The Journal of Systems and Software 126 (2017) 57–84

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

New trends and ideas

A survey of the use of crowdsourcing in software engineering

Ke Mao*, Licia Capra, Mark Harman, Yue Jia

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

ARTICLE INFO

Article history:
Received 9 January 2016
Revised 11 September 2016
Accepted 13 September 2016
Available online 20 September 2016

Keywords:
Crowdsourced software engineering
Software crowdsourcing
Crowdsourcing
Literature survey

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.

1. Introduction

Crowdsourcing is an emerging distributed problem-solving model based on the combination of human and machine computation. The term 'crowdsourcing' was jointly[1] coined by Howe and Robinson in 2006 (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 Brajawidagda, 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

* Corresponding author.
E-mail addresses: k.mao@cs.ucl.ac.uk (K. Mao), l.capra@ucl.ac.uk (L. Capra), mark.harman@ucl.ac.uk (M. Harman), yue.jia@ucl.ac.uk (Y. Jia).

[1] Jeff Howe attributes the creation of the term to Mark Robinson and himself (Howe, 2006a).

http://dx.doi.org/10.1016/j.jss.2016.09.015
0164-1212/

Table 1
Terms for online library search.

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

to strive for breadth in this survey. The included literature may directly crowdsource 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 typical 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[2] 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).

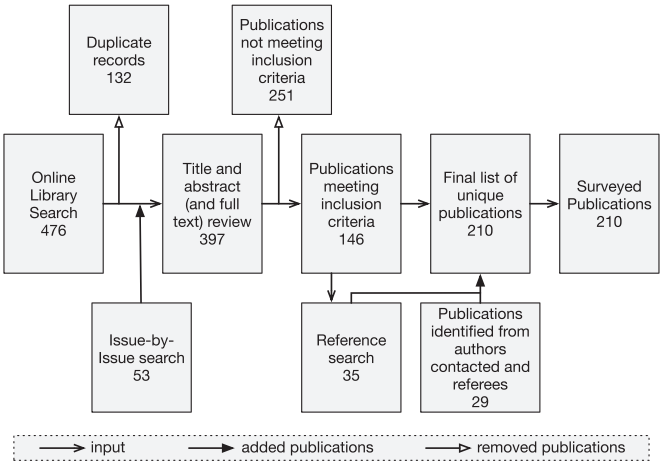

Fig. 1. Results of literature search and selection on using crowdsourcing to support software engineering activities.

We conducted a screening process[3] 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 and filtered out unrelated publications manually. We also performed a 'pseudo-crowdsourced' checking process for this survey. We contacted the authors (via email), to check whether we had missed any important references and whether there was any inaccurate information regarding our description of their work. We then further revised the survey according to the authors' comments: we refined imprecise descriptions of their work and further included relevant papers that satisfied our inclusion criteria.

A detailed workflow of above steps and the number of resulting publications for each step are depicted in Fig. 1. The initial type of search via online digital libraries produced 476 publications in total, where 132 of them are duplicated. The complementary issue-by-issue manual search led to another 53 unique papers. In total, 397 publications were reviewed by examining their titles and abstracts against our inclusion criteria. When the title and abstract of one publication did not give enough information for making a decision, we further reviewed full-text of the paper. This step ex-

[2] 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.

[3] The screening process is iterative, e.g., we also screened the publications suggested by the authors contacted and the anonymous reviewers.

Table 2
Selected conference proceedings and journals for manual search.

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

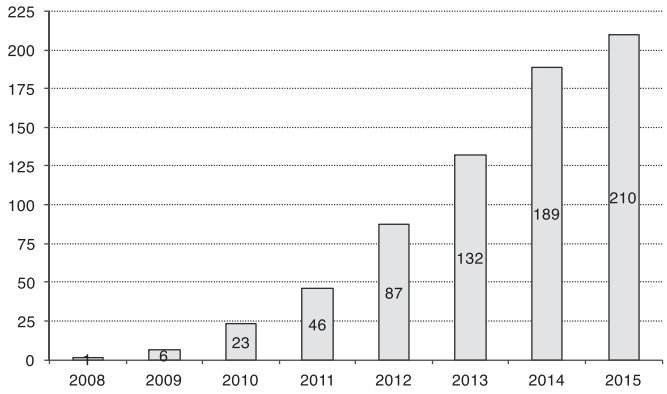

Fig. 2. Cumulative growth of Crowdsourced Software Engineering papers published before April 2015.

