A Survey of the Use of Crowdsourcing in Software Engineering

Ke Mao¹, Licia Capra, Mark Harman, Yue Jia

Department of Computer Science, University College London, Malet Place, London, WC1E 6BT, UK

Abstract

The term ‘crowdsourcing’ was initially introduced in 2006 to describe an emerging distributed problem-solving model by online workers. Since then it has been widely studied and practiced to support software engineering. In this paper we provide a comprehensive survey of the use of crowdsourcing in software engineering, seeking to cover all literature on this topic. We first review the definitions of crowdsourcing and derive our definition of Crowdsourcing Software Engineering together with its taxonomy. Then we summarise industrial crowdsourcing practice in software engineering and corresponding case studies. We further analyse the software engineering domains, tasks and applications for crowdsourcing and the platforms and stakeholders involved in realising Crowdsourced Software Engineering solutions. We conclude by exposing trends, open issues and opportunities for future research on Crowdsourced Software Engineering.

Keywords: Crowdsourced software engineering, software crowdsourcing, crowdsourcing, literature survey

1. Introduction

Crowdsourcing is an emerging distributed problem-solving model based on the combination of human and machine computation. The term ‘crowdsourcing’ was jointly coined by Howe and Robinson in 2006 (Howe, 2006b). According to the widely accepted definition presented in the 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 (CSE) derives from crowdsourcing. Using an open call, it recruits global online labour to work on various types of software engineering tasks, such as requirements extraction, design, coding and testing. This emerging model has been claimed to reduce time-to-market by increasing parallelism (Lakhani et al., 2010; LaToza et al., 2013a; Stol and Fitzgerald, 2014c), and to lower costs and defect rates with flexible development capability (Lakhani et al., 2010). Crowdsourced Software Engineering is implemented by many successful crowdsourcing platforms, such as TopCoder, AppStori, uTest, Mob4Hire and TestFlight.

The crowdsourcing model has been applied to a wide range of creative and design-based activities (Cooper et al., 2010; Norman et al., 2011; Brabham et al., 2009; Chatfield and Brajwida, 2014; Alonso et al., 2008). Crowdsourced Software Engineering has also rapidly gained increasing interest in both industrial and academic communities. Our pilot study of this survey reveals a dramatic rise in recent work on the use of crowdsourcing in software engineering, yet many authors claim that there is ‘little work’ on crowdsourcing for/in software engineering (Schiller and Ernst, 2012; Schiller, 2014; Zogaj et al., 2014). These authors can easily be forgiven for this misconception, since the field is growing quickly and touches many disparate aspects of software engineering, forming a literature that spreads over many different software engineering application areas. Although previous work demonstrates that crowdsourcing is a promising approach, it usually targets a specific activity/domain in software engineering. Little is yet known about the overall picture of what types of tasks have been applied in software engineering, which types are more suitable to be crowdsourced, and what the limitations of and issues for Crowdsourced Software Engineering are. This motivates the need for the comprehensive survey that we present here.

The purpose of our survey is two-fold: First, to provide a comprehensive survey of the current research progress on using crowdsourcing to support software engineering activities. Second, to summarise the challenges for Crowdsourced Software Engineering and to reveal to what extent these challenges were addressed by existing work. Since this field is an emerging, fast-expanding area in software engineering yet to achieve full maturity, we aim to strive for breadth in this survey. The included literature may directly crowdsourse software engineering tasks to the general public, indirectly reuse existing crowdsourced knowledge, or propose a framework to enable the realisation/improvement of Crowdsourced Software Engineering.

The remaining parts of this paper are organised as follows. Section 2 introduces the methodology on literature search and selection, with detailed numbers for each step. Section 3 presents background information on Crowdsourced Software Engineering. Section 4 describes practical platforms for Crowdsourced Software Engineering, together with their typ-
ical processes and relevant case studies. Section 5 provides a finer-grained view of Crowdsourced Software Engineering based on their application domains in software development life-cycle. Sections 6 and 7 describe current issues, open problems and opportunities. Section 8 discusses the limitations of this survey. Section 9 concludes.

2. Literature Search and Selection

The aim of conducting a comprehensive survey of all publications related to Crowdsourced Software Engineering necessitates a careful and thorough paper selection process. The process contains several steps which are described as follows:

To start with, we defined the inclusion criteria of the surveyed publications: The main criterion for including a paper in our survey is that the paper should describe research on crowdsourcing that addresses at least one activity (directly or indirectly) involved in software engineering. A ‘software engineering activity’ can be any activity in the development, operation and maintenance of software, according to the IEEE Computer Society definition of software engineering (Abran et al., 2004). Also, the literature must be presented in English as conference papers, journal papers, theses, technical reports or books.

We performed three types of searches on related publications 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 terms and their combinations we used are presented in Table 1. We searched each of the term combinations using exact match queries (e.g., “software crowdsourcing”, “crowdsourced software development”, “crowd testing”, etc.) in both the meta-data and full-text (when available) of the publications.

- Issue-by-issue search of major 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. Our searched conference proceedings and journals are listed in Table 2.

- Reference search for identifying missed publications by going through citations from included ones (snowballing).

We conducted a screening process to filter the collected literature by removing any that did not meet our inclusion criteria. We read the title and abstract (and the full text when necessary) of each publication carefully, applied the inclusion criteria

\[ \text{Note that since the crowdsourcing concept itself is expanding, its definition is still debated in the literature. Therefore, in order to ensure that our survey remains comprehensive, our inclusion criteria cover not only studies that meet our definition, but also those in which the authors claim to use crowdsourcing.} \]

\[ \text{The screening process is iterative, e.g., we also screened the publications suggested by the authors contacted and the anonymous reviewers.} \]
Table 1: Terms for Online Library Search

<table>
<thead>
<tr>
<th>Category</th>
<th>Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>(software crowdsourcing) (crowd OR crowdsourcing OR crowdsourced) AND (software engineering) (crowd OR crowdsourcing OR crowdsourced) AND (software development)</td>
</tr>
<tr>
<td>Domain</td>
<td>(crowd OR crowdsourcing OR crowdsourced) AND (software requirements) (crowd OR crowdsourcing OR crowdsourced) AND (software design)</td>
</tr>
<tr>
<td></td>
<td>(crowd OR crowdsourcing OR crowdsourced) AND (software coding) (crowd OR crowdsourcing OR crowdsourced) AND (software testing)</td>
</tr>
<tr>
<td></td>
<td>(crowd OR crowdsourcing OR crowdsourced) AND (software verification) (crowd OR crowdsourcing OR crowdsourced) AND (software evolution)</td>
</tr>
<tr>
<td></td>
<td>(crowd OR crowdsourcing OR crowdsourced) AND (software maintenance)</td>
</tr>
</tbody>
</table>

Table 2: Selected Conference Proceedings and Journals for Manual Search

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICSE</td>
<td>International Conference on Software Engineering</td>
</tr>
<tr>
<td>ESEC/FSE</td>
<td>European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering</td>
</tr>
<tr>
<td>OOPSLA</td>
<td>Conference on Object-Oriented Programming Systems, Languages, and Applications</td>
</tr>
<tr>
<td>ASE</td>
<td>International Conference on Automated Software Engineering</td>
</tr>
<tr>
<td>ISSTA</td>
<td>International Symposium on Software Testing and Analysis</td>
</tr>
<tr>
<td>ICST</td>
<td>International Conference on Software Testing, Verification and Validation</td>
</tr>
<tr>
<td>RE</td>
<td>International Requirements Engineering Conference</td>
</tr>
<tr>
<td>CSCW</td>
<td>Conference on Computer-Supported Cooperative Work and Social Computing</td>
</tr>
<tr>
<td>ISSC</td>
<td>International Symposium on Software Crowdsourcing</td>
</tr>
<tr>
<td>CSI-SE</td>
<td>International Workshop on Crowdsourcing in Software Engineering</td>
</tr>
<tr>
<td>TSE</td>
<td>Transactions on Software Engineering</td>
</tr>
<tr>
<td>TOSEM</td>
<td>Transactions on Software Engineering Methodology</td>
</tr>
<tr>
<td>IEEE SW</td>
<td>IEEE Software</td>
</tr>
<tr>
<td>IET</td>
<td>IET Software</td>
</tr>
<tr>
<td>IST</td>
<td>Information and Software Technology</td>
</tr>
<tr>
<td>JSS</td>
<td>Journal of Systems and Software</td>
</tr>
<tr>
<td>SQJ</td>
<td>Software Quality Journal</td>
</tr>
<tr>
<td>SPE</td>
<td>Software: Practice and Experience</td>
</tr>
</tbody>
</table>

We first review definitions of crowdsourcing, before proceeding to the focus of Crowdsourced Software Engineering.

3. Definitions, Trends and Landscape

The term ‘Crowdsourcing’ was first widely accepted in 2006. Jeff Howe used the term in his article ‘The Rise of Crowdsourcing’, which was published in Wired (Howe, 2006b). In a companion blog post (Howe, 2006a) 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 of crowdsourcing. Howe argued that crowdsourced work can be done by cooperation or by sole individuals (Howe, 2006b).

