourcing testing crowdsourced testing crowd verification crowdsourcing verification crowdsourced verification crowd software evolution crowdsourcing software evolution crowdsourced software evolution crowd software maintenance crowdsourcing software maintenance crowdsourced software maintenance

ACKNOWLEDGMENTS

REFERENCES

[1] J. Howe, "Crowdsourcing: A definition," http://crowdsourcing.typepad.com/cs/2006/06/crowdsourcing_a.html, June 2006.
[2] ——, "The rise of crowdsourcing," *Wired magazine*, vol. 14, no. 6, pp. 1–4, 2006.
[3] K. R. Lakhani, D. A. Garvin, and E. Lonstein, "TopCoder(A): Developing software through crowdsourcing," *Harvard Business School Case*, 610-032, January 2010.
[4] T. D. LaToza, W. Ben Towne, A. van der Hoek, and J. D. Herbsleb, "Crowd development," in *Proceedings of the 6th International Workshop on Cooperative and Human Aspects of Software Engineering*, May 2013, pp. 85–88.
[5] K.-J. Stol and B. Fitzgerald, "Two's company, three's a crowd: A case study of crowdsourcing software development," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 187–198.
[6] S. Cooper, F. Khatib, A. Treuille, J. Barbero, J. Lee, M. Beenen, A. Leaver-Fay, D. Baker, Z. Popović, *et al.*, "Predicting protein structures with a multiplayer online game," *Nature*, vol. 466, no. 7307, pp. 756–760, 2010.
[7] T. C. Norman, C. Bountra, A. M. Edwards, K. R. Yamamoto, and S. H. Friend, "Leveraging crowdsourcing to facilitate the discovery of new medicines," *Science Translational Medicine*, vol. 3, no. 88mr1, 2011.
[8] D. C. Brabham, T. W. Sanchez, and K. Bartholomew, "Crowdsourcing public participation in transit planning: preliminary results from the next stop design case," *Transportation Research Board*, 2009.
[9] A. T. Chatfield and U. Brajawidagda, "Crowdsourcing hazardous weather reports from citizens via twittersphere under the short warning lead times of EF5 intensity tornado conditions," in *Proceedings of the 47th Hawaii International Conference on System Sciences*. IEEE, 2014, pp. 2231–2241.
[10] O. Alonso, D. E. Rose, and B. Stewart, "Crowdsourcing for relevance evaluation," in *ACM SigIR Forum*, vol. 42, no. 2. ACM, 2008, pp. 9–15.
[11] T. W. Schiller and M. D. Ernst, "Reducing the barriers to writing verified specifications," in *Proceedings of the 27th ACM International Conference on Object-Oriented Programming Systems, Languages, and Applications*, 2012, pp. 95–112.
[12] T. W. Schiller, "Reducing the usability barrier to specification and verification," Ph.D. dissertation, University of Washington, 2014.
[13] K.-J. Stol and B. Fitzgerald, "Researching crowdsourcing software development: Perspectives and concerns," in *Proceedings of the 1st International Workshop on CrowdSourcing in Software Engineering*, 2014, pp. 7–10.
[14] K.-j. Stol and B. Fitzgerald, "Research protocol for a case study of crowdsourcing software development," Available from: http://staff.lero.ie/stol/publications, University of Limerick, 2014.
[15] S. Zogaj, U. Bretschneider, and J. M. Leimeister, "Managing crowdsourced software testing: A case study based insight on the challenges of a crowdsourcing intermediary," *Journal of Business Economics*, vol. 84, no. 3, pp. 375–405, 2014.
[16] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," School of Computer Science and Mathematics, Keele University, Technical Report EBSE-2007-01, 2007.
[17] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering - a systematic literature review," *Information and Software Technology*, vol. 51, no. 1, pp. 7 – 15, 2009.
[18] S. L. Lim, "Social networks and collaborative filtering for large-scale requirements elicitation," Ph.D. dissertation, University of New South Wales, 2010.
[19] J. Manzoor, "A crowdsourcing framework for software localization," Master's thesis, KTH Royal Institute of Technology, 2011.
[20] M. Kallenbach, "HelpMeOut-Crowdsourcing suggestions to programming problems for dynamic, interpreted languages," Master's thesis, RWTH Aachen University, 2011.
[21] S. Leone, "Information components as a basis for crowdsourced information system development," Ph.D. dissertation, Swiss Federal Institute of Technology in Zurich, 2011.
[22] S. Nag, "Collaborative competition for crowdsourcing spaceflight software and STEM education using SPHERES Zero Robotics," Master's thesis, Massachusetts Institute of Technology, 2012.
[23] M. Saengkhattiya, M. Sevandersson, and U. Vallejo, "Quality in crowdsourcing - How software quality is ensured in software crowdsourcing," Master's thesis, Lund University, 2012.
[24] A. Gritti, "Crowd outsourcing for software localization," Master's thesis, Universitat Politécnica de Catalunya, 2012.
[25] L. Ponzanelli, "Exploiting crowd knowledge in the IDE," Master's thesis, University of Lugano, 2012.
[26] D. Phair, "Open crowdsourcing: Leveraging community software developers for IT projects," PhD. in Computer Sci., Colorado Technical University, 2012.
[27] M. Bruch, "IDE 2.0: Leveraging the wisdom of the software engineering crowds," Ph.D. dissertation, Technische Universität Darmstadt, 2012.
[28] M. Goldman, "Software development with real-time collaborative editing," Ph.D. dissertation, Massachusetts Institute of Technology, 2012.
[29] A. Mijnhardt, "Crowdsourcing for enterprise software localization," Master thesis, Utrecht University, 2013.
[30] A. Teinum, "User testing tool towards a tool for crowdsource-enabled accessibility evaluation of websites," Master's thesis, University of Agder, 2013.
[31] O. Starov, "Cloud platform for research crowdsourcing in mobile testing," Master's thesis, East Carolina University, 2013.
[32] P. K. Chilana, "Supporting users after software deployment through selection-based crowdsourced contextual help," Ph.D. dissertation, University of Washington, 2013.
[33] D. Wightman, "Search interfaces for integrating crowdsourced code snippets within development environments," Ph.D. dissertation, Queen's University, 2013.
[34] H. Xue, "Using redundancy to improve security and testing," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2013.
[35] J. Lin, "Understanding and capturing people's mobile app privacy preferences," Ph.D. dissertation, Carnegie Mellon University, 2013.
[36] R. Snijders, "Crowd-centric requirements engineering: A method based on crowdsourcing and gamification," Master's thesis, Utrecht University, 2015.
[37] S. James, "The wisdom of the crowds," *New York: Randome House*, 2004.

[38] B. Kogut and A. Metiu, "Open-source software development and distributed innovation," *Oxford Review of Economic Policy*, vol. 17, no. 2, pp. 248–264, 2001.

[39] D. Sobel, *Longitude: The true story of a lone genius who solved the greatest scientific problem of his time.*   Macmillan, 2005.