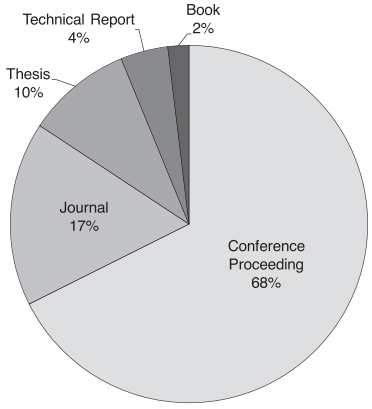

Fig. 3. Publication type of surveyed papers.

cluded 146 publications that did not meet the inclusion criteria. In addition, 35 publications were identified from reference lists of eligible publications. Regarding the 'pseudo-crowdsourced' step, for each included publication, we distributed the copy of this survey to at least one author. In total, we contacted 303 authors and received 83 replies. Twenty-two publications were suggested by these authors and another 7 were identified from the anonymous referees of this survey. Finally, a list of 210 unique publications remained, and were analysed in this survey. The growth trend in publications is presented in Fig. 2. The distribution of these papers' publication types and a specific list of Master/PhD theses can be found in Fig. 3 and Table 3, respectively. As can be seen, there is a noticeable rise in publications on Crowdsourced Software Engineering, resulting in a significant body of literature which we study in this survey.

We have built a repository which contains the meta-data of our collected papers. The meta-data includes the author, title, publication year, type and the conference proceeding/journal information of the paper. Based on this repository, we conducted our analysis of the reviewed papers. This repository is publicly available online[4].

3. Definitions, trends and landscape

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

[4] http://www.cs.ucl.ac.uk/staff/k.mao/cserep.

3.1. Crowdsourcing

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).

Table 3
A list of master and PhD theses on Crowdsourced Software Engineering.

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

Turning to online crowdsourcing, early Internet-based crowdsourcing activities can be found in 2001, when 'InnoCentive'[5] 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 750000 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[6] cites the definition which appeared in the Merriam-Webster dictionary in 2011[7]. It stresses the large group of workers and an online community, but drops any mention of 'undefined labour' and 'open call' format[8]. 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[9] 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 Brajawidagda, 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; Stolee 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 Morisaki, 2011; Wu et al., 2013a; 2013b; 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

[5] http://www.innocentive.com.
[6] http://en.wikipedia.org/wiki/crowdsourcing.
[7] http://www.merriam-webster.com/info/newwords11.htm.
[8] http://www.merriam-webster.com/dictionary/crowdsourcing.
[9] http://en.wikipedia.org/wiki/list of crowdsourcing projects.

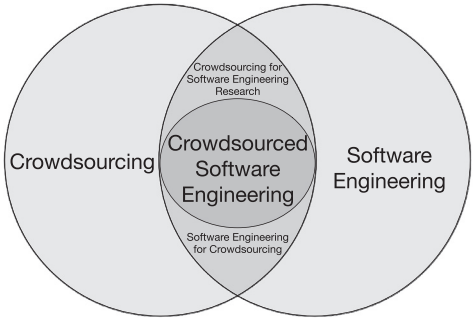

Fig. 4. Crowdsourcing and software engineering.

Table 4
Cited Crowdsourcing definitions

Def.	None	Howe	Wiki	Own	Other
Count	144 (69%)	37 (18%)	4 (2%)	7 (3%)	22 (10%)

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 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[10]), 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).

Stol and Fitzgerald's definition (Stol and Fitzgerald, 2014c) refines Howe's crowdsourcing definition to the software development domain, requiring the undefined labour force to have

[10] One single paper may cite multiple definitions.

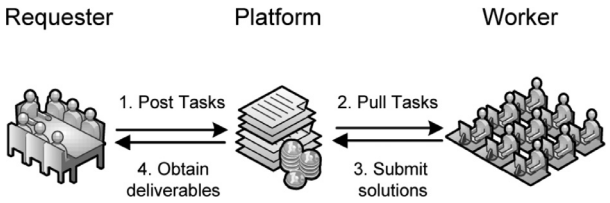

Fig. 5. Actors in Crowdsourced Software Engineering.

requisite specialist knowledge. The definition from the 2013 Dagstuhl seminar on software crowdsourcing (Huhns et al., 2013)[11] is formalised as a Wikipedia page[12] on software crowdsourcing. It also specifies the tasks for software development, according to which the labour force remains unconstrained, yet the characteristic of a large potential workforce is not mentioned.

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. (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.

[11] Subsequently published as a book: http://www.springer.com/gb/book/9783662470107.

[12] http://en.wikipedia.org/wiki/crowdsourcing_software_development.

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[13] for software formal verification and launched the Verigames website to facilitate the practice[14]. NASA and Harvard business school established the NASA Tournament Laboratory for crowdsourcing software solutions for NASA systems[15]. Microsoft crowdsourced partial software development activities in Office 2010[16], Windows 8.1[17] and Windows 10[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[19], the IEEE Taxonomy of Software Engineering Standards[20] and the 2014 IEEE Keywords Taxonomy[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 (3% in our collection of papers) employed crowdsourcing to evaluate software engineering research. This type of publication may not use crowdsourcing-related keywords in their meta information. We performed extra manual retrievals for related research. Nevertheless, there may be more papers which fall into this category yet which remain uncovered in our survey; this category is not the focus of our survey.