This idea echoes the earlier book ‘The Wisdom of the Crowds’ (James, 2004) and also finds some resonance in the principles of Open Source Software (OSS) development (Kogut and Metiu, 2001). 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 (Sobel, 1995).

Turning to online crowdsourcing, early Internet-based crowdsourcing activities can be found in 2001, when ‘InnoCen-
tive’ 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 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 was proposed (Hughes, 2010). 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 (Brabham, 2008), crowdsourcing is viewed as an online model for distributed production and problem-solving. The Wikipedia page on crowdsourcing cites the definition which appeared in the Merriam-Webster dictionary in 2011. It stresses the large group of workers and an online community, but drops any mention of ‘undefined labour’ and ‘open call’ format. Estellés-Arolas et al. (2012) collected 40 definitions from 32 articles published during 2006 to 2011, and proposed an integrated definition, which is compatible with the ones we have introduced and further specifies the mutual benefits between workers and 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 finely grained as photo tagging or as coarsely grained as software development (Kittur et al., 2011; Xiao and Paik, 2014). A list of more than 160 crowdsourcing projects has been compiled (using crowdsourcing to compile the list).

Crowdsourcing has been used extensively in various disciplines, such as protein structure prediction (Cooper et al., 2010; Khatib et al., 2011), drug discovery (Norman et al., 2011; Johnson, 2014), transportation planning (Brabham et al., 2009; Misra et al., 2014), weather forecasting (Chatfield and Brajavidaga, 2014; Muller et al., 2015), information retrieval (Alonso et al., 2008; Lease and Yilmaz, 2012), and software engineering (Stol and Fitzgerald, 2014c; Breaux and Schaub, 2014; Schiller and Ernst, 2012; Cochran et al., 2015; Storey et al., 2010; Stoee and Elbaum, 2010), to which we now turn.

3.2. Crowdsourced Software Engineering

We use the term ‘Crowdsourced Software Engineering’ to denote the application of crowdsourcing techniques to support software development (in its broadest sense). Some authors refer to this as ‘Crowdsourced Software Development’, ‘Crowdsourcing Software Development’ and ‘Software Crowdsourcing’ in previous studies (Usui and Morikasi, 2011; Wu et al., 2013a; Tajedin and Nevo, 2013; Stol and Fitzgerald, 2014c; Xu and Wang, 2014a; Tajedin and Nevo, 2014; Prikladnicki et al., 2014; Li et al., 2015). However, 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. Example activities include project planning, requirements elicitation, security augmentation and test case refinement.

However, although our definition is inclusive of all software engineering activities, we wish to distinguish Crowdsourced Software Engineering from the research activities on software engineering that happen to be supported by Crowdsourcing (see Fig. 4). 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

---

5http://www.innocentive.com
6http://en.wikipedia.org/wiki/crowdsourcing
7http://www.merriam-webster.com/info/newwords11.htm
8http://www.merriam-webster.com/dictionary/crowdsourcing
9http://en.wikipedia.org/wiki/list of crowdsourcing projects
<table>
<thead>
<tr>
<th>Year</th>
<th>Author</th>
<th>Degree</th>
<th>University</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>Lim (2010)</td>
<td>PhD</td>
<td>University of New South Wales</td>
<td>Social Networks and Collaborative Filtering for Large-Scale Requirements Elicitation</td>
</tr>
<tr>
<td>2011</td>
<td>Leone (2011)</td>
<td>PhD</td>
<td>ETH Zurich - Swiss Federal Institute of Technology</td>
<td>Information Components as a Basis for Crowdsourced Information System Development</td>
</tr>
<tr>
<td>2012</td>
<td>Ponzanelli (2012)</td>
<td>Master</td>
<td>University of Lugano</td>
<td>Exploiting Crowd Knowledge in the IDE</td>
</tr>
<tr>
<td>2013</td>
<td>Chilana (2013)</td>
<td>PhD</td>
<td>University of Washington</td>
<td>Supporting Users After Software Deployment through Selection-Based Crowdsourced Contextual Help</td>
</tr>
<tr>
<td>2013</td>
<td>Wightman (2013)</td>
<td>PhD</td>
<td>Queen’s University</td>
<td>Search Interfaces for Integrating Crowdsourced Code Snippets within Development Environments</td>
</tr>
<tr>
<td>2013</td>
<td>Xue (2013)</td>
<td>PhD</td>
<td>University of Illinois at Urbana-Champaign</td>
<td>Using Redundancy to Improve Security and Testing</td>
</tr>
<tr>
<td>2014</td>
<td>Schiller (2014)</td>
<td>PhD</td>
<td>University of Washington</td>
<td>Reducing the Usability Barrier to Specification and Verification</td>
</tr>
</tbody>
</table>
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 survey this topic for completeness. In addition, as shown in Fig. 4, software engineering techniques can also be used to support the implementation of generic crowdsourcing, e.g., building a novel crowdsourcing system to support the crowdsourcing process (Aparicio et al., 2012). This type of study is out of the scope of 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 4), 69% of our surveyed papers use (or echo) the concept of crowdsourcing without citing any definition. For those papers that cite one or more definitions, the most widely cited is Howe’s definition (18%). Out of all the 210 publications we reviewed, only two give an explicit definition of what it means for crowdsourcing to be applied specifically to software engineering activities (Stol and Fitzgerald, 2014c; Huhns et al., 2013). One single paper may cite multiple definitions.

<table>
<thead>
<tr>
<th>Def.</th>
<th>None</th>
<th>Howe</th>
<th>Wiki</th>
<th>Own</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>144 (69%)</td>
<td>37 (18%)</td>
<td>4 (2%)</td>
<td>7 (3%)</td>
<td>22 (10%)</td>
</tr>
</tbody>
</table>

Figure 4: Crowdsourcing and software engineering.

Table 4: Cited Crowdsourcing Definitions

Since Howe’s definition is the most widely accepted crowdsourcing definition in the papers we surveyed, we choose to define Crowdsourced Software Engineering (CSE) 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.

Note that in this survey, we also include the work that uses crowd knowledge to support software engineering activities, which can be regarded as a form of indirect use of crowdsourcing. For example, those ‘crowd debugging’ (Chen and Kim, 2015; Mujumdar et al., 2011; Hartmann et al., 2010) studies on collecting and mining crowdsourced knowledge. We observed that the software engineering community generally considers this as an instance of using crowdsourcing to support software engineering, because the knowledge used was gathered from ‘crowd workers’, as defined in Howe’s definition.

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. 5 briefly depicts these three types of actors and the general process for Crowdsourced Software Engineering. More detailed background knowledge on the Crowdsourced Software Engineering actors, principles, process and frameworks can be found in the recent book by Li et al. (Li et al., 2015).

3.2.1. Claimed Advantages and Growth Trends

Crowdsourced Software Engineering has several potential advantages 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 was claimed to exhibit the ability to deliver customer requested software assets with a lower defect rate at lower cost in less time (Lakhani et al., 2010). TopCoder claimed that their crowdsourced development was capable of reducing cost by 30%-80% when compared with in-house development or outsourcing (Lydon, 2012). Furthermore, in the TopCoder American Online case study (Lakhani et al., 2010), 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 (Lakhani et al., 2013), 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 three orders 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 and academic institutions to large IT companies. DARPA created Crowdsourced Formal Verification (CSFV) program\(^\text{13}\) for software formal verification and launched the Verigames website to facilitate the practice\(^\text{14}\). NASA and Harvard business school established the NASA Tournament Laboratory for crowdsourcing software solutions for NASA systems\(^\text{15}\). Microsoft crowdsourced partial software development activities in Office 2010\(^\text{16}\), Windows 8.1\(^\text{17}\) and Windows 10\(^\text{18}\).

2) Workers: Based on an industrial report from Massolution (Massolution, 2012), the number of workers engaged in software crowdsourcing increased by 151% in the year of 2011.

3) Platforms: There is a growing number of crowdsourcing platforms built for software development domain, such as AppStori and Mob4Hire. These commercial platforms will be described in more detail in Section 4.

The flourishing Crowdsourced Software Engineering landscape is also revealed by the increasing number of relevant publications published in recent years, as shown in Fig. 2. Crowdsourced Software Engineering is also proving to be an attractive topic for student dissertations. Specifically, 20 out of the total 210 publications are Master/PhD theses. A detailed list of these theses can be found in Table 3.

3.2.2. Research Topics

To classify the papers, we first carefully analysed the 210 papers we collected, revealing four top-level categories based on their study types: Study of Practice, Theories and Models, Applications to Software Engineering and Evaluations of Software Engineering Research. We referred to the ACM Computing Classification System\(^\text{19}\), the IEEE Taxonomy of Software Engineering Standards\(^\text{20}\) and the 2014 IEEE Keywords Taxonomy\(^\text{21}\) 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. A detailed taxonomy of Crowdsourced Software Engineering research is given in Fig. 6.