[40] "Innocentive," http://www.innocentive.com, Accessed: 2015-03-01.

[41] "Topcoder," http://www.topcoder.com, Accessed: 2015-03-01.

[42] J. M. Hughes, "Systems and methods for software development," August 2010, US Patent 7778866 B2.

[43] D. C. Brabham, "Crowdsourcing as a model for problem solving an introduction and cases," *Convergence: the international journal of research into new media technologies*, vol. 14, no. 1, pp. 75–90, 2008.

[44] "Wikipedia for crowdsourcing," http://en.wikipedia.org/wiki/Crowdsourcing, Accessed: 2015-03-01.

[45] "New dictionary words for 2011," http://http://www.merriam-webster.com/info/newwords11.htm, Accessed: 2015-03-01.

[46] "Crowdsourcing definition," http://www.merriam-webster.com/dictionary/crowdsourcing, Accessed: 2015-03-01.

[47] E. Estellés-Arolas and F. González-Ladrón-De-Guevara, "Towards an integrated crowdsourcing definition," *Journal of Information Science*, vol. 38, no. 2, pp. 189–200, Apr. 2012.

[48] A. Kittur, B. Smus, S. Khamkar, and R. E. Kraut, "CrowdForge: Crowdsourcing complex work," in *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, 2011, pp. 43–52.

[49] L. Xiao and H.-Y. Paik, "Supporting complex work in crowdsourcing platforms: A view from service-oriented computing," in *Proceedings of the 23rd Australian Software Engineering Conference*, Apr. 2014, pp. 11–14.

[50] "List of crowdsourcing projects," http://en.wikipedia.org/wiki/List_of_crowdsourcing_projects, Accessed: 2015-03-01.

[51] F. Khatib, F. DiMaio, S. Cooper, M. Kazmierczyk, M. Gilski, S. Krzywda, H. Zabranska, I. Pichova, J. Thompson, Z. Popović, *et al.*, "Crystal structure of a monomeric retroviral protease solved by protein folding game players," *Nature Structural and Molecular Biology*, vol. 18, no. 10, pp. 1175–1177, 2011.

[52] R. Johnson, "Natural products: Crowdsourcing drug discovery," *Nature chemistry*, vol. 6, no. 2, pp. 87–87, 2014.

[53] A. Misra, A. Gooze, K. Watkins, M. Asad, and C. A. Le Dantec, "Crowdsourcing and its application to transportation data collection and management," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2414, no. 1, pp. 1–8, 2014.

[54] C. Muller, L. Chapman, S. Johnston, C. Kidd, S. Illingworth, G. Foody, A. Overeem, and R. Leigh, "Crowdsourcing for climate and atmospheric sciences: current status and future potential," *International Journal of Climatology*, 2015.

[55] M. Lease and E. Yilmaz, "Crowdsourcing for information retrieval," in *ACM SIGIR Forum*, vol. 45, no. 2.   ACM, 2012, pp. 66–75.

[56] T. D. Breaux and F. Schaub, "Scaling requirements extraction to the crowd: Experiments with privacy policies," in *Proceedings of the 22nd IEEE International Requirements Engineering Conference*, Aug. 2014, pp. 163–172.

[57] R. A. Cochran, L. D'Antoni, B. Livshits, D. Molnar, and M. Veanes, "Program boosting: Program synthesis via crowd-sourcing," in *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2015, pp. 677–688.

[58] M.-A. Storey, C. Treude, A. van Deursen, and L.-T. Cheng, "The impact of social media on software engineering practices and tools," in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ser. FoSER '10, 2010, pp. 359–364.

[59] K. T. Stolee and S. Elbaum, "Exploring the use of crowdsourcing to support empirical studies in software engineering," in *Proceedings of the 4th ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2010, pp. 1–4.

[60] Y. Usui and S. Morisaki, "An Approach for Crowdsourcing Software Development," *Proceedings of the Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*, pp. 32–33, 2011.

[61] W. Wu, W.-T. Tsai, and W. Li, "An evaluation framework for software crowdsourcing," *Frontiers of Computer Science*, vol. 7, no. 5, pp. 694–709, Aug. 2013.

[62] W. Wu, W. T. Tsai, and W. Li, "Creative software crowdsourcing: from components and algorithm development to project concept formations," *International Journal of Creative Computing*, vol. 1, no. 1, pp. 57–91, 2013.

[63] H. Tajedin and D. Nevo, "Determinants of success in crowdsourcing software development," in *Proceedings of the 2013 annual conference on Computers and people research*, 2013, pp. 173–178.

[64] X. L. Xu and Y. Wang, "Crowdsourcing Software Development Process Study on Ultra-Large-Scale System," *Advanced Materials Research*, vol. 989-994, pp. 4441–4446, July 2014.

[65] H. Tajedin and D. Nevo, "Value-adding intermediaries in software crowdsourcing," in *Proccedings of the 47th Hawaii International Conference on System Sciences*, Jan. 2014, pp. 1396–1405.

[66] R. Prikladnicki, L. Machado, E. Carmel, and C. R. B. de Souza, "Brazil software crowdsourcing: A first step in a multi-year study," in *Proceedings of the 1st International Workshop on CrowdSourcing in Software Engineering*, June 2014, pp. 1–4.

[67] M. Aparicio, C. J. Costa, and A. S. Braga, "Proposing a system to support crowdsourcing," in *Proceedings of the 2012 Workshop on Open Source and Design of Communication*, 2012, pp. 13–17.

[68] M. N. Huhns, W. Li, and W.-T. Tsai, "Cloud-based software crowdsourcing (Dagstuhl seminar 13362)," *Dagstuhl Reports*, vol. 3, no. 9, pp. 34–58, 2013.

[69] "Wikipedia for software crowdsourcing," http://en.wikipedia.org/wiki/Crowdsourcing_software_development, Accessed: 2015-03-01.

[70] "TopCoder - A platform for innovation overview," http://www.nasa.gov/pdf/651447main_TopCoder_Mike_D1_830am.pdf, Accessed: 2015-03-01.

[71] K. R. Lakhani, K. J. Boudreau, P.-R. Loh, L. Backstrom, C. Baldwin, E. Lonstein, M. Lydon, A. MacCormack, R. A. Arnaout, and E. C. Guinan, "Prize-based contests can provide solutions to computational biology problems," *Nature Biotechnology*, vol. 31, no. 2, pp. 108–111, 2013.

[72] "The crowd sourced formal verification (CSFV) program," http://www.darpa.mil/Our_Work/I2O/Programs/Crowd_Sourced_Formal_Verification_(CSFV).aspx, Accessed: 2015-03-01.

[73] "Verigames," http://www.verigames.com, Accessed: 2015-03-01.

[74] "Challenge platform: NTL," http://www.nasa.gov/sites/default/files/files/ntl-overview-sheet.pdf, Accessed: 2015-03-01.