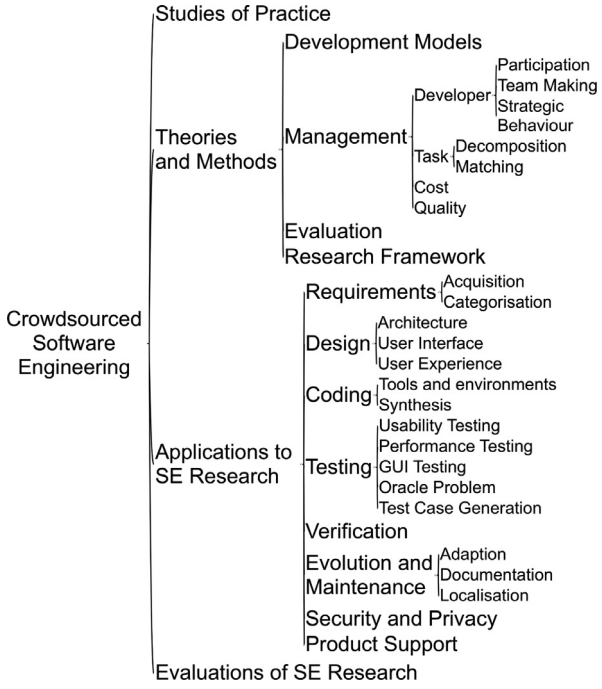

Fig. 6. Taxonomy of research on Crowdsourced Software Engineering.

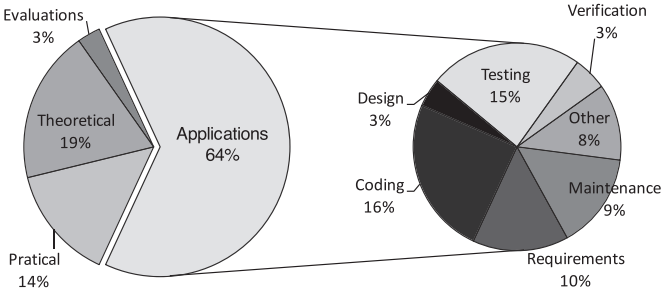

Fig. 7. Research topic distribution.

3.2.3. CSE research landscape

We present the CSE research landscape from two views: a specific view of the Software Development Life Cycle (SDLC) and a general view of the problem-solving process. The former view illustrates a picture of the activities that have been covered in published literature on applying crowdsourcing to software engineering. The latter points out a series of questions and corresponding variables on why and how to use crowdsourcing to tackle a problem in software engineering. These questions have not been widely discussed in a software engineering context. The latter view also identifies several issues remaining open to CSE researchers (who wish to design new crowdsourcing-based approaches) and practitioners (who intend to adopt existing crowdsourcing-based approaches), thus reflecting the landscape of future CSE research topics from a process perspective.

In Fig. 8, we illustrate the domains in which crowdsourcing has been integrated into the SDLC. Crowdsourcing can be used

[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.
[21] http://www.ieee.org/documents/taxonomy_v101.pdf.

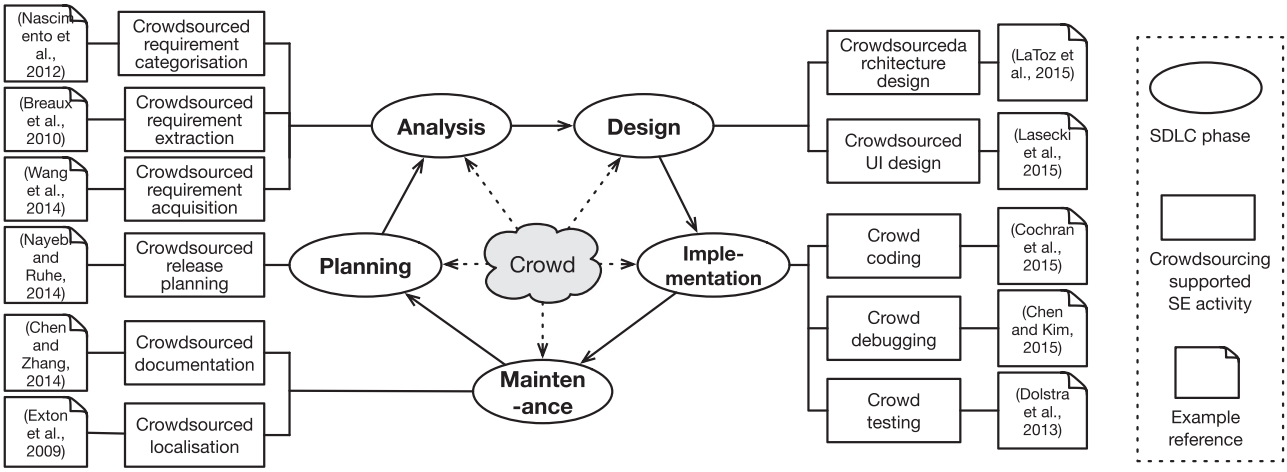

Fig. 8. Integration of crowdsourcing into the Software Development Life Cycle (SDLC).