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 average percentage agreement of 91.2%. The distribution of the literature over the research topics is shown in Fig. 7. The most prominent class is Applications to Software Engineering (64%), followed by theoretical studies (19%) and practical studies (14%). A few studies

\(^{13}\)http://www.darpa.mil/program/crowd-sourced-formal-verification
\(^{14}\)http://www.verigames.com
\(^{15}\)http://www.nasa.gov/sites/default/files/files/ntl-overview-sheet.pdf
\(^{16}\)http://www.wired.com/2010/06/microsoft-office-2010
\(^{17}\)http://www.forbes.com/sites/andygreenberg/2013/06/19/microsoft-finally-offers-to-pay-hackers-for-security-bugs-with-100000-bounty
\(^{18}\)http://thetechieguy.com/how-microsoft-is-cleverly-crowdsourcing-windows-10-development-from-its-customers
\(^{19}\)http://www.acm.org/about/class/class/2012
\(^{20}\)http://ieeexplore.ieee.org/servlet/opac?punumber=2601
model has been adapted in several ways, each phase may vary in and an implementation phase. Since Simon’s problem-solving (pects regarding a series of stages in realising CSE are sum-

\[\text{success, as shown in Fig.}\]

\[\text{scape from such a general perspective of problem-solving pro-

\[\text{which characterises the steps towards approaching and ratio-

\[\text{an SDLC phase, it requires a general problem-solving process}

\[\text{in Section}\]

\[\text{details are discussed in detail}

\[\text{in Section}\]

\[\text{When adopting crowdsourcing to support a certain activity in}

\[\text{an SDLC phase, it requires a general problem-solving process}

\[\text{which characterises the steps towards approaching and ratio-

\[\text{nally solving a problem. We illustrate the CSE research land-

\[\text{from such a general perspective of problem-solving pro-

\[\text{cess, as shown in Fig.}\]

\[\text{important questions and as-

\[\text{pects regarding a series of stages in realising CSE are sum-

\[\text{arised. The stages follow Simon’s problem-solving model}

\[\text{(Simon, 1960), which consists of two phases: a decision phase}

\[\text{and an implementation phase. Since Simon’s problem-solving}

\[\text{model has been adapted in several ways, each phase may vary in}

\[\text{stages. Our decision phase contains the three typical stages: int-

\text{elligence, design and choice. While the implementation phase includes an implementation stage and a following review stage.}

\[\text{Intelligence (why): In the intelligence stage, the problem is}

\[\text{defined and the requester should justify the motivation for adopting CSE. What are the potential advantages and disad-

\[\text{vantages? Previous research on CSE usually argues that the cost, efficiency and scalability are the benefits. Meanwhile, the potential issues on intellectual property and the quality of the crowd work may need to be considered.}

\[\text{Design (what): The design stage relates to the development of alternative solutions. This phase may require research into the potential options. When mapping to the CSE stage, the re-

\[\text{quester should think about, what is being crowdsourced? What is}

\[\text{the granularity of the task? Is it a micro-task such as a re-

\[\text{quirement classification or a macro task such as a software com-

\[\text{ponent development? Does the task require expert knowledge}

\[\text{such as programming? What are the incentives for a certain type of crowd? What is the expected size of the crowd?}

\[\text{Choice (which): In the choice stage, alternative solutions are}

\[\text{evaluated. The output of this stage is a decision that can be}

\[\text{implemented. To make such a decision usually requires addi-

\[\text{tional information, which has not been collected in the previous design stage. Other choices would also be considered, such as the quality assurance strategy and the type of open call to be adopted.}

\[\text{Implementation (how): The implementation stage is where}

\[\text{the decision is finally carried out. Both CSE researchers and}

\[\text{practitioners need to deal with a series of questions to imple-

\[\text{ment the crowdsourcing activity. For example, what intermedi-

\[\text{ary platform should be used in order to accomplish the work? Subsequently, how to manage the tasks and workers?}

\[\text{Review (outcome): The final review stage evaluates the im-

\[\text{plementation’s outcome. If the outcome is a success, the prod-

\[\text{uct may be further validated to check whether it satisfies users’}

\[\text{requirements. If failed, lessons should be learned and a new}

\[\text{iteration of the problem-solving process is expected. For CSE,}

\[\text{the requester may need to think about how to aggregate and}

\[\text{validate the crowd solutions. If the use of crowdsourcing in}

\[\text{supporting software engineering failed, what is the barrier to a}

\[\text{successful CSE solution?}

\[\text{The above process model may provide guidance to CSE re-

\[\text{searchers and practitioners for realising CSE. Several of the}

\[\text{questions underlying each stage remain open problems, point-

\[\text{ing to important research questions. Very few previous studies consid-

\[\text{er the overall process of CSE at a macro level (such as the one presented in Fig.}\]

\[\text{9). Instead, where the process studies were conducted, they usually focused on the problems per-

\[\text{tained to a specific stage.}

\[\text{In the why stage, many previous studies report on the bene-

\[\text{fits of CSE in terms of low cost, fast delivery and high flexibil-

\[\text{ity (Schneider and Cheung, 2011; Lakhani et al., 2010; Begel et al., 2013; LaToza et al., 2013a; Nebeling et al., 2013b; Ra-

\[\text{makrishnan and Srinivasaraghavan, 2014; Li et al., 2015},]

\[\text{but there is a lack of comparative studies on validating these ben-

\[\text{efits by comparing crowdsourcing-based development with tra-

\[\text{ditional in-house or outsourced development. A few other au-

\[\text{thors, see for example (Stol and Fitzgerald, 2014c,b; LaToza}

Figure 7: Research topic distribution.
and van der Hoek, 2015), highlight the key concerns associated with CSE. In the what stage, research topics of ‘what to crowdsource’, ‘who is the crowd’, and ‘what is the motivation’ have been briefly touched upon. Stol and Fitzgerald (2014c) point out that self-contained, less complex software tasks are more suitable to be crowdsourced, according to an in-depth study on TopCoder. Schiller and Ernst (2012), and Pastore et al. (2013) discuss using ad-hoc versus contract workers and professional versus generic workers, respectively. A detailed discussion of the motivation and composition of the crowd will be presented in Sections 6.4 and 7.1. In the following which, how and outcome stages, it is surprising to see that not much research work has been done so far. For ‘which CSE model’, a few process models have been proposed, which will be introduced in Section 6.1. For ‘which quality assurance strategy’, relevant studies will be addressed in Section 6.5. Regarding the how stage, the challenges of task decomposition and crowd coordination will be discussed in Sections 6.2 and 6.3.

In contrast to the lack of investigation of the CSE process, most studies reviewed in this survey integrate crowdsourcing into their approaches to support activities in the SDLC. In later sections, we first summarise these papers (Section 5) and then turn back to the CSE process, discussing the issues and open problems studied in previous work (Section 6), and opportunities for future research (Section 7).

4. 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 we collected were based 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.

4.1. Commercial Platforms

A list of existing commercial crowdsourcing platforms that support software engineering are presented in Table 5. These platforms employ various types of open call formats, such as the widely used online competition, on-demand matching (where 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, respectively. There are also general crowdsourcing marketplaces such as Amazon Mechanical Turk (AMT), 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 is a pioneer for practising Crowdsourced Software Engineering. It has its unique process and development model, which is known as the TopCoder Competition Methodology. The platform supports the independent graphic design, development, data science challenges, and 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 which is rated through a review process. The review board is also made up of crowd developers from the TopCoder community.

2) AppStori is a more recent platform for crowdsourcing mobile app development. 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 is one of the leading platforms for crowdsourced software testing. It claims to support the world’s largest open community for software testing22. The crowd testing community enables a wide range of virtual on-demand testing services, such as functional testing, usability testing, localisation testing and load testing. The crowdsourced testing process starts with a phase in which the clients can specify their testing needs. Flexible choices concerning testing devices, operating systems, geographic locations and budgets are provided by the platform. Appropriate testers are selected from the community, based on several metrics such as their previous performance, skills, languages, testing devices and locations. 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.

4) StackOverflow23 is a question and answer website which provides crowdsourced programming knowledge for software developers. Although such crowd knowledge is passively ‘pulled’ by developers with issues rather than being an active part of the development process, it poses a positive impact on open source software development (Vasilescu et al., 2013, 2014) as well as conventional software development process. It has been used to improve integrated software development environments (Zagalsky et al., 2012; Bacchelli et al., 2012; Ponzanelli, 2012; Ponzanelli et al., 2013a,b; de Souza et al., 2014) and software API documentation (Jiau and Yang, 2012; Parnin et al., 2012).

22http://www.utest.com/about-us
23http://www.stackoverflow.com
5) Bountify is a platform similar to StackOverflow. However, it has more ‘self-contained’, micro programming tasks. Each yields a payment of a certain amount of money, ranging from 1 to 100 US dollars. A study on program synthesis (Cochran et al., 2015) used this platform to obtain initial seeds for their genetic programming algorithm.