[75] Wired, "How Microsoft crowdsourced the making of Office 2010," http://www.wired.com/2010/06/microsoft-office-2010, Accessed: 2015-03-01.

[76] A. Greenberg, "Microsoft finally offers to pay hackers for security bugs with 100,000 bounty," http://www.forbes.com/sites/andygreenberg/2013/06/19/microsoft-finally-offers-to-pay-hackers-for-security-bugs-with-100000-bounty, Accessed: 2015-03-01.

[77] TheTechieGuy, "How microsoft is cleverly crowdsourcing windows 10 development from its customers," http://thetechieguy.com/how-microsoft-is-cleverly-crowdsourcing-windows-10-development-from-its-customers, Accessed: 2015-03-01.

[78] Massolution, "Crowdsourcing industry report," http://www.crowdsourcing.org/editorial/enterprise-crowdsourcing-trends-infographic/18725, Accessed: 2015-03-01.

[79] "AppStori," http://www.appstori.com, Accessed: 2015-03-01.

[80] "Mob4Hire," http://www.mob4hire.com, Accessed: 2015-03-01.

[81] "The 2012 ACM computing classification system," http://www.acm.org/about/class/class/2012, Accessed: 2015-03-01.

[82] "IEEE standard taxonomy for software engineering standards," http://ieeexplore.ieee.org/servlet/opac?punumber=2601, Accessed: 2015-03-01.

[83] "2014 IEEE Taxonomy," https://www.ieee.org/documents/taxonomy_v101.pdf, Accessed: 2015-03-01.

[84] "TopCoder competition methodology," http://www.topcoder.com/wp-content/uploads/sampleprojects/1/index.html, Accessed: 2015-03-07.

[85] "uTest - Getting started," http://www.utest.com/getting-started, Accessed: 2015-03-23.

[86] A. Begel, R. DeLine, and T. Zimmermann, "Social media for software engineering," in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, 2010, pp. 33–38.

[87] E. Dolstra, R. Vliegendhart, and J. Pouwelse, "Crowdsourcing GUI tests," in *Proceedings of the 6th IEEE International Conference on Software Testing, Verification and Validation*, Mar. 2013, pp. 332–341.

[88] F. Pastore, L. Mariani, and G. Fraser, "CrowdOracles: Can the crowd solve the oracle problem?" in *Proceedings of the 6th IEEE International Conference on Software Testing, Verification and Validation*, Mar. 2013, pp. 342–351.

[89] "Stack Overflow," http://stackoverflow.com/, Accessed: 2015-03-01.

[90] B. Vasilescu, V. Filkov, and A. Serebrenik, "StackOverflow and GitHub: Associations between software development and crowd-sourced knowledge," in *Proceedings of the 2013 International Conference on Social Computing*, Sept. 2013, pp. 188–195.

[91] B. Vasilescu, A. Serebrenik, P. Devanbu, and V. Filkov, "How social Q&A sites are changing knowledge sharing in open source software communities," *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work and Social Computing*, pp. 342–354, 2014.

[92] A. Zagalsky, O. Barzilay, and A. Yehudai, "Example Overflow: Using social media for code recommendation," in *Proceedings of the 3rd International Workshop on Recommendation Systems for Software Engineering*, June 2012, pp. 38–42.

[93] A. Bacchelli, L. Ponzanelli, and M. Lanza, "Harnessing Stack Overflow for the IDE," in *Proceedings of the 3rd International Workshop on Recommendation Systems for Software Engineering*, June 2012, pp. 26–30.

[94] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Leveraging crowd knowledge for software comprehension and development," in *Proceedings of the 17th European Conference on Software Maintenance and Reengineering*, Mar. 2013, pp. 57–66.

[95] ——, "Seahawk: Stack Overflow in the IDE," in *Proceedings of the 35th International Conference on Software Engineering*, May 2013, pp. 1295–1298.

[96] L. B. L. de Souza, E. C. Campos, and M. D. A. Maia, "Ranking crowd knowledge to assist software development," in *Proceedings of the 22nd International Conference on Program Comprehension*, 2014, pp. 72–82.

[97] H. C. Jiau and F.-P. Yang, "Facing up to the inequality of crowdsourced API documentation," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 1, pp. 1–9, Jan. 2012.

[98] C. Parnin, C. Treude, L. Grammel, and M. Storey, "Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow," Georgia Institute of Technology, Tech. Rep., 2012.

[99] X. Peng, M. Ali Babar, and C. Ebert, "Collaborative Software Development Platforms for Crowdsourcing," *IEEE Software*, vol. 31, no. 2, pp. 30–36, 2014.

[100] D. Fried, "Crowdsourcing in the software development industry," *Nexus of Entrepreneurship and Technology Initiative*, 2010.

[101] "Amazon Mechanical Turk," https://www.mturk.com, Accessed: 2015-03-01.

[102] "MathWorks," http://www.mathworks.com, Accessed: 2015-03-07.

[103] N. Archak, "Money, glory and cheap talk: Analyzing strategic behavior of contestants in simultaneous crowdsourcing contests on TopCoder.com," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 21–30.

[104] S. Nag, I. Heffan, A. Saenz-Otero, and M. Lydon, "SPHERES Zero Robotics software development: Lessons on crowdsourcing and collaborative competition," in *Proceedings of the 2012 IEEE Aerospace Conference*, Mar. 2012, pp. 1–17.

[105] K. Li, J. Xiao, Y. Wang, and Q. Wang, "Analysis of the key factors for software quality in crowdsourcing development: An empirical study on TopCoder.com," in *Proceedings of the IEEE 37th Annual Computer Software and Applications Conference Analysis*, 2013, pp. 812–817.

[106] B. Fitzgerald and K.-J. Stol, "The dos and don'ts of crowdsourcing software development," in *SOFSEM 2015: Theory and Practice of Computer Science*, ser. Lecture Notes in Computer Science, 2015, vol. 8939, pp. 58–64.

[107] "Zero robotics," http://zerorobotics.mit.edu, Accessed: 2015-05-06.

[108] S. Nag, J. G. Katz, and A. Saenz-Otero, "Collaborative gaming and competition for cs-stem education using SPHERES zero robotics," *Acta Astronautica*, vol. 83, no. 0, pp. 145 – 174, 2013.

[109] J. Farrell and M. Rabin, "Cheap talk," *The Journal of Economic Perspectives*, vol. 10, no. 3, pp. 103–118, 1996.

[110] S. Zogaj and U. Bretschneider, "Crowdtesting with testcloud - managing the challenges of an intermediary in a crowdsourcing business model," in *Proceedings of the 21st European Conference on Information Systems*, 2013.

[111] B. Bergvall-Kåreborn and D. Howcroft, "The Apple business model: Crowdsourcing mobile applications," *Accounting Forum*, vol. 37, no. 4, pp. 280–289, Dec. 2013.

[112] L. Machado, G. Pereira, R. Prikladnicki, E. Carmel, and C. R. B. de Souza, "Crowdsourcing in the Brazilian it industry: What we know and what we don't know," in *Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies*, 2014, pp. 7–12.