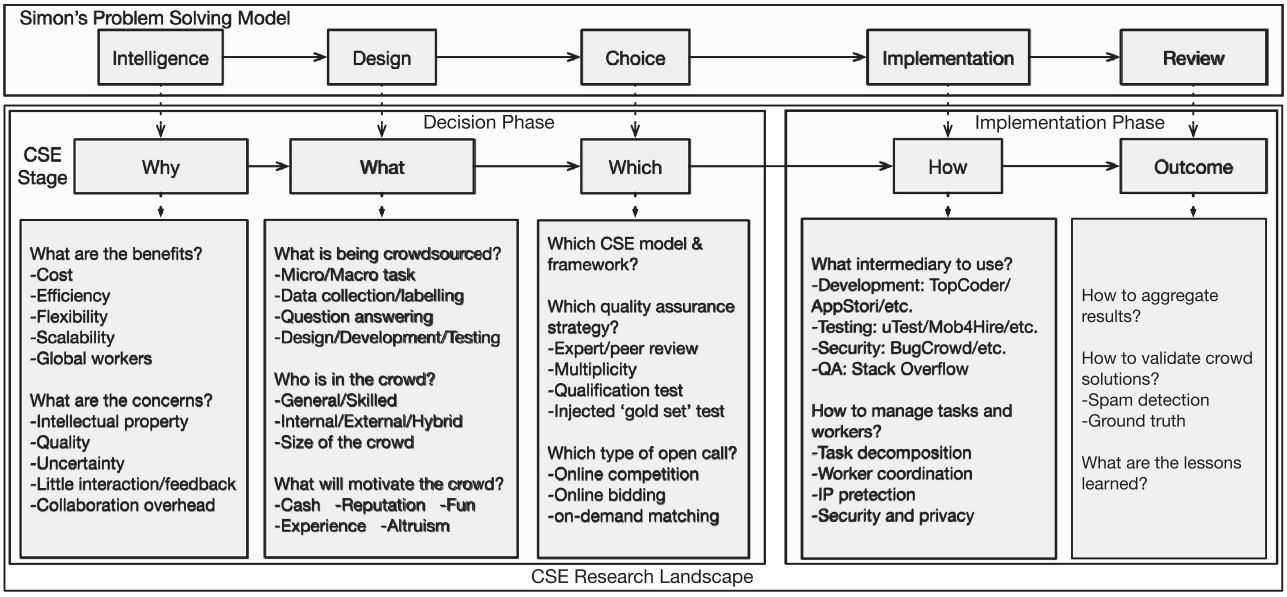

Fig. 9. Problem-solving model of Crowdsourced Software Engineering (adapted from the outsourcing stage model by Dibbern et al. (2004))).

to support activities involved in the planning, analysis, design, implementation and maintenance phases of a software system. For each specific activity in the SDLC, we also provide an example reference on realising this integration. The summary of the papers that apply crowdsourcing to software engineering activities and their mappings to the SDLC are discussed in detail in Section 5.

When adopting crowdsourcing to support a certain activity in an SDLC phase, it requires a general problem-solving process which characterises the steps towards approaching and rationally solving a problem. We illustrate the CSE research landscape from such a general perspective of problem-solving process, as shown in Fig. 9, where important questions and aspects regarding a series of stages in realising CSE are summarised. The stages follow Simon's problem-solving model (Simon, 1960), which consists of two phases: a decision phase and an implementation phase. Since Simon's problem-solving model has been adapted in several ways, each phase may vary in stages. Our decision phase contains the three typical stages: intelligence, design and choice. While the implementation phase includes an implementation stage and a following review stage.

Intelligence (why): In the intelligence stage, the problem is defined and the requester should justify the motivation for adopting CSE. What are the potential advantages and disadvantages? 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.

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 requester should think about, what is being crowdsourced? What is the granularity of the task? Is it a micro-task such as a requirement classification or a macro task such as a software component development? Does the task require expert knowledge such as programming? What are the incentives for a certain type of crowd? What is the expected size of the crowd?

Choice (which): In the choice stage, alternative solutions are evaluated. The output of this stage is a decision that can be implemented. To make such a decision usually requires additional 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.

Implementation (how): The implementation stage is where the decision is finally carried out. Both CSE researchers and practi-

tioners need to deal with a series of questions to implement the crowdsourcing activity. For example, what intermediary platform should be used in order to accomplish the work? Subsequently, how to manage the tasks and workers?

Review (outcome): The final review stage evaluates the implementation's outcome. If the outcome is a success, the product may be further validated to check whether it satisfies users' requirements. If failed, lessons should be learned and a new iteration of the problem-solving process is expected. For CSE, the requester may need to think about how to aggregate and validate the crowd solutions. If the use of crowdsourcing in supporting software engineering failed, what is the barrier to a successful CSE solution?

The above process model may provide guidance to CSE researchers and practitioners for realising CSE. Several of the questions underlying each stage remain open problems, pointing to important research questions. Very few previous studies consider the overall process of CSE at a macro level (such as the one presented in Fig. 9). Instead, where the process studies were conducted, they usually focused on the problems pertained to a specific stage.