Other more general-purpose crowdsourcing platforms such as Amazon Mechanical Turk and CrowdFlower also have been widely used in software engineering research:

1) Amazon Mechanical Turk (AMT) is a popular crowdsourcing marketplace for micro-tasks. By employing crowd workers on the platform to exploit human computation, small teams may mitigate the challenges in developing complex software systems (Begel et al., 2010). This platform has been employed to support program synthesis (Cochran et al., 2015), graphical user interface (GUI) testing (Dolstra et al., 2013), oracle problem mitigation (Pastore et al., 2013), and program verification (Schiller and Ernst, 2012) in software engineering.

2) CrowdFlower is a micro-task crowdsourcing platform that is similar to Amazon Mechanical Turk. It focuses more on solving data problems such as data collection, cleaning and labelling. Afshan et al. (2013) employed this platform to evaluate the human readability of test string inputs, generated by a search-based test data generation technique with a natural language model.

Several studies provided further information on existing commercial platforms for software engineering. An introduction to software crowdsourcing platforms (Peng et al., 2014) 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. (2010) summarised three types of crowdsourcing platforms for the software industry: platforms such as Amazon Mechanical Turk24 that support the use of human knowledge in an inexpensive way; platforms such as TopCoder that support

24http://www.mturk.com
contest-based software development; and platforms like MathWorks25 that support programming competitions with a unique ‘competitive collaboration’ feature. Wu et al. (2013a) proposed an evaluation framework for assessing software crowdsourcing processes with respect to 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.

4.2. 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 (Archak, 2010; Lakhani et al., 2010; Nag et al., 2012; Nag, 2012; Li et al., 2013; Wu et al., 2013b; Tajedin and Nevo, 2014; Stol and Fitzgerald, 2014c).

Stol et al. (2014c) 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 (Stol and Fitzgerald, 2014a) 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 advice for crowdsourced software development. For instance, the requester should provide the crowd with clear documents and avoid anonymously interaction with crowd developers (Fitzgerald and Stol, 2015).

Tajedin and Nevo (2014) 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. (2013b) highlighted the lessons learned from their collected software crowdsourcing data. Two crowdsourced software development processes employed by TopCoder and AppStori were examined. The paper argued that the ‘min-max’ competition behaviour contributes to the quality and creativity of crowdsourced software development.

Nag et al. (2012) reported their collaboration with TopCoder to crowdsourc spaceflight software development for the SPHERES Zero Program, supported by NASA, DARPA and Aurora Flight Sciences. A detailed version can be found in Nag’s Master’s thesis (Nag, 2012).

Lakhani et al. (2010) described the development of TopCoder from the year of 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 (2010) conducted an empirical analysis of developers’ strategic behaviour on TopCoder. The cheap talk (Farrell and Rabin, 1996) 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 marketplace. 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

<table>
<thead>
<tr>
<th>Platform</th>
<th>Since</th>
<th>URL</th>
<th>Task Domain</th>
<th>Open Call Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>TopCoder</td>
<td>2001</td>
<td><a href="http://www.topcoder.com">www.topcoder.com</a></td>
<td>Software Development</td>
<td>Online Competition</td>
</tr>
<tr>
<td>GetACoder</td>
<td>2004</td>
<td><a href="http://www.getacoder.com">www.getacoder.com</a></td>
<td>Software Development</td>
<td>Online Bidding</td>
</tr>
<tr>
<td>AppStori</td>
<td>2013</td>
<td><a href="http://www.appstori.com">www.appstori.com</a></td>
<td>Mobile App Development</td>
<td>Crowd Funding, Online Recruiting</td>
</tr>
<tr>
<td>Bountify</td>
<td>2013</td>
<td><a href="http://www.bountify.co">www.bountify.co</a></td>
<td>Small Coding Tasks</td>
<td>Online Competition</td>
</tr>
<tr>
<td>uTest</td>
<td>2007</td>
<td><a href="http://www.uteest.com">www.uteest.com</a></td>
<td>Software Testing</td>
<td>On-demand Matching, Online Com-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>petition</td>
</tr>
<tr>
<td>Testbats</td>
<td>2013</td>
<td><a href="http://www.testbats.com">www.testbats.com</a></td>
<td>Software Testing</td>
<td>On-demand Matching</td>
</tr>
<tr>
<td>Mob4hire</td>
<td>2008</td>
<td><a href="http://www.mob4hire.com">www.mob4hire.com</a></td>
<td>Mobile App Testing</td>
<td>Online Bidding</td>
</tr>
<tr>
<td>Ce WooYun</td>
<td>2012</td>
<td>ce.wooyun.org</td>
<td>Software Security Testing</td>
<td>On-demand Matching</td>
</tr>
</tbody>
</table>

Table 5: A List of Commercial Platforms for Crowdsourced Software Engineering

---

25http://www.mathworks.com
payment and the number of requirements factors were identified as significant predictors for final submission quality. Li et al. (2013) also conducted a case study on TopCoder. A set of 23 quality factors were identified from the aspects of project and platform.

Regarding the case studies that were based on the platforms other than TopCoder: Zogaj et al. (2013; 2014) 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 (2013) 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, e.g., effectively outsourced their development tasks to global online developers while sidestepping some costs incurred by directly employing high-tech workers.

Some case studies focused on region-specific practices in crowdsourced software development. For example, one case study (Machado et al., 2014; Prikladnicki et al., 2014) 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 (Phair, 2012) 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 (Martin et al., 2013) and crowdsourced e-government software development (Shah et al., 2009; Warner, 2011).

5. 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 6. The references that map to each of the software engineering tasks are given in Table 7. The commercial and experimental crowdsourcing platforms in these studies follow the scheme in Fig. 10.

A timeline of the introduction of various ideas and concepts is illustrated in Fig. 11. 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 8, 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. 12 and Fig. 13 respectively.

5.1. Crowdsourcing for Software Requirements Analysis

Requirements analysis is a widely accepted critical step that impacts the success of software projects (Standish, 1994). A series of studies (Lim et al., 2010b,a; Seyff et al., 2010; Lim et al., 2011; Ali et al., 2011; Lim and Finkelstein, 2012; Adepetu et al., 2012; Nascimento et al., 2012; Muganda et al., 2012; Greenwood et al., 2012; Hosseini et al., 2013; Lim and Neube, 2013; Snijders and Dalmia, 2014; Wang et al., 2014; Nayebi and Ruhe, 2014; Breaux and Schaub, 2014; Hosseini et al., 2015) have investigated crowdsourcing to support this process.
<table>
<thead>
<tr>
<th>SE Phase</th>
<th>SE Task</th>
<th>Why</th>
<th>Bespoke Tool</th>
<th>Stakeholder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Acquisition</td>
<td>Requirements Categorisation</td>
<td>User needs</td>
<td>None</td>
<td>Requirements engineers, Designers</td>
</tr>
<tr>
<td>Requirements</td>
<td>Requirements Acquisi-</td>
<td>StakeSource, StakeSource2.0, StakeNet,</td>
<td>StakeNet, iRequire</td>
<td>Email, StakeSource, StakeSource2.0, StakeNet,</td>
</tr>
<tr>
<td></td>
<td>tion</td>
<td>StakeRare, StakeRare, tRequire</td>
<td></td>
<td>iDesign, Bespoke, AOI, AMT</td>
</tr>
<tr>
<td>Design</td>
<td>User Interface Design</td>
<td>User needs, Quality</td>
<td>None</td>
<td>Designers, Non-technical end users</td>
</tr>
<tr>
<td></td>
<td>Architecture Design</td>
<td>Quality, Diversity</td>
<td>None</td>
<td>Email</td>
</tr>
<tr>
<td></td>
<td>Design Revision</td>
<td>Quality, Diversity</td>
<td>None</td>
<td>Email</td>
</tr>
<tr>
<td>Coding</td>
<td>IDE Enhancement</td>
<td>Debugging, API aid</td>
<td>BlueFix, Calcine, Example Over-flow, Sea-hawk, SnapMatch</td>
<td>Developers</td>
</tr>
<tr>
<td></td>
<td>Program Optimisation</td>
<td>Human solutions</td>
<td>None</td>
<td>Bountify, AMT, Software Feedback</td>
</tr>
<tr>
<td></td>
<td>Crowd Programming</td>
<td>Automation, Human solutions</td>
<td>CrowdLang, Collabode</td>
<td>Developers, Users, Developers</td>
</tr>
<tr>
<td></td>
<td>Support</td>
<td></td>
<td>CIDRE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Developers, Teachers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Usability Testing</td>
<td>Cost, Time</td>
<td>CrowdStudy</td>
<td>Testers, Bespoke, Users</td>
</tr>
<tr>
<td></td>
<td>Performance Testing</td>
<td>Real-world measure</td>
<td>None</td>
<td>Users</td>
</tr>
<tr>
<td></td>
<td>GUI Testing</td>
<td>Cost, Scalability</td>
<td>None</td>
<td>Users</td>
</tr>
<tr>
<td></td>
<td>QoE Testing</td>
<td>Cost, Diversity</td>
<td>Quadrant of Euphoria</td>
<td>Researchers, Users</td>
</tr>
<tr>
<td></td>
<td>Test Generation</td>
<td>Human inputs</td>
<td>PAT</td>
<td>Twitter</td>
</tr>
<tr>
<td></td>
<td>Oracle Problem</td>
<td>Human solutions, Automation</td>
<td>None</td>
<td>AMT</td>
</tr>
<tr>
<td></td>
<td>Crowd Testing Support</td>
<td>Human inputs</td>
<td>CrowdBlaze</td>
<td>Developers, Users, Unqualified</td>
</tr>
<tr>
<td></td>
<td>General Evaluation</td>
<td>User needs, Diversity</td>
<td>None</td>
<td>Bespoke, AMT, Mobileworks, Email</td>
</tr>
<tr>
<td>Verification</td>
<td>Non-expert Verification</td>
<td>Cost, Speed</td>
<td>Verification Games, VeriWeb</td>
<td>Developers, Users, Developers</td>
</tr>
<tr>
<td>Evolution</td>
<td>Software Adaptation</td>
<td>User costs, Cost, Diversity, Speed</td>
<td>MoWA, CrowdAdapt</td>
<td>Bespoke, Facebook, Online commu-</td>
</tr>
<tr>
<td>Maintenance</td>
<td>Software Documenta-</td>
<td>Domain knowledge</td>
<td>COFAQ</td>
<td>Developers, Users, Researcher</td>
</tr>
<tr>
<td></td>
<td>tion</td>
<td></td>
<td></td>
<td>Q&amp;A, Stack Overflow, SciPy Commu-</td>
</tr>
<tr>
<td></td>
<td>Software Localisation</td>
<td>Domain knowledge, Cost, Speed</td>
<td>None</td>
<td>Developers, Researchers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AMT</td>
</tr>
<tr>
<td>Security and Privacy Augmentation</td>
<td>Security and Privacy Augmentation</td>
<td>Diversity, Domain knowledge, User needs</td>
<td>Crowdroid, Modding-Interface, Crowd-Source, SmartNotes, CrowdAdapt, Modding-Interface, Crowd-Source,</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>End User Support</td>
<td>Domain knowledge</td>
<td>LemonAid</td>
<td>Developers, Researchers</td>
</tr>
<tr>
<td></td>
<td>Software Ideation</td>
<td>User needs, Open innovation, contribution</td>
<td>SAPlents, IdeaMax</td>
<td>Client Companies</td>
</tr>
</tbody>
</table>