[113] S. F. Martin, H. Falkenberg, T. F. Dyrlund, G. A. Khoudoli, C. J. Mageean, and R. Linding, "PROTEINCHALLENGE: crowd sourcing in proteomics analysis and software development." *Journal of Proteomics*, vol. 88, pp. 41–6, Aug. 2013.

[114] N. Shah, A. Dhanesha, and D. Seetharam, "Crowdsourcing for e-Governance: Case study," in *Proceedings of the 3rd International Conference on Theory and Practice of Electronic Governance*, 2009, pp. 253–258.

[115] J. Warner, "Next steps in e-government crowdsourcing," in *Proceedings of the 12th Annual International Digital Government Research Conference on Digital Government Innovation in Challenging Times*, 2011, pp. 177–181.

[116] S. L. Lim, D. Quercia, and A. Finkelstein, "StakeSource: Harnessing the power of crowdsourcing and social networks in stakeholder analysis," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, vol. 2, 2010, pp. 239–242.

[117] S. Lim, D. Quercia, and A. Finkelstein, "StakeNet: Using social networks to analyse the stakeholders of large-scale software projects," *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, vol. 2010, 2010.

[118] N. Seyff, F. Graf, and N. Maiden, "Using mobile re tools to give end-users their own voice," in *Proceedings of the 18th IEEE International Conference on Requirements Engineering*, 2010, pp. 37–46.

[119] S. L. Lim, D. Damian, and A. Finkelstein, "StakeSource2.0: Using social networks of stakeholders to identify and prioritise requirements," in *Proceeding of the 33rd international conference on Software engineering*, 2011, pp. 1022–1024.

[120] R. Ali, C. Solis, M. Salehie, I. Omoronyia, B. Nuseibeh, and W. Maalej, "Social sensing: When users become monitors," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11, 2011, pp. 476–479.

[121] S. L. Lim and A. Finkelstein, "StakeRare: Using social networks and collaborative filtering for large-scale requirements elicitation," *IEEE Transactions on Software Engineering*, vol. 38, no. 3, pp. 707–735, 2012.

[122] A. Adepetu, K. Ahmed, and Y. A. Abd, "CrowdREquire: A requirements engineering crowdsourcing platform," AAAI, Tech. Rep. Goodin 2005, 2012.

[123] P. Nascimento, R. Aguas, D. Schneider, and J. de Souza, "An approach to requirements categorization using Kano's model and crowds," in *Proceedings of the 16th IEEE International Conference on Computer Supported Cooperative Work in Design*, May 2012, pp. 387–392.

[124] N. Muganda, D. Asmelash, and S. Mlay, "Groupthink decision making deficiency in the requirements engineering process: Towards a crowdsourcing model," *SSRN Electronic Journal*, 2012.

[125] P. Greenwood, A. Rashid, and J. Walkerdine, "UDesignIt: Towards social media for community-driven design," *Proceedings of the 34th International Conference on Software Engineering*, pp. 1321–1324, June 2012.

[126] M. Hosseini, K. Phalp, J. Taylor, and R. Ali, "Towards crowdsourcing for requirements engineering," in *Proceedings of the 20th International working conference on Requirements engineering: foundation for software quality (Empirical Track)*, 2013.

[127] S. L. Lim and C. Ncube, "Social networks and crowdsourcing for stakeholder analysis in system of systems projects," in *Proceeding of the 8th International Conference on System of Systems Engineering*, June 2013, pp. 13–18.

[128] R. Snijders and F. Dalpiaz, "Crowd-centric requirements engineering," in *Proceedings of the 2nd International Workshop on Crowdsourcing and Gamification in the Cloud*, 2014.

[129] H. Wang, Y. Wang, and J. Wang, "A participant recruitment framework for crowdsourcing based software requirement acquisition," in *Proceedings of the 9th IEEE International Conference on Global Software Engineering*, Aug. 2014, pp. 65–73.

[130] M. Nayebi and G. Ruhe, "An open innovation approach in support of product release decisions," in *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, 2014, pp. 64–71.

[131] M. Hosseini, A. Shahri, K. Phalp, J. Taylor, R. Ali, and F. Dalpiaz, "Configuring crowdsourcing for requirements elicitation," in *Proceedings of the 9th International Conference on Research Challenges in Information Science*, 2015.

[132] M. S. Bernstein, "Crowd-powered interfaces," in *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, 2010, pp. 347–350.

[133] M. Nebeling, S. Leone, and M. Norrie, "Crowdsourced web engineering and design," in *Proceedings of the 12th International Conference on Web Engineering*, 2012, pp. 1–15.

[134] T. D. LaToza, M. Chen, L. Jiang, M. Zhao, and A. V. D. Hoek, "Borrowing from the crowd : A study of recombination in software design competitions," in *Proceedings of the 37nd ACM/IEEE International Conference on Software Engineering*, 2015.

[135] W. S. Lasecki, J. Kim, N. Rafter, O. Sen, J. P. Bigham, and M. S. Bernstein, "Apparition: Crowdsourced user interfaces that come to life as you sketch them," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, 2015, pp. 1925–1934.

[136] B. Hartmann, D. Macdougall, J. Brandt, and S. R. Klemmer, "What would other programmers do? Suggesting solutions to error messages," in *Proceedings of the 28th ACM Conference on Human Factors in Computing Systems*, 2010, pp. 1019–1028.

[137] M. Mooty, A. Faulring, J. Stylos, and B. a. Myers, "Calcite: Completing code completion for constructors using crowds," in *Proceedings of the 2010 IEEE Symposium on Visual Languages and Human-Centric Computing*, Sept. 2010, pp. 15–22.

[138] M. Bruch, E. Bodden, M. Monperrus, and M. Mezini, "IDE 2.0: Collective intelligence in software development," in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ser. FoSER '10, 2010, pp. 53–58.

[139] C. Watson, F. W. B. Li, and J. L. Godwin, "BlueFix: Using crowd-sourced feedback to support programming students in error diagnosis and repair," in *Proceedings of the 11th International Conference on Web-Based Learning*, 2012, pp. 228–239.

[140] O. Barzilay, C. Treude, and A. Zagalsky, "Facilitating crowd sourced software engineering via stack overflow," in *Finding Source Code on the Web for Remix and Reuse*.   Springer New York, 2013, pp. 289–308.

[141] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza, "Mining stackoverflow to turn the ide into a self-confident programming prompter," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 102–111.

[142] ——, "Prompter: A self-confident recommender system," in *Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution*, Sept 2014, pp. 577–580.

[143] E. Fast, D. Steffee, L. Wang, J. R. Brandt, and M. S. Bernstein, "Emergent, crowd-scale programming practice in the IDE," in *Proceedings of the 32nd annual ACM conference on Human factors in Computing Systems*, 2014, pp. 2491–2500.

[144] M. Goldman, G. Little, and R. C. Miller, "Real-time collaborative coding in a web IDE," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, 2011, pp. 155–164.