In the *why* stage, many previous studies report on the benefits of CSE in terms of low cost, fast delivery and high flexibility (Schneider and Cheung, 2011; Lakhani et al., 2010; Begel et al., 2013; LaToza et al., 2013a; Nebeling et al., 2013b; Ramakrishnan and Srinivasaraghavan, 2014; Li et al., 2015), but there is a lack of comparative studies on validating these benefits by comparing crowdsourcing-based development with traditional in-house or outsourced development. A few other authors, see for example (Stol and Fitzgerald, 2014c; 2014b; LaToza 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 testing[22]. 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) StackOverflow [23] 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

[22] http://www.utest.com/about-us.
[23] http://www.stackoverflow.com.

Table 5
A list of commercial platforms for Crowdsourced Software Engineering.

Platform	Since	URL	Task domain	Open call form
TopCoder	2001	www.topcoder.com	Software Development	Online Competition
GetACoder	2004	www.getacoder.com	Software Development	Online Bidding
AppStori	2013	www.appstori.com	Mobile App Development	Crowd Funding, Online Recruiting
Bountify	2013	www.bountify.co	Small Coding Tasks	Online Competition
uTest	2007	www.utest.com	Software Testing	On-demand Matching, Online Competition
Passbrains	2012	www.passbrains.com	Software Testing	On-demand Matching
99Tests	2010	www.99tests.com	Software Testing	On-demand Matching
TestBirds	2011	www.testbirds.com	Software Testing	On-demand Matching
Testbats	2013	www.testbats.com	Software Testing	On-demand Matching
Pay4Bugs	2009	www.pay4bugs.com	Software Testing	On-demand Matching
CrowdTesters	2014	www.crowdtesters.com.au	Software Testing	On-demand Matching
TestFlight	2010	www.testflightapp.com	Mobile App Testing	On-demand Matching
Mob4hire	2008	www.mob4hire.com	Mobile App Testing	Online Bidding
Testin	2011	www.itestin.com	Mobile App Testing	On-demand Matching
Ce.WooYun	2012	ce.wooyun.org	Software Security Testing	On-demand Matching
Bugcrowd	2012	www.bugcrowd.com	Software Security Testing	Online Competition

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; 2013b; de Souza et al., 2014) and software API documentation (Jiau and Yang, 2012; Parnin et al., 2012).

(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 (2010) summarised three types of crowdsourcing platforms for the software industry: platforms such as Amazon Mechanical Turk[24] that support the use of human knowledge in an inexpensive way; platforms such as TopCoder that support contest-based software development; and platforms like MathWorks[25] 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 and Fitzgerald (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 crowdsource spaceflight software development for the SPHERES Zero Program, supported by NASA, DARPA and Aurora Flight

[24] http://www.mturk.com.
[25] http://www.mathworks.com.

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 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 and Bretschneider (2013); Zogaj et al. (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-Kå reborn 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.

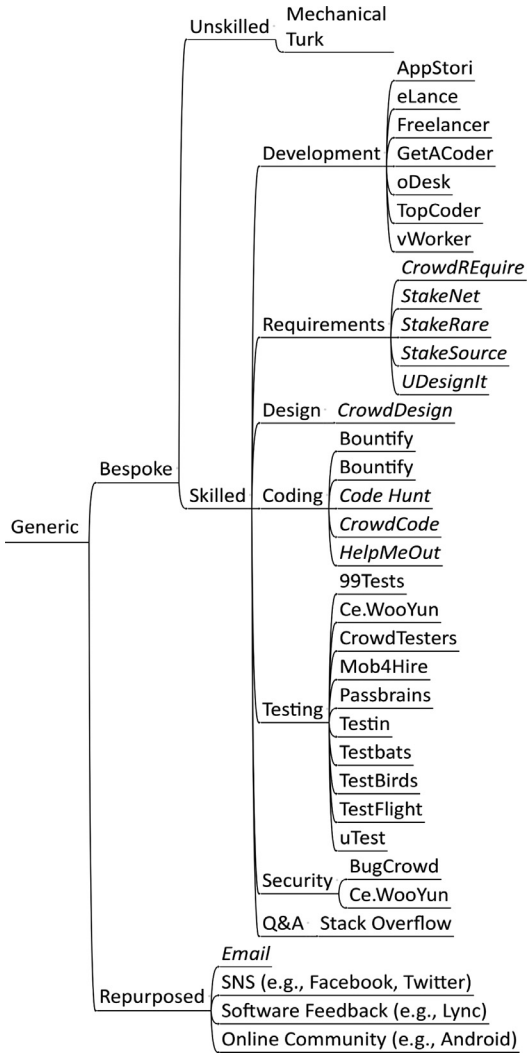

Fig. 10. Scheme of Crowdsourced software engineering platforms. (The *italic text* indicates an experimental/non-commercial platform.)

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 Figs. 12 and 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; 2010a; 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 Ncube, 2013; Snijders and Dalpiaz, 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 6
An overview of the research on Crowdsourced Software Engineering.