Table 6: An Overview of the Research on Crowdsourced Software Engineering
Table 7: Reference Mapping of the Research on Crowdsourced Software Engineering

<table>
<thead>
<tr>
<th>SE Phase</th>
<th>SE Task</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>Requirements Acquisition</td>
<td>(Lim et al., 2010b; Seyff et al., 2010; Lim et al., 2011; Ali et al., 2011; Lim and Finkelstein, 2012; Adepeta et al., 2012; Nascimento et al., 2012; Muganda et al., 2012; Greenwood et al., 2012; Hosseini et al., 2013; Lim and Neube, 2013; Snijders and Dalpiaz, 2014; Wang et al., 2014; Nayebe and Ruhne, 2014; Breaux and Schaub, 2014; Hosseini et al., 2015)</td>
</tr>
<tr>
<td></td>
<td>Requirements Categorisation</td>
<td>(Muganda et al., 2012; Nascimento et al., 2012)</td>
</tr>
<tr>
<td>Design</td>
<td>User Interface Design</td>
<td>(Bernstein, 2010; Nebeling et al., 2012a; LaToza et al., 2015; Lasecki et al., 2015)</td>
</tr>
<tr>
<td></td>
<td>Architecture Design</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Design Revision</td>
<td>(LaToza et al., 2015)</td>
</tr>
<tr>
<td>Coding</td>
<td>IDE Enhancement</td>
<td>(Hartmann et al., 2010; Mooy et al., 2010; Bruch et al., 2010; Kallenbach, 2011; Zagalsky et al., 2012; Bruch, 2012; Watson et al., 2012; Bacchelli et al., 2012; Ponzanelli, 2012; Ponzanelli et al., 2013a;b; Wightman, 2013; Barzilay et al., 2013; Ponzanelli et al., 2014a;b; de Souza et al., 2014; Fast et al., 2014; Chen and Kim, 2015)</td>
</tr>
<tr>
<td></td>
<td>Program Optimisation</td>
<td>(Auler et al., 2014; Cochran et al., 2015)</td>
</tr>
<tr>
<td></td>
<td>Crowd Programming Support</td>
<td>(Goldman et al., 2011; Goldman, 2011; Minder and Bernstein, 2011, 2012; Goldman, 2012; Ball et al., 2014; Xie et al., 2015)</td>
</tr>
<tr>
<td>Testing</td>
<td>Performance Testing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GUI Testing</td>
<td>(Vliegendhart et al., 2012; Dolstra et al., 2013)</td>
</tr>
<tr>
<td></td>
<td>QoE Testing</td>
<td>(Chen et al., 2010; Gardoi et al., 2014; Hossfeld et al., 2014a,b)</td>
</tr>
<tr>
<td></td>
<td>Test Generation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Oracle Problem Mitigation</td>
<td>(Kosonore et al., 2015)</td>
</tr>
<tr>
<td></td>
<td>Crowd Testing Support</td>
<td>(Xia, 2013; Yan et al., 2014; Liang et al., 2014)</td>
</tr>
<tr>
<td>Verification</td>
<td>Non-expert Verification</td>
<td>(Dietl et al., 2012; Li et al., 2012; Schiller and Ernst, 2012; Schiller, 2014)</td>
</tr>
<tr>
<td>Evolution</td>
<td>Software Adaptation</td>
<td>(Bacon et al., 2009; Nebeling and Norrie, 2011a;b; Maalej and Pagano, 2011; Ali et al., 2011, 2012; Akiki et al., 2013; Challiol et al., 2013; Nebeling et al., 2013a; He et al., 2014; Almalki et al., 2014; Hamidi et al., 2014)</td>
</tr>
<tr>
<td>Maintenance</td>
<td>Software Documentation</td>
<td>(Jiao and Yang, 2012; Parmin et al., 2012; Barzilay et al., 2013; Chen and Zhang, 2014; Pashik et al., 2014)</td>
</tr>
<tr>
<td></td>
<td>Software Localisation</td>
<td>(Estion et al., 2009; Manzoor, 2011; Gitt, 2012; Mijnhardt, 2015)</td>
</tr>
<tr>
<td>Other</td>
<td>Security and Privacy Augmentation</td>
<td>(Arellano et al., 2010; Burguera et al., 2011; Sharifi et al., 2011; Lin et al., 2012, 2013; Agarwal and Hall, 2013; Papamartizonas et al., 2014; Sae et al., 2014; Ismail et al., 2015)</td>
</tr>
<tr>
<td></td>
<td>End User Support</td>
<td>(Chilana et al., 2012; Chilana, 2013)</td>
</tr>
<tr>
<td></td>
<td>Software Ideation</td>
<td>(Elmer et al., 2008; Kiemar et al., 2009; Jayakrishnan and Sundrarajan, 2011a,b)</td>
</tr>
</tbody>
</table>
Figure 11: Timeline for 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).

Traditional stakeholder analysis tools require experts’ manual effort to extract stakeholders’ information. Lim et al. (2010b) 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 a complementary to their previously proposed StakeNet (Lim et al., 2010a), which recommends stakeholders via social networking. The authors further improved this tool and proposed StakeSource2.0 (Lim et al., 2011). 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 (Lim and Finkelstein, 2012) method, an approach for large-scale requirements elicitation based on social network analysis and collaborative filtering techniques. Lim and Ncube (2013) subsequently showed the application of the tool for systems projects. The tool is publicly available online.

Hosseini et al. (2013) 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. (2014) 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. (2012) investigated the use of crowdsourcing for requirements categorisation based on Kano’s model. The model uses a 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 (2014) presented the Analytical Open Innovation (AOI) approach to assist developers in making 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. This is a laborious task when performed manually, to extract requirements from large natural language text source. However, such data are frequently needed as the ground truth for evaluation. This limits the generalisation of evaluations to automatic requirements extraction methods.
Breaux and Schaub (2014) 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. (2012) 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.

5.2. 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. (2013) leveraged the crowd to map between mobile app wireframes and design examples over the Internet. Lasecki et al. (2015) 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 (LaToza et al., 2015). LaToza et al. (2015) 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. (2012a) also proposed to evolve software designs based on 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.