[145] M. Goldman, "Role-based interfaces for collaborative software development," in *Proceedings of the 24th Annual ACM Symposium Adjunct on User Interface Software and Technology*, 2011, pp. 23–26.

[146] P. Minder and A. Bernstein, "CrowdLang - First steps towards programmable human computers for general computation," in *Proceedings of the 3rd Human Computation Workshop*, JAN 2011, pp. 103–108.

[147] ——, "CrowdLang: A programming language for the systematic exploration of human computation systems," in *Proceedings of the 4th International Conference on Social Informatics*.   Lausanne: Springer, DEC 2012.

[148] T. Ball, S. Burckhardt, J. de Halleux, M. Moskal, and N. Tillmann, "Beyond open source: The touchdevelop cloud-based integrated development and runtime environment, Tech. Rep. MSR-TR-2014-63, May 2014.

[149] S. Ahmad, A. Battle, Z. Malkani, and S. Kamvar, "The jabberwocky programming environment for structured social computing," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, 2011, pp. 53–64.

[150] D. W. Barowy, C. Curtsinger, E. D. Berger, and A. McGregor, "AutoMan: A platform for integrating human-based and digital computation," in *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, 2012, pp. 639–654.

[151] G. Little, L. B. Chilton, M. Goldman, and R. C. Miller, "TurKit: Human computation algorithms on Mechanical Turk," in *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology*, 2010, pp. 57–66.

[152] T. D. LaToza, E. Chiquillo, W. Ben Towne, C. Adriano, and A. van der Hoek, "CrowdCode - A platform for crowd development," in *CrowdConf 2013*, 2013.

[153] T. Xie, J. Bishop, R. N. Horspool, N. Tillmann, and J. de Halleux, "Crowdsourcing code and process via Code Hunt," in *Proceedings of the 2nd International Workshop on CrowdSourcing in Software Engineering*, May 2015.

[154] C. Schneider and T. Cheung, "The power of the crowd: Performing usability testing using an on-demand workforce," in *Proceedings of the 20th International Conference on Information Systems Development Cutting edge research on Information Systems*, 2011.

[155] D. Liu, R. G. Bias, M. Lease, and R. Kuipers, "Crowdsourcing for usability testing," in *Proceedings of the American Society for Information Science and Technology*, vol. 49, no. 1, Jan. 2012, pp. 1–10.

[156] M. Nebeling, M. Speicher, M. Grossniklaus, and M. C. Norrie, "Crowdsourced web site evaluation with crowdstudy," in *Proceedings of the 12th International Conference on Web Engineering*, 2012, pp. 494–497.

[157] F. Meier, A. Bazo, M. Burghardt, and C. Wolff, "Evaluating a web-based tool for crowdsourced navigation stress tests," in *Proceedings of the 2nd International Conference on Design, User Experience, and Usability: Web, Mobile, and Product Design*, 2013, pp. 248–256.

[158] M. Nebeling, M. Speicher, and M. C. Norrie, "CrowdStudy: General toolkit for crowdsourced evaluation of web interfaces," in *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 2013.

[159] V. H. M. Gomide, P. A. Valle, J. O. Ferreira, J. R. G. Barbosa, A. F. da Rocha, and T. M. G. d. A. Barbosa, "Affective crowdsourcing applied to usability testing," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 1, pp. 575–579, 2014.

[160] 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 Software*, vol. 30, no. 4, pp. 38–45, July 2013.

[161] R. Vliegendhart, E. Dolstra, and J. Pouwelse, "Crowdsourced user interface testing for multimedia applications," in *Proceedings of the ACM multimedia 2012 workshop on Crowdsourcing for multimedia*, 2012, pp. 21–22.

[162] K.-t. Chen, C.-j. Chang, A. Sinica, C.-c. Wu, Y.-c. Chang, and C.-l. Lei, "Quadrant of Euphoria: A crowdsourcing platform for QoE assessment," *IEEE Network*, no. April, pp. 28–35, 2010.

[163] B. Gardlo, S. Egger, M. Seufert, and R. Schatz, "Crowdsourcing 2.0: Enhancing execution speed and reliability of web-based QoE testing," in *Proceedings of the 2014 IEEE International Conference on Communications*, June 2014, pp. 1070–1075.

[164] T. Hossfeld, C. Keimel, M. Hirth, B. Gardlo, J. Habigt, and K. Diepold, "Best practices for QoE crowdtesting : QoE assessment with crowdsourcing," *IEEE Transactions on Multimedia*, vol. 16, no. 2, pp. 541–558, 2014.

[165] T. Hossfeld, C. Keimel, and C. Timmerer, "Crowdsourcing quality-of-experience assessments," *Computer*, pp. 98–102, 2014.

[166] N. Chen and S. Kim, "Puzzle-based automatic testing: Bringing humans into the loop by solving puzzles," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, 2012, pp. 140–149.

[167] R. Pham, L. Singer, and K. Schneider, "Building test suites in social coding sites by leveraging drive-by commits," in *Proceedings of the 35th International Conference on Software Engineering*, May 2013, pp. 1209–1212.

[168] M. Yan, H. Sun, and X. Liu, "iTest: Testing software with mobile crowdsourcing," in *Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies*, 2014, pp. 19–24.

[169] C.-J. M. Liang, N. D. Lane, N. Brouwers, L. Zhang, B. F. Karlsson, H. Liu, Y. Liu, J. Tang, X. Shan, R. Chandra, and F. Zhao, "Caiipa : Automated large-scale mobile app testing through contextual fuzzing," in *Proceedings of the 20th annual international conference on Mobile computing and networking*, 2014.

[170] R. Blanco, H. Halpin, D. M. Herzig, P. Mika, J. Pound, H. S. Thompson, and T. Tran Duc, "Repeatable and reliable search system evaluation using crowdsourcing," in *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2011, pp. 923–932.

[171] N. Sherief, N. Jiang, M. Hosseini, K. Phalp, and R. Ali, "Crowdsourcing software evaluation," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, 2014, pp. 1–4.

[172] N. Sherief, "Software evaluation via users' feedback at runtime," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, 2014, pp. 1–4.

[173] W. Dietl, S. Dietzel, M. D. Ernst, N. Mote, B. Walker, S. Cooper, T. Pavlik, and Z. Popović, "Verification games: Making verification fun," in *Proceedings of the 14th Workshop on Formal Techniques for Java-like Programs*, 2012, pp. 42–49.

[174] W. Li, S. Seshia, and S. Jha, "CrowdMine: Towards crowdsourced human-assisted verification," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 2–3.

[175] D. F. Bacon, Y. Chen, D. Parkes, and M. Rao, "A market-based approach to software evolution," in *Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, 2009, pp. 973–980.

[176] M. Nebeling and M. C. Norrie, "Context-aware and adaptive web interfaces : A crowdsourcing approach," in *Proceedings of the 11th International Conference on Web Engineering*, 2011, pp. 167–170.