SE Phase	SE Task	Why	Bespoke Tool	Stakeholder		
				Requester	Platform	Worker
Requirements	Requirements Acquisition	Cost, User needs, Domain knowledge, Automation, Quality	StakeSource, StakeSource2.0, StakeNet, StakeRare, iRequire	Requirements engineers, Designers, Software teams, Researchers	Email, StakeSource, StakeSource2.0, StakeNet, StakeRare, CrowdREquire, UDesignIt, Bespoke, AOI, AMT	All stakeholders, Users, Undefined crowd
	Requirements Categorisation	User needs	None	Requirements engineers, Designers	Unspecified	Users
Design	User Interface Design	User needs, Quality, Diversity	None	Designers, Non-technical end users	Bespoke, AMT, CrowdDesign, Email	Users
	Architecture Design	Quality, Diversity	None	Researchers	Email	Designers
	Design Revision	Quality, Diversity	None	Researchers	Email	Designers
Coding	IDE Enhancement	Debugging, API aid	BlueFix, Calcite, Example Over-flow, Seahawk, Prompter, SnipMatch	Developers	HelpMeOut, Stack Overflow, oDesk	Developers
	Program Optimisation	Human solutions	None	Developers, Researchers	Bountify, AMT, Software Feedback	Developers, Undefined crowd, Users
	Crowd Programming Support	Automation, Human solutions	CrowdLang, CIDRE, Collabode	Developers, Teachers	Bespoke, AMT, Code Hunt	Users, Developers
Testing	Usability Testing	Cost, Time	CrowdStudy	Testers	CrowdStudy, Bespoke, AMT, CrowdFlower	Users
	Performance Testing	Real-world measure	None	Client companies	Lync	Users
	GUI Testing	Cost, Scalability	None	Testers	AMT	Undefined crowd
	QoE Testing	Cost, Diversity	Quadrant of Euphoria	Researchers	Quadrant of Euphoria, Bespoke, AMT, Microworkers	Undefined crowd
	Test Generation	Human inputs	PAT	Testers, Researchers	Twitter	Undefined crowd
	Oracle Problem	Human solutions, Automation	None	Testers, Researchers	AMT	Qualified / Unqualified crowd
	Crowd Testing Support	Human inputs	CrowdBlaze	Testers, Researchers	Bespoke, AMT, Mobileworks, Email	Undefined crowd
	General Evaluation	User needs, Diversity	None	Researchers	Bespoke, AMT	Users
Verification	Non-expert Verification	Cost, Speed	Verification Games, VeriWeb	Developers, Researchers	Bespoke, AMT, vWorker	Undefined crowd
Evolution	Software Adaptation	User needs, Cost, Diversity, Speed	MoWA, CrowdAdapt	Developers, Designers, Users, Researcher	Bespoke, Facebook, Online community	Users
Maintenance	Software Documentation	Domain knowledge	COFAQ	Developers, Researchers	Q&A, Stack Overflow, SciPy Community	Developers, Researchers
	Software Localisation	Domain knowledge, Cost, Speed	None	Developers, Researchers	AMT	Undefined crowd
Other	Security and Privacy Augmentation	Diversity, Domain knowledge, User needs	Crowdroid, Modding-Interface, CrowdSource, SmartNotes, ProtectMyPrivacy	Developers, Researchers	Android User Community	Users
	End User Support	Domain knowledge	LemonAid	Developers, Researchers	AMT	Users
	Software Ideation	User needs, Open innovation, Recruitment	SAPiens, IdeaMax	Client Companies	Repurposed, Bespoke	Users

Table 7
Reference mapping of the research on Crowdsourced Software Engineering.

SE Phase	SE Task	Reference
Requirements	Requirements Acquisition	(Lim et al., 2010b; 2010a; 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 Ncube, 2013; Snijders and Dalpiaz, 2014; Wang et al., 2014; Nayebi and Ruhe, 2014; Breaux and Schaub, 2014; Hosseini et al., 2015)
	Requirements Categorisation	(Muganda et al., 2012; Nascimento et al., 2012)
Design	User Interface Design	(Bernstein, 2010; Nebeling et al., 2012a; LaToza et al., 2015; Lasecki et al., 2015)
	Architecture Design	(LaToza et al., 2015)
	Design Revision	(LaToza et al., 2015)
Coding	IDE Enhancement	(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; 2013b; Wightman, 2013; Barzilay et al., 2013; Ponzanelli et al., 2014a; 2014b; de Souza et al., 2014; Fast et al., 2014; Chen and Kim, 2015)
	Program Optimisation	(Auler et al., 2014; Cochran et al., 2015)
	Crowd Programming Support	(Goldman et al., 2011; Goldman, 2011; Minder and Bernstein, 2011; 2012; Goldman, 2012; Ball et al., 2014; Xie et al., 2015)
Testing	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)
	QoE Testing	(Chen et al., 2010; Gardlo et al., 2014; Hossfeld et al., 2014a; 2014b)
	Test Generation	(Chen and Kim, 2012; Pham et al., 2013b)
	Oracle Problem Mitigation	(Pastore et al., 2013)
	Crowd Testing Support	(Xue, 2013; Yan et al., 2014; Liang et al., 2014)
	General Evaluation	(Blanco et al., 2011; Sherief et al., 2014; Sherief, 2014)
Verification	Non-expert Verification	(Dietl et al., 2012; Li et al., 2012; Schiller and Ernst, 2012; Schiller, 2014)
Evolution	Software Adaptation	(Bacon et al., 2009; Nebeling and Norrie, 2011a; 2011b; 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; Almaliki et al., 2014; Hamidi et al., 2014)
Maintenance	Software Documentation	(Jiau and Yang, 2012; Parnin et al., 2012; Barzilay et al., 2013; Chen and Zhang, 2014; Pawlik et al., 2015)
	Software Localisation	(Exton et al., 2009; Manzoor, 2011; Gritti, 2012; Mijnhardt, 2013)
Other	Security and Privacy Augmentation	(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)
	End User Support	(Chilana et al., 2012; 2013; Chilana, 2013)
	Software Ideation	(Ebner et al., 2008; Krcmar et al., 2009; Jayakanthan and Sundararajan, 2011a; 2011b)