5.3. Crowdsourcing for Software Coding

Using crowdsourcing for software coding has focused on three sub-areas: crowd programming environments, program optimisation and integrated development environment (IDE) enhancement. Unlike work that crowdsources software engineering tasks directly to the general public, studies on crowd programming environments and IDE enhancement indirectly use crowdsourcing to support software engineering activities: The studies in the former category enable the use of crowdsourcing for coding tasks in building software. Those in the latter category tend to leverage existing pre-collected crowd knowledge to aid software coding and/or debugging. We also include these papers in this survey, because they meet our inclusion criteria on using crowdsourcing in software engineering.

1) Crowd programming environments: Crowdsourcing intermediaries play a key role in managing and coordinating the crowd workers to accomplish the requesters’ tasks. Many studies focused on providing systems to support crowd-based coding tasks (Goldman et al., 2011; Goldman, 2011; Minder and Bernstein, 2011, 2012; Goldman, 2012; Ball et al., 2014; LaToza et al., 2014a; Xie et al., 2015).

Goldman (2011) 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. (2014) demonstrated the design of the Cloud-based Integrated Development and Runtime Environment (CIDRE), and its implementation TouchDevelop ( Tillmann et al., 2011). 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. (2015) proposed to use Code Hunt (Bishop et al., 2015) from Microsoft Research as a platform for crowd programming. The platform provides coding duel games with various difficulty levels to attract online developers’ participation.
<table>
<thead>
<tr>
<th>Phase</th>
<th>SE Task</th>
<th>Platform</th>
<th>Crowd</th>
<th>Size</th>
<th>Subject</th>
<th>Effort</th>
<th>Reward</th>
<th>Result</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>StakeNet Stakeholders</td>
<td>68</td>
<td>RALIC</td>
<td>-</td>
<td>StakeNet can identify stakeholders and their roles with high recall, and can prioritize them accurately.</td>
<td>-</td>
<td>-</td>
<td>(Lim et al., 2010a)</td>
<td></td>
</tr>
<tr>
<td>Requirement Extraction</td>
<td>StakeNet Stakeholders</td>
<td>87</td>
<td>RALIC</td>
<td>-</td>
<td>StakeNet can predict and prioritize stakeholder needs accurately.</td>
<td>-</td>
<td>-</td>
<td>(Lim et al., 2010a)</td>
<td></td>
</tr>
<tr>
<td>Requirements</td>
<td>AMT Unskilled</td>
<td>38</td>
<td>-</td>
<td>20</td>
<td>The approach can reduce 60% cost and increase 16% coverage in manual extraction.</td>
<td>-</td>
<td>-</td>
<td>(Bouraux and Schaub, 2014)</td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td>Email Students</td>
<td>20</td>
<td>12.9 hours (average)</td>
<td>$100 each person + 4*1000 prizes</td>
<td>All participants borrowed others’ design ideas and most improved the design quality.</td>
<td>-</td>
<td>-</td>
<td>(LaTour et al., 2015)</td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td>StackOverFlow community</td>
<td>-</td>
<td>579 HITs</td>
<td>$347.16 ($0.20 per HIT)</td>
<td>Crowdsourced evaluation tasks can be repeated over time and maintain reliable results.</td>
<td>-</td>
<td>-</td>
<td>(Bianco et al., 2011)</td>
<td></td>
</tr>
<tr>
<td>Coding</td>
<td>Unskilled</td>
<td>33</td>
<td>597 HITs</td>
<td>$347.16 ($0.20 per HIT)</td>
<td>Reduced cost and time, however quality was worse compared to the testing in a lab setting.</td>
<td>-</td>
<td>-</td>
<td>(Liu et al., 2012)</td>
<td></td>
</tr>
<tr>
<td>IDE Enhancement</td>
<td>AMT Unskilled</td>
<td>84</td>
<td>282 HITs</td>
<td>$347.16 ($0.20 per HIT)</td>
<td>The crowdsourced usability testing shared similar results with a laboratory setting.</td>
<td>-</td>
<td>-</td>
<td>(Nebeling et al., 2013b)</td>
<td></td>
</tr>
<tr>
<td>GUI Testing</td>
<td>AMT Unskilled</td>
<td>100</td>
<td>0.15 HITs</td>
<td>$0.05-$0.25 per task</td>
<td>Consistent program boosts in accuracy can be achieved with modest monetary cost.</td>
<td>-</td>
<td>-</td>
<td>(Chen et al., 2010)</td>
<td></td>
</tr>
<tr>
<td>Code Testing</td>
<td>AWS</td>
<td>33</td>
<td>$25</td>
<td>$0.10-$0.15 per HIT</td>
<td>The approach is feasible and reliable, although there was a delay in obtaining results.</td>
<td>-</td>
<td>-</td>
<td>(Boehle et al., 2013)</td>
<td></td>
</tr>
<tr>
<td>Quality Assurance</td>
<td>AMT Unskilled</td>
<td>100</td>
<td>0.15 HITs</td>
<td>$0.05-$0.25 per task</td>
<td>With consistent quality assurance, crowdsourcing can yield results as well as laboratory experiments.</td>
<td>-</td>
<td>-</td>
<td>(Chen et al., 2010)</td>
<td></td>
</tr>
<tr>
<td>Test Generation</td>
<td>Unskilled</td>
<td>65</td>
<td>0.15 HITs</td>
<td>$0.0625 per rating</td>
<td>The proposed methods experienced a step in making crowdsourcing sufficiently mature for wide adoption.</td>
<td>-</td>
<td>-</td>
<td>(Rose et al., 2014)</td>
<td></td>
</tr>
<tr>
<td>GUI Testing</td>
<td>AMT Unskilled</td>
<td>33</td>
<td>0.15 HITs</td>
<td>$0.0834 per rating</td>
<td>The approach was able to evaluate an experimental user interface within a few days at low costs.</td>
<td>-</td>
<td>-</td>
<td>(Chen et al., 2013)</td>
<td></td>
</tr>
<tr>
<td>Verification</td>
<td>AMT Unskilled</td>
<td>14</td>
<td>3 hours (average)</td>
<td>$6.22 per hour &gt; $0.25 per HIT</td>
<td>VeriWeb can save time and money with contracted workers.</td>
<td>-</td>
<td>-</td>
<td>(Schelle and Ernst, 2012)</td>
<td></td>
</tr>
</tbody>
</table>

(RALIC is the acronym of “Replacement Access, Library and ID Card”, which is an access control system developed by University College London.
A Human Intelligent Task (HIT) is a single, self-contained task that workers can perform.
AMT and CF refer to Amazon Mechanical Turk and CrowdFlower, respectively.
Tribler is a peer-to-peer file-sharing client. KDE and Xfce are two desktop environments for Unix-like platforms.)
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’ duelling process, the multiple attempts with evolving code versions can serve for education purpose.

2) Program optimisation: More recently, crowdsourcing has been used to support compilation optimisation (Auler et al., 2014) and program synthesis (Cochran et al., 2015). Auler et al. (2014) presented a crowdsourced adaptive compiler 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. (2015) proposed an approach called Program Boosting, which uses crowd knowledge and genetic programming 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 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.

3) 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 (Hartmann et al., 2010; Mooty et al., 2010; Bruch et al., 2010; Kallenbach, 2011; Zagalsky et al., 2012; Bruch, 2012; Watson et al., 2012; Bacchelli et al., 2012; Ponzanelli, 2012; Ponzanelli et al., 2013a,b; Wightman, 2013; Barzilay et al., 2013; Ponzanelli et al., 2014a,b; de Souza et al., 2014; Fast et al., 2014; Chen and Kim, 2015), each of which we describe below.

HelpMeOut (Hartmann et al., 2010) 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 (Mujumdar et al., 2011) was proposed. More recently, Chen and Kim (2015) presented the ‘crowd debugging’ technique that reveals defective code blocks and further suggests solutions for fixing the defects. To achieve the automated detection and recommendation, mass crowd knowledge (question-answer pairs from Stack Overflow) were collected and analysed. The proposed approach was able to identify 171 bugs in 8 high-quality open source Java projects.

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 (Langdon and Harman, 2015; Petke et al., 2014; Orlov and Sipper, 2011; White et al., 2011), and in particular work on automated bug fixing (aka ‘patching’ or ‘automated program repair’) (Le Goues et al., 2012). 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 (Langdon and Harman, 2015; Langdon et al., 2015; Petke et al., 2014; Orlov and Sipper, 2011; White et al., 2011), energy (Bruce et al., 2015; Manotas et al., 2014; Li et al., 2014) and dynamic memory (Wu et al., 2015) consumption and functionality, both by fixing bugs (Le Goues et al., 2013) and by adding new features (Harman et al., 2014b). Work on automated repair has also harvested human developed patches in order to improve the automated repair process (Nguyen et al., 2013). 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 7.3.

BlueFix (Watson et al., 2012) is an online tool concerned with the problem of interpreting and understanding compiler error messages 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 (Mooty et al., 2010) 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 increase their completion rate by 40%.