[177] ——, "Tools and architectural support for crowdsourced adaptation of web interfaces," in *Proceedings of the 11th International Conference on Web Engineering*, 2011, pp. 243–257.

[178] W. Maalej and D. Pagano, "On the socialness of software," in *Proceedings of the 9th International Conference on Dependable, Autonomic and Secure Computing*, Dec 2011, pp. 864–871.

[179] R. Ali, C. Solis, I. Omoronyia, M. Salehie, and B. Nuseibeh, "Social adaptation: When software gives users a voice," in *Proceedings of the 7th International Conference Evaluation of Novel Approaches to Software Engineering*, June 2012.

[180] P. Akiki, A. Bandara, and Y. Yu, "Crowdsourcing user interface adaptations for minimizing the bloat in enterprise applications," in *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems*, 2013, pp. 121–126.

[181] C. Challiol, S. Firmenich, G. A. Bosetti, S. E. Gordillo, and G. Rossi, "Crowdsourcing mobile web applications," in *Proceedings of the ICWE 2013 Workshops*, 2013, pp. 223–237.

[182] M. Nebeling, M. Speicher, and M. C. Norrie, "CrowdAdapt: Enabling crowdsourced web page adaptation for individual viewing conditions and preferences," in *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 2013, pp. 23–32.

[183] H. He, Z. Ma, H. Chen, and W. Shao, "How the crowd impacts commercial applications: A user-oriented approach," in *Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies*, 2014, pp. 1–6.

[184] M. Almaliki, C. Ncube, and R. Ali, "The design of adaptive acquisition of users feedback: An empirical study," in *Proceedings of the 9th International Conference on Research Challenges in Information Science*, 2014.

[185] S. Hamidi, P. Andritsos, and S. Liaskos, "Constructing adaptive configuration dialogs using crowd data," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, 2014, pp. 485–490.

[186] C. Chen and K. Zhang, "Who asked what: Integrating crowdsourced FAQs into API documentation," in *Proceedings of the 36th International Conference on Software Engineering (ICSE Companion)*, 2014, pp. 456–459.

[187] A. Pawlik, J. Segal, M. Petre, and H. Sharp, "Crowdsourcing scientific software documentation: a case study of the NumPy documentation project," *Computing in Science and Engineering*, 2014.

[188] C. Exton, A. Wasala, J. Buckley, and R. Schäler, "Micro crowdsourcing: A new model for software localisation," *Localisation Focus*, vol. 8, no. 1, 2009.

[189] C. Arellano, O. Díaz, and J. Iturrioz, "Crowdsourced web augmentation : A security model," in *Proceedings of the 11 International Conference on Web Information Systems Engineering*, 2010, pp. 294–307.

[190] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-based malware detection system for Android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, 2011, pp. 15–26.

[191] P. K. Chilana, A. J. Ko, and J. O. Wobbrock, "LemonAid: Selection-based crowdsourced contextual help for web applications," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '12, 2012, pp. 1549–1558.

[192] P. K. Chilana, A. J. Ko, J. O. Wobbrock, and T. Grossman, "A multi-site field study of crowdsourced contextual help : Usage and perspectives of end users and software teams," in *Proceedings of the 31st Annual CHI Conference on Human Factors in Computing Systems*, 2013.

[193] 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 *Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, Jan. 2008, pp. 417–417.

[194] H. Krcmar, U. Bretschneider, M. Huber, and J. M. Leimeister, "Leveraging crowdsourcing: Activation-supporting components for IT-based ideas competition," *Journal of Management Information Systems*, vol. 26, no. 1, pp. 197–224, 2009.

[195] R. Jayakanthan and D. Sundararajan, "Enterprise crowdsourcing solution for software development in an outsourcing organization," in *Proceedings of the 11th International Conference on Web Engineering*, 2011, pp. 177–180.

[196] ——, "Enterprise crowdsourcing solutions for software development and ideation," in *Proceedings of the 2nd international workshop on Ubiquitous crowdsouring*, 2011, pp. 25–28.

[197] G. Standish, "The chaos report," http://www.standishgroup.com/sample_research_files/chaos_report_1994.pdf, Accessed: 2015-01-27.

[198] Y.-C. Huang, C.-I. Wang, and J. Hsu, "Leveraging the crowd for creating wireframe-based exploration of mobile design pattern gallery," in *Proceedings of the companion publication of the 2013 international conference on Intelligent user interfaces companion*, 2013, pp. 17–20.

[199] D. Mujumdar, M. Kallenbach, B. Liu, and B. Hartmann, "Crowdsourcing suggestions to programming problems for dynamic web development languages," in *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems*, 2011, pp. 1525–1530.

[200] W. B. Langdon and M. Harman, "Optimising existing software with genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 1, pp. 118–135, Feb 2015.

[201] J. Petke, M. Harman, W. B. Langdon, and W. Weimer, "Using genetic improvement & code transplants to specialise a C++ program to a problem class," in *Proceedings of the 17th European Conference on Genetic Programming*, April 2014, pp. 132–143.

[202] M. Orlov and M. Sipper, "Flight of the FINCH through the java wilderness," *IEEE Transactions Evolutionary Computation*, vol. 15, no. 2, pp. 166–182, 2011.

[203] D. R. White, A. Arcuri, and J. A. Clark, "Evolutionary improvement of programs," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 4, pp. 515–538, 2011.

[204] C. Le Goues, T. Nguyen, S. Forrest, and W. Weimer, "GenProg: A generic method for automatic software repair," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 54–72, 2012.

[205] W. B. Langdon, B. Lam, J. Petke, and M. Harman, "Improving cuda dna analysis software with genetic programming," in *Proceedings of the 2015 Genetic and evolutionary computation conference*, July 2015.

[206] B. Bruce, J. Petke, and M. Harman, "Reducing energy consumption using genetic improvement," in *Proceedings of the 2015 Genetic and evolutionary computation conference*, July 2015.

[207] I. Manotas, L. Pollock, and J. Clause, "SEEDS: A software engineer's energy-optimization decision support framework," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 503–514.

[208] D. Li, A. H. Tran, and W. G. J. Halfond, "Making web applications more energy efficient for OLED smartphones," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 527–538.

[209] F. Wu, M. Harman, Y. Jia, J. Krinke, and W. Weimer, "Deep parameter optimisation," in *Proceedings of the Genetic and evolutionary computation conference*, July 2015.

[210] C. Le Goues, S. Forrest, and W. Weimer, "Current challenges in automatic software repair," *Software Quality Journal*, vol. 21, no. 3, pp. 421–443, 2013.

[211] M. Harman, W. B. Langdon, and Y. Jia, "Babel pidgin: SBSE can grow and graft entirely new functionality into a real world system," in *Proceedings of the 6th Symposium on Search Based Software Engineering*, August 2014, pp. 247–252.