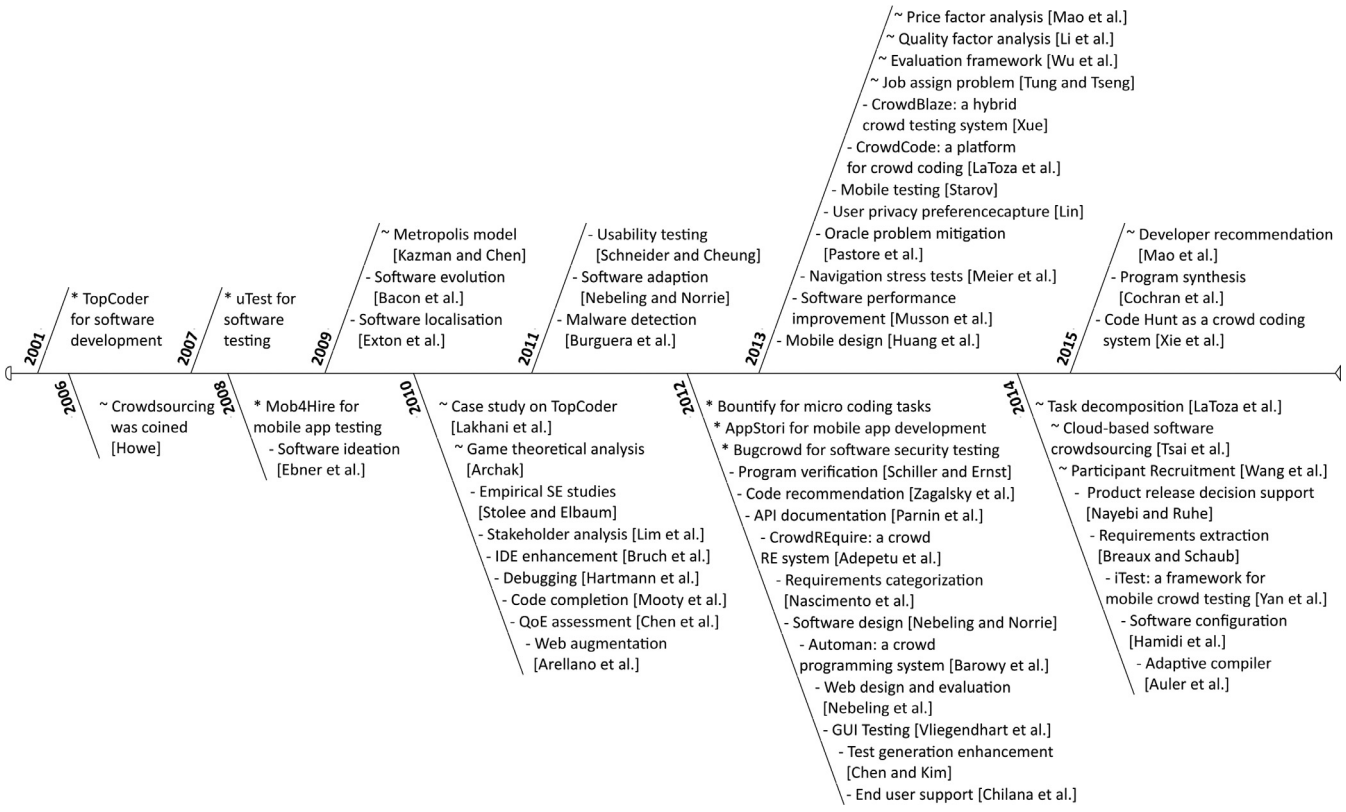

Fig. 11. Timeline for the development of Crowdsourced Software Engineering ('*' indicates the establishment of a platform. ' ' shows the first practical/theoretical study and '-' stands for the first application work).