Example Overflow (Zagalsky et al., 2012; Barzilay et al., 2013) 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 functionality is based on Apache Lucene. A preliminary evaluation on a subset of coding tasks indicated that the results suggested by the system were better than the ones from other existing tools studied in the experiments.

Seahawk (Bacchelli et al., 2012; Ponzanelli, 2012; Ponzanelli et al., 2013a,b) 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 (Bacchelli et al., 2012). By enabling developers to link imported code snippets to their documents via language-independent annotations, Seahawk helps developers share documents with their teammates (Ponzanelli et al., 2013b). The evaluation experiments were performed on 35 exercises from Java training courses (Ponzanelli et al., 2013a). The results were generally promising. Although the tool might not always suggest useful documents, it sometimes aided developers with surprising insights.

WordMatch and SnipMatch (Wightman, 2013) 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.

Souza et al. (2014) 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.

Amann et al. (2014) 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 (2012) 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 Web 2.0 can be leveraged to improve the developer’s IDE.

Fast et al. (2014) 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. (2014), 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 (Afshan et al., 2013; Bozkurt and Harman). As with the potential for the 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.

5.4. 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.

5.4.1. 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.

Crowdsourcing has been applied to various types of testing activities, including usability testing (Schneider and Cheung, 2011; Liu et al., 2012; Nebeling et al., 2012b; Meier et al., 2013; Nebeling et al., 2013b; Teinum, 2013; Gomide et al., 2014), performance testing (Musson et al., 2013), GUI testing (Vliegendhart et al., 2012; Dolstra et al., 2013), test case generation (Chen and Kim, 2012; Pham et al., 2013b), and the oracle problem (Pastore et al., 2013). We discuss each of these below:

1) Usability Testing: Traditional usability testing is labour-intensive and can be expensive and time-consuming (Liu et al., 2012). 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’ (Schneider and Cheung, 2011). However, Liu et al. (2012) 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. (2013b) 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 (2011) 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. (2012) 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. (2012b; 2013b) proposed a framework with a toolkit implementation named CrowdStudy for crowdsourced website usability.
testing. For identifying outliers in the crowdsourced usability testing results, Gomide et al. (2014) proposed an approach that employs 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. (2013) 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 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 built-in telemetries (performance testing framework) (Akhave and Felt, 2013).

3) GUI Testing: Automated GUI test case generation is difficult, while manual GUI testing is too slow for many applications (Memon et al., 2003). It is a challenging task to test a GUI continuously. Crowdsourcing is considered as a promising approach for continuous GUI testing (Dolstra et al., 2013).

Vliegenghart et al. (2012) 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. (2013) 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 (Lakhotia et al., 2009), due to several non-trivial tasks that are difficult for programs but may not be so hard for humans (Chen and Kim, 2012). Chen and Kim (2012) investigated object mutation and constraint solving issues, underlying existing test generation tools such as jCUTE (Sen and Agha, 2006), Randoop (Pacheco et al., 2007) and Pex (Tillmann and de Halleux, 2008). 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 coverage of two state-of-art test case generation methods.

Pham et al. (2013a) 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 (Pham et al., 2013b). 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 (Weyuker, 1982; Barr et al., 2015). Such oracles may need to rely on human input (Peters and Parnas, 1998), which makes it hard to fully automate software testing. Pastore et al. (2013) 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 (Xue, 2013; Yan et al., 2014; Liang et al., 2014):

CrowdBlaze (Xue, 2013) 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. Compared to employing automatic testing alone, the proposed system was demonstrated to cover 66.6% more user interfaces according to the evaluation results.

iTest (Yan et al., 2014) is a framework for mobile applications with more automation features than existing industrial mobile application testing service platforms such as uTest and MobHire: the crowd testers are selected via a greedy algorithm, and the generated test results and logs in the framework are submitted automatically.

Caiipa (Liang et al., 2014) 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 (2012) summarised three types of cooperative testing and
analysis: human-tool, tool-tool and human-human cooperation. The 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 (Blanco et al., 2011; Sherief et al., 2014; Sherief, 2014) and more specific evaluation of Quality of Experience (QoE) (Chen et al., 2010; Gardlo et al., 2014; Hossfeld et al., 2014a,b).

5.4.2. Crowdsourcing for Software Verification

Current software verification techniques generally require skilled workers, thereby raising cost issues. Crowdsourcing may reduce the skill barriers and costs for software verification (Dietl et al., 2012; Li et al., 2012; Schiller and Ernst, 2012; Akiki et al., 2013; Schiller, 2014).

DARPA published a solicitation for game-based large scale software verification in 2011, which is named the Crowd Sourced Formal Verification (CSFV) program. A series of research and practice (Ernst and Popović, 2012; Dietl et al., 2012; Watro et al., 2014) were conducted under this program. Dietl et al. (2012) 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. (2012) 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 predefined templates.

Schiller and Ernst (2012) 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 (Schiller, 2014).

5.5. 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 (Bacon et al., 2009; Exton et al., 2009; Manzoor, 2011; Maalej and Pagano, 2011; Parnin et al., 2012; Iiau and Yang, 2012; Gritti, 2012; Ali et al., 2012; Mijnhardt, 2013; Chen and Zhang, 2014; Almaliki et al., 2014; Hamidi et al., 2014; Pawlik et al., 2014; He et al., 2014).

5.5.1. Crowdsourced Software Evolution

Formal or automated verification methods may fail to scale to large software systems (Bacon et al., 2009). To help scalability, a market-based software evolution mechanism was proposed by Bacon et al. (Bacon et al., 2009). 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.

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. (2011) 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 (Ali et al., 2012), 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. (2014) 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. (2013) proposed a crowdsourcing approach for adapting mobile web applications based on client-side adaptation.

Nebeling and Norrie (2011a; 2011b) 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 (Nebeling et al., 2013a) 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. (2013) focused on utilising crowdsourcing 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 results on usability, efficiency and effectiveness of the approach.

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. (2014) 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%.

5.5.2. 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 (Cook and Visconti, 1994;
Visconti and Cook, 2002; Kajko-Mattsson, 2005). Several researchers have investigated crowdsourcing models to enhance software documentation (Jiau and Yang, 2012; Parnin et al., 2012; Barzilay et al., 2013; Chen and Zhang, 2014; Pawlik et al., 2014).

Jiau and Yang (2012) 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. (2012) 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 (2014) 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.

Pawlik et al. (2014) 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.

5.5.3. Crowdsourcing for Software Localisation

Software localisation is also relevant to ‘software internationalisation’ or ‘globalisation’ (Manzoor, 2011), 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 (Esselink, 2000).

Research on utilising crowdsourcing for software localisation (Exton et al., 2009; Manzoor, 2011; Gritti, 2012; Mijnhardt, 2013) aim to reduce the cost and time-to-market periods of the traditional developer-based localisation process.

Exton et al. (2009) first proposed the idea to use crowdsourcing for software localisation. Manzoor (2011) 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 (2012) also worked on a similar project and established a prototype system for crowdsourced translation and software localisation.

5.6. Crowdsourcing for Other Software Engineering Activities

Crowdsourcing has also been applied to support other software engineering activities, such as software security and privacy analysis (Arellano et al., 2010; Burguera et al., 2011; Sharifi et al., 2011; Lin et al., 2012; Lin, 2013; Agarwal and Hall, 2013; Papamartzivanos et al., 2014; Saxe et al., 2014; Ismail et al., 2015), software end user support (Chilana et al., 2012, 2013; Chilana, 2013) and software ideation (Eben et al., 2008; Krcmar et al., 2009; Jayakaranth and Sundararajan, 2011a,b).

Many previous studies have demonstrated that crowdsourcing is an effective way to augment software security (Arellano et al., 2010; Burguera et al., 2011; Sharifi et al., 2011; Agarwal and Hall, 2013; Papamartzivanos et al., 2014; Saxe et al., 2014): Arellano (2010) 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. Sharifi et al. (2011) implemented a system called SmartNotes for detecting security threats underlying web browsing.

The increasing number of malicious mobile apps makes malware analysis an urgent problem. Burguera et al. (2011) presented a novel crowdsourced framework named Crowdroid for detecting Android malware. 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 3 self-written apps. In another real-world app experiment, the detection accuracies were 85% and 100% for two real malware specimens.

Users frequently struggle with reviewing permissions requested by mobile apps. Inappropriately granted permission may cause privacy leaks. Lin (2013) collected 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 to provide default permission settings for mitigating user burden. An 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. Agarwal and Hall (2013) introduced a crowdsourced recommendation engine called ProtectMyPrivacy, which detects and deals with privacy leaks for iOS devices. Papamartzianos et al. (2014) introduced a cloud-based architecture which is driven by the crowd for privacy analysis of mobile apps. Ismail et al. (2015) proposed a crowd manage strategy for security configuration exploration, aiming to find minimal permission sets that preserve app usability. The experiment conducted via a small crowd of 26 participants demonstrated the efficiency of the proposed strategy and the usefulness of the recommended configurations.