[212] H. D. T. Nguyen, D. Qi, A. Roychoudhury, and S. Chandra, "SemFix: program repair via semantic analysis," in *Proceedings of the 35th International Conference on Software Engineering*, B. H. C. Cheng and K. Pohl, Eds., May 18-26 2013, pp. 772–781.

[213] "Bing code search," http://codesnippet.research.microsoft.com, Accessed: 2015-05-06.

[214] S. Amann, S. Proksch, and M. Mezini, "Method-call recommendations from implicit developer feedback," in *Proceedings of the 1st International Workshop on CrowdSourcing in Software Engineering*, June 2014, pp. 5–6.

[215] S. Afshan, P. McMinn, and M. Stevenson, "Evolving readable string test inputs using a natural language model to reduce human oracle cost," *Proceedings of the 6th IEEE International Conference on Software Testing, Verification and Validation*, vol. 0, pp. 352–361, 2013.

[216] M. Bozkurt and M. Harman, "Automatically generating realistic test input from web services," in *Proceedings of the 6th IEEE International Symposium on Service Oriented System Engineering*, December.

[217] T. D. LaToza, W. B. Towne, C. M. Adriano, and A. van der Hoek, "Microtask programming: Building software with a crowd," in *Proceedings of the 27th annual ACM symposium on User interface software and technology*, 2014, pp. 43–54.

[218] N. Tillmann, M. Moskal, J. de Halleux, and M. Fahndrich, "TouchDevelop: Programming cloud-connected mobile devices via touchscreen," in *Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, 2011, pp. 49–60.

[219] J. Bishop, R. N. Horspool, T. Xie, N. Tillmann, and J. de Halleux, "Code Hunt: Experience with coding contests at scale," *Proceedings of the 37th International Conference on Software Engineering - JSEET*, June 2015.

[220] A. Bernstein, M. Klein, and T. W. Malone, "Programming the global brain," *Communications of the ACM*, vol. 55, no. 5, pp. 41–43, May 2012.

[221] R. Auler, E. Borin, and P. D. Halleux, "Addressing JavaScript JIT engines performance quirks : A crowdsourced adaptive compiler," in *Proceedings of the 23rd International Conference on Compiler Construction*, 2014, pp. 218–237.

[222] "Microsoft Lync," http://office.microsoft.com/lync, Accessed: 2015-01-11.

[223] "Chrome Telemetry," http://www.chromium.org/developers/telemetry, Accessed: 2015-04-30.

[224] "Firefox Telemetry," https://telemetry.mozilla.org, Accessed: 2015-04-30.

[225] D. Akhawe and A. P. Felt, "Alice in warningland: A large-scale field study of browser security warning effectiveness," in *Proceedings of the 22Nd USENIX Conference on Security*, 2013, pp. 257–272.

[226] A. Memon, I. Banerjee, and A. Nagarajan, "GUI ripping: Reverse engineering of graphical user interfaces for testing," in *Proceedings of the 10th Working Conference on Reverse Engineering*, Nov 2003, pp. 260–269.

[227] K. Lakhotia, P. McMinn, and M. Harman, "Automated test data generation for coverage: Haven't we solved this problem yet?" in *Proceedings of the 4th Testing Academia and Industry Conference — Practice And Research Techniques*, 4th–6th September 2009, pp. 95–104.

[228] K. Sen and G. Agha, "CUTE and jCUTE: Concolic unit testing and explicit path model-checking tools," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, T. Ball and R. Jones, Eds., 2006, vol. 4144, pp. 419–423.

[229] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball, "Feedback-Directed Random Test Generation," *Proceedings of the 29th International Conference on Software Engineering*, pp. 75–84, May 2007.

[230] N. Tillmann and J. de Halleux, "Pex-White Box Test Generation for .NET," in *Proceedings of the 2nd International Conference on Tests and Proofs*, ser. Lecture Notes in Computer Science, B. Beckert and R. Hhnle, Eds., 2008, vol. 4966, pp. 134–153.

[231] R. Pham, L. Singer, O. Liskin, F. Figueira Filho, and K. Schneider, "Creating a shared understanding of testing culture on a social coding site," in *Proceedings of the 2013 International Conference on Software Engineering*, 2013, pp. 112–121.

[232] E. J. Weyuker, "On testing non-testable programs," *The Computer Journal*, vol. 25, no. 4, pp. 465–470, Nov. 1982.

[233] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, 2015, to appear.

[234] D. Peters and D. Parnas, "Using test oracles generated from program documentation," *IEEE Transactions on Software Engineering*, vol. 24, no. 3, pp. 161–173, Mar 1998.

[235] T. Xie, "Cooperative testing and analysis: Human-tool, tool-tool, and human-human cooperations to get work done," in *Proceedings of the 12th IEEE International Working Conference on Source Code Analysis and Manipulation (Keynote)*, 2012.

[236] M. D. Ernst and Z. Popović, "Crowd-sourced program verification," University OF Washington, Tech. Rep., 2012.

[237] R. Watro, K. Moffitt, T. Hussain, D. Wyschogrod, J. Ostwald, D. Kong, C. Bowers, E. Church, J. Guttman, and Q. Wang, "Ghost Map: Proving software correctness using games," in *The 8th International Conference on Emerging Security Information, Systems and Technologies*, 2014.

[238] C. Cook and M. Visconti, "Documentation is important," *CrossTalk*, vol. 7, no. 11, pp. 26–30, 1994.

[239] M. Visconti and C. Cook, "An overview of industrial software documentation practice," in *Proceedings. 22nd International Conference of the Chilean Computer Science Society*, 2002, pp. 179–186.

[240] M. Kajko-Mattsson, "A survey of documentation practice within corrective maintenance," *Empirical Software Engineering*, vol. 10, no. 1, pp. 31–55, 2005.

[241] B. Esselink, *A practical guide to localization*.   John Benjamins Publishing, 2000, vol. 4.

[242] Z. P. Fry and W. Weimer, "A human study of fault localization accuracy," in *Proceedings of the 26th IEEE International Conference on Software Maintenance*, 2010.

[243] Z. P. Fry, B. Landau, and W. Weimer, "A human study of patch maintainability," in *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, 2012, pp. 177–187.

[244] J. Cleland-Huang, Y. Shin, E. Keenan, A. Czauderna, G. Leach, E. Moritz, M. Gethers, D. Poshyvanyk, J. H. Hayes, and W. Li, "Toward actionable, broadly accessible contests in software engineering," in *Proceedings of the 34th International Conference on Software Engineering*, June 2012, pp. 1329–1332.

[245] M. Harman and B. F. Jones, "Search-based software engineering," *Information and Software Technology*, vol. 43, no. 14, pp. 833–839, 2001.

[246] Y.-H. Tung and S.-S. Tseng, "A novel approach to collaborative testing in a crowdsourcing environment," *Journal of Systems and Software*, vol. 86, no. 8, pp. 2143–2153, Aug. 2013.

[247] K. Stolee and S. Elbaum, "Identification, impact, and refactoring of smells in pipe-like web mashups," *IEEE Transactions on Software Engineering*, vol. 39, no. 12, pp. 1654–1679, Dec 2013.

[248] K. T. Stolee, S. Elbaum, and D. Dobos, "Solving the search for source code," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 3, pp. 26:1–26:45, June 2014.

[249] T. D. LaToza and A. van der Hoek, "A vision of crowd development," in *Proceedings of the 37th International Conference on Software Engineering, NIER Track*, 2015, to appear.

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

[251] ——, "The metropolis model and its implications for the engineering of software ecosystems," in *Proceedings of the 2010 FSE/SDP workshop on Future of software engineering research*, 2010, pp. 187–190.

[252] K. Mao, Y. Yang, M. Li, and M. Harman, "Pricing Crowdsourcing Based Software Development Tasks," in *Proceedings of the 2013 International Conference on Software Engineering (NIER Track)*, 2013, pp. 1205–1208.

[253] G. D. Saxton, O. Oh, and R. Kishore, "Rules of Crowdsourcing: Models, Issues, and Systems of Control," *Information Systems Management*, vol. 30, no. 1, pp. 2–20, Jan. 2013.

[254] W.-T. Tsai, W. Wu, and M. N. Huhns, "Cloud-based software crowdsourcing," *IEEE Internet Computing*, vol. 18, no. 3, pp. 78–83, May 2014.

[255] Z. Hu and W. Wu, "A game theoretic model of software crowdsourcing," in *Proceedings of the 8th IEEE International Symposium on Service Oriented System Engineering*, Apr. 2014, pp. 446–453.

[256] X. L. Xu and Y. Wang, "On the Process Modeling of Software Crowdsourcing Based on Competitive Relation," *Advanced Materials Research*, vol. 989-994, pp. 4708–4712, July 2014.

[257] T. D. LaToza, W. B. Towne, and A. V. D. Hoek, "Harnessing the crowd : Decontextualizing software work," in *Proceedings of the 1st International Workshop on Context in Software Development Workshop*, 2014, pp. 2–3.

[258] L. Tran-Thanh, S. Stein, A. Rogers, and N. R. Jennings, "Efficient crowdsourcing of unknown experts using bounded multi-armed bandits," *Artificial Intelligence*, vol. 214, pp. 89–111, Sept. 2014.

[259] K. Mao, Y. Yang, Q. Wang, Y. Jia, and M. Harman, "Developer recommendation for crowdsourced software development tasks," in *Proceedings of the 9th IEEE International Symposium on Service-Oriented System Engineering*, 2015, pp. 347–356.

[260] M. V. Mäntylä and J. Itkonen, "More testers - the effect of crowd size and time restriction in software testing," *Information and Software Technology*, vol. 55, no. 6, pp. 986–1003, June 2013.

[261] H. Sharp, N. Baddoo, S. Beecham, T. Hall, and H. Robinson, "Models of motivation in software engineering," *Information and Software Technology*, vol. 51, no. 1, pp. 219–233, 2009.

[262] S. Beecham, N. Baddoo, T. Hall, H. Robinson, and H. Sharp, "Motivation in software engineering: A systematic literature review," *Information and Software Technology*, vol. 50, no. 9, pp. 860–878, 2008.

[263] B. W. Boehm *et al.*, *Software engineering economics*.   Prentice-hall Englewood Cliffs (NJ), 1981, vol. 197.

[264] L. R. Varshney, "Participation in crowd systems," in *Proceedings of the 50th Annual Allerton Conference on Communication, Control, and Computing*, Oct. 2012, pp. 996–1001.

[265] D. L. Olson and K. Rosacker, "Crowdsourcing and open source software participation," *Service Business*, vol. 7, no. 4, pp. 499–511, Nov. 2012.

[266] S. Ramakrishnan and V. Srinivasaraghavan, "Delivering software projects using captive university crowd," in *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, 2014, pp. 115–118.

[267] P. G. Ipeirotis, F. Provost, and J. Wang, "Quality management on amazon mechanical turk," in *Proceedings of the 2010 ACM SIGKDD Workshop on Human Computation*, 2010, pp. 64–67.

[268] M.-C. Yuen, I. King, and K.-S. Leung, "A survey of crowdsourcing systems," in *Proceedings of the 2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third Inernational Conference on Social Computing*, Oct 2011, pp. 766–773.

[269] M. Allahbakhsh, B. Benatallah, A. Ignjatovic, H. Motahari-Nezhad, E. Bertino, and S. Dustdar, "Quality control in crowdsourcing systems: Issues and directions," *IEEE Internet Computing*, vol. 17, no. 2, pp. 76–81, March 2013.

[270] S. M. Wolfson and M. Lease, "Look before you leap: legal pitfalls of crowdsourcing," *Proceedings of the American Society for Information Science and Technology*, vol. 48, no. 1, pp. 1–10, 2011.

[271] L. P. Cox, "Truth in crowdsourcing," *IEEE Journal on Security and Privacy*, vol. 9, no. 5, pp. 74–76, 2011.

[272] M. Vukovic, J. Laredo, and S. Rajagopal, "Challenges and experiences in deploying enterprise," in *Proceedings of the 10th International Conference on Web Engineering*, 2010.

[273] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Eds., *Software Engineering for Self-Adaptive Systems (Dagstuhl Seminar)*, ser. Dagstuhl Seminar Proceedings, vol. 08031, 2008.

[274] M. Harman, Y. Jia, W. B. Langdon, J. Petke, I. H. Moghadam, S. Yoo, and F. Wu, "Genetic improvement for adaptive software engineering (keynote)," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2014, pp. 1–4.

[275] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf, "An architecture-based approach to self-adaptive software," *IEEE Intelligent Systems*, vol. 14, pp. 54–62, May 1999.

[276] M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: MSR for App Stores," in *Proceedings of the 9th Working Conference on Mining Software Repositories*, June 2012, pp. 108–111.

[277] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study," in *Proceedings of the 21st IEEE International Conference on Requirements Engineering*, July 2013, pp. 125–134.

[278] E. Guzman and W. Maalej, "How do users like this feature? A fine grained sentiment analysis of app reviews," in *Proceedings of the 22nd International Conference on Requirements Engineering*, Aug 2014, pp. 153–162.

[279] N. Chen, J. Lin, S. C. Hoi, X. Xiao, and B. Zhang, "AR-Miner: Mining informative reviews for developers from mobile app marketplace," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 767–778.

[280] K. Adamopoulos, M. Harman, and R. M. Hierons, "Mutation testing using genetic algorithms: A co-evolution approach," in *Proceedings of the 2004 Genetic and Evolutionary Computation Conference*, June 2004, pp. 1338–1349.

[281] A. Arcuri, D. R. White, J. A. Clark, and X. Yao, "Multi-objective improvement of software using co-evolution and smart seeding," in *Proceedings of the 7th International Conference on Simulated Evolution and Learning*, vol. 5361, December 2008, pp. 61–70.