Regarding crowdsourced end user support, Chilana et al. (2012; 2013; 2013) 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 the screen. The evaluation performed on Amazon Mechanical Turk showed that LemonAid was able to 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 study (by deploying LemonAid to 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 (Fry and Weimer, 2010; Stol and Elbaum, 2010; Afshan et al., 2013; Fry et al., 2012; Hartmann et al., 2010). We summarised a few studies on using crowdsourced evaluation for software engineering research in Table 9. 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 (Cleland-Huang et al., 2012) such as Predictive Models in Software Engineering (PROMISE), Mining of Software Repositories (MSR) and Search Based Software Engineering (Harman and Jones, 2001) (SBSE) challenges.

Several authors have anticipated that crowdsourcing will be applied to address more challenges in software engineering research (Chen and Kim, 2012; Hosseini et al., 2013; Tung and Tseng, 2013).

6. Issues and Open Problems

Despite the extensive applications of crowdsourcing in 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. However, few research studies have focused on solutions to address these issues.

According to an in-depth industrial case study on TopCoder (Stol and Fitzgerald, 2014c), 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 (2014b) presented a research framework inspired by the issues identified in the TopCoder case study (Stol and Fitzgerald, 2014c). 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. (2013a) 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 (LaToza and van der Hoek, 2015).

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

6.1. 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 (Kazman and Chen, 2009, 2010; Mao et al., 2013).

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 (Kazman and Chen, 2009, 2010), 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. (2013) 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. (2014) 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.
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 (Wu et al., 2013b). Hu and Wu (2014) 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.

### 6.3. Planning and Scheduling

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

LaToza et al. (2014a) developed an approach for decomposing programming work into micro-tasks. The method breaks down a single higher level task into multiple lower level tasks iteratively, and coordinates work by tracking changes linked to artefacts. A platform called CrowdCode (LaToza et al., 2013b) 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. (2014b) 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 5), two previous studies also offered decomposition approaches: Chen and Kim (2012) decomposed the test generators’ complex constraint solving and object mutation problems into small puzzles, which can be solved by crowd labours. Schiller and Ernst (2012) proposed an online IDE for verification named VertiWeb, which can decompose the verifiable specifications task into manageable sub-problems.

### 6.4. Motivation and Remuneration

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

Tran-Thanh et al. (2014) proposed a bounded multi-armed bandit model for expert crowdsourcing. Specifically, the proposed $\epsilon$-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 (2013) 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 that 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. (2015) employed a 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 Ikonnen (2013) studied how the crowd size and allocated time can affect the performance of software testing. Their results, conducted on 130 students, indicated that multiple crowd workers under time pressure had 71% higher effectiveness (measured by the number of detected bugs) than the single workers 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 sufficient high participation levels in Crowdsourced Software Engineering tasks, Wang et al. (2014) 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.
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. (2013) 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. (2009) 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 (2012) 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 (2014) presented intrinsic motivational factors (e.g., skill variety and peer pressure) and extrinsic motivational factors (e.g., monetary reward and recognition) 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.

6.5. Quality Assurance

Crowd labour is transient and workers vary in expertise and background. The use of such an undefined workforce inherently raises quality questions for crowdsourcing in general (Ipeirotis et al., 2010; Yuen et al., 2011; Allahbakhsh et al., 2013) as well as Crowdsourced Software Engineering (Stol and Fitzgerald, 2014c; Li et al., 2013; LaToza et al., 2013a; Saengkhattiya et al., 2012).

Li et al. (2013) 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. (2012) investigated how crowdsourcing companies deal with the quality assurance challenge by conducting interviews with four companies: Microworkers, Clickchore, 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 (2013) 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 evaluations of Crowdsourced Software Engineering quality assurance as a pressing topic for future work.

6.6. 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 (Wolfson and Lease, 2011; Cox, 2011; Saxton et al., 2013). 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 are also important issues. 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.

7. 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.

7.1. Who is the Crowd?

Except for few studies (Varshney, 2012; Jayakanthan and Sundararajan, 2011b; Vukovic et al., 2010; Akiki et al., 2013), 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 the 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 its users. As the technologies develop, we may (hope to) witness a merging of crowdsourcing with adaptive software engineering (Cheng et al., 2008; Harman et al., 2014a; Oreizy et al., 1999).

7.2. 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.

7.3. Hybrid Crowdsourced Software Engineering

The Crowdsourced Software Engineering solutions surveyed in this paper typically concern the substitution of a crowd-sourced 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 crowdsourced activities will lead to the further development of Hybrid Crowdsourced Software Engineering. Tools such as CrowdBlaze (Section 5.4) 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 5.3).

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 (Harman et al., 2012; Pagano and Maalej, 2013; Guzman and Maalej, 2014; Chen et al., 2014), 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.

7.4. 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, in a previous study on using crowdsourcing for program synthesis (Cochran et al., 2015), 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. Also, 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 (Adamopoulos et al., 2004) and co-evolutionary patching (Arcuri et al., 2008).

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. 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.

7.5. 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 perhaps striking to see the waterfall-like model used by existing platforms such as TopCoder, making such a strong resurgence as a practical methodology underpinning much crowdsourced development work (see Section 4.1). This phenomenon may be 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 (LaToza et al., 2015)). 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.

8. Threats to Validity of this Survey The most relevant threats to validity for this survey study are the potential bias in the literature selection and misclassification.

Literature search and selection. Our online library search was driven by the keywords related to crowdsourcing and software engineering. It is possible that our search missed some studies that implicitly use crowdsourcing without mentioning the term ‘crowdsourcing’, or those studies that explicitly use crowdsourcing in the software engineering activities which cannot be covered by our search terms. To mitigate this issue, we performed an issue-by-issue manual search of the major software engineering conferences and journals published from 2006 to 2015, in order to identify those ‘implicit’ crowdsourcing papers and those leverage crowdsourcing for ‘minority’ activities in software engineering. The term ‘crowdsourcing’ was proposed in 2006, yet there may be related work published before 2006. A subset of this work has been identified via a further ‘snowballing’ process of reference search. We also tried to address this issue by reaching out to authors, asking whether we missed any relevant work.

Another factor that may cause literature selection bias is the definition of crowdsourcing itself. Numerous definitions have been proposed (Estellés-Arolas and González-Ladrón-De-Guevara, 2012; Hetmank, 2013) since 2006. These definitions are still debated in literature and are perhaps, somewhat vague in the context of software engineering (Estellés-Arolas and González-Ladrón-De-Guevara, 2012; Stol and Fitzgerald, 2014c). Our survey may not be exhaustive: it may not cover all the possible (and reasonable) definitions of crowdsourcing. To mitigate the issue, we gave our own definition of Crowdsourced Software Engineering based on the most widely cited definition (Howe, 2006b). We also included those papers in which their authors claim to use crowdsourcing, in order to be comprehensive on the use of crowdsourcing in software engineering.

Literature classification. We manually classified all 210 papers into four top-level categories based on their study types and further classified them into fine-grained sub-categories (see Figure 6) based on their targeted domains/tasks. There is no ground truth labelling for such classification. Even though we referred to the ACM Computing Classification System, the IEEE Taxonomy of Software Engineering Standards and the 2014 IEEE Keywords Taxonomy, there is no well-defined standard methodology of classification regarding the used schemes. To minimise any potential classification error, we carefully analysed the full text of the collected papers and performed the classification by three authors, reaching an average inter-rater agreement of 91.2%. The disagreed/controversial papers were resolved by a further discussion.

9. Conclusions In this survey, we have analysed existing literature on the use of crowdsourcing in software engineering activities and research into these activities. The study has revealed a steadily increasing rate of publication and has presented a snapshot of the research progress of this area from the perspectives of theories, practices and applications. Specifically, theories on crowdsourced software development models, major commercial platforms for software engineering and corresponding case studies, and crowdsourcing applications to software engineering research have been summarised. The study also highlights potential issues in Crowdsourced Software Engineering, together with related analysis and solutions conducted in previous studies. Finally, the survey is used to identify gaps in the literature and open problems for future work.

Acknowledgments The authors would like to thank the many authors who contributed their valuable feedback in the ‘pseudo-crowdsourced’ checking process of this survey, and the anonymous referees for their comments.

Ke Mao is funded by the UCL Graduate Research Scholarship (GRS), and the UCL Overseas Research Scholarship (ORS). This work is also supported by the Dynamic Adaptive Automated Software Engineering (DAASE) programme grant (EP/J017515), which fully supports Yue Jia, partly supports Mark Harman.
References


Mao, K., Yang, Y., Li, M., Harman, M., 2013. Pricing Crowdsourcing Based


Wu, W., Tsai, W.T., Li, W., 2013b. Creative software crowdsourcing: from components and algorithm development to project concept formations. International Journal of Creative Computing 1, 57–91.