Table 8
An overview of the Crowdsourcing experiments conducted in the application papers

Phase	SE Task	Platform	Crowd	Size	Subject	Effort	Reward	Result	Reference
Require-ments	Requirements Elicitation	StakeNet	Stakeholders	68	RALIC[a]	-	-	StakeNet can identify stakeholders and their roles with high recall, and can prioritise them accurately.	(Lim et al., 2010a)
	Requirements Elicitation	StakeRare	Stakeholders	87	RALIC	-	-	StakeRare can predict and prioritise stakeholder needs accurately.	(Lim and Finkelstein, 2012)
	Requirements Extraction	AMT	Unskilled	76	-	448 classifications	$0.15 per task	The approach can reduce 60% cost and increase 16% coverage in manual extraction.	(Breaux and Schaub, 2014)
				38	-	135 classifications	$0.15 per task		
				-	-	-	$0.08-$0.10 per task		
Design	Architecture Design	Email	Students	20	a traffic flow simulation program	12.9 hours (average)	$100 each person + 4*$1000 prizes	All participants borrowed others' design ideas and most improved the design quality.	(LaToza et al., 2015)
	User Experience Design			20		21.3 hours (average)			
Coding	IDE Enhancement	StackOverflow	StackOverflow community	-	12 Java, 12 C++ and 11 .NET programming tasks	-	-	For 77.14% of the assessed tasks at least one useful Q&A pair can be recommended.	(de Souza et al., 2014)
	IDE Enhancement - Code Annotation	oDesk	Developers	-	500 Ruby code snippets	500 annotations	-	Among the annotated snippets, 86% correspond to a useful programming task; 96% can be embedded into a standalone function, and 91% do not have another more common form.	(Fast et al., 2014)
	Program Synthesis	Bountify	Developers	5	4 regular expression tasks	wrote 14 regular expression	$10	Consistent prgoram boosts in accuracy can be achieved with modest monetary cost	(Cochran et al., 2015)
		AMT	Unskilled	5		classified strings as valid or invalid	$0.05-$0.25 per task		
Testing	System evaluation	AMT	Unskilled	65	Web query data from multiple semantic search engines	579 HITs[b]	$347.16 ($0.20 per HIT)	Crowdsourced evaluation tasks can be repeated over time and maintain reliable results.	(Blanco et al., 2011)
				69		421 HITs	-		
	Usability Testing	AMT	Unskilled	11	a graduate school's website	11 HITs	$2.92 ($0.15 per HIT)	Reduced cost and time. However quality was worse compared to the testing in a lab setting.	(Liu et al., 2012)
		AMT+CF[c]		44		44 HITs	$347.16 ($0.20 per HIT)		
	Usability Testing	AMT	Unskilled	28	a subpage of a university website	4-5m (average) to answer all 8 questions	-	The crowdsourced usability testing shared similar results with a laboratory setting.	(Meier et al., 2013)
	Usability Testing	CrowdStudy + AMT	Unskilled	93	A news article page	28 custom layouts, 143 ratings and 32 answers	-	The usefulness and the ability to be configured for different scenarios were demonstrated.	(Nebeling et al., 2013b)
				84	Wikipedia website	33 different type tasks	-		
	Performance Testing	Lync	End users	48,000	Lync (Microsoft instant-messaging client)	Usage behaviours	0	The approach had been successfully deployed and had improved development decisions at Microsoft.	(Musson et al., 2013)
	GUI Testing	AMT	Unskilled	100	Tribler	100 assignments	$25	The approach was able to evaluate an experimental user interface feature within a few days at low costs.	(Vliegendhart et al., 2012)
				100		100 assignments	$25		
	GUI Testing	AMT	Unskilled	398	Tribler, KDE, Xfce[d]	700 assignments	$0.10-$0.15 per HIT	The approach is feasible and reliable, although there was a quality issue in continuous testing.	(Dolstra et al., 2013)
	QoE Testing	Quadrant of Euphoria + AMT	Unskilled	-	a three-minute-long speech audio file and a benchmark video clip	2130 runs of experiments	$21.3	With consistency assurance crowdsourcing can yield results as well as laboratory experiments.	(Chen et al., 2010)
	QoE Testing	Microworkers	Unskilled	10,737	3 videos with different content classes	10737 ratings	$0.2625 per rating	The proposed methods represented a step in making crowd-testing sufficiently mature for wide adoption.	(Gardlo et al., 2014)
				1,593		1593 ratings	$0.0834 per rating		
	Test Generation	Twitter	Unskilled	120	the Apache Commons Math and Collections libraries	1 minute per puzzle	0	84 of the top 100 constraint solving puzzles and 24 of the top 100 object mutation puzzles were successfully solved.	(Chen and Kim, 2012)
	Oracle Problem Mitigation	AMT	Unqualified	-	the Java Stack class	200 assignments	$0.15-$0.20 per assignment	CrowdOracles is a promising solution to mitigate the oracle problem, however geting useful results from an untrained crowd is difficult.	(Pastore et al., 2013)
		AMT	Qualified	-	the Java Stack class, the Java libraries Trove4J and j8583,	500 assignments			
	Crowd Testing Support	Mobileworks + Email	End users	75	8 popular Android apps from Google Play	-	-	Statically Aided Interactive Dynamic Analysis consistently obtained greater coverage than other techniques.	(Xue, 2013)
Verification	Non-expert	vWorker	Developers	14	StackAr (an implementation of an array-based stack in Java)	3 hours (average)	$6-$22 per hour	VeriWeb can save time and money with contracted workers.	(Schiller and Ernst, 2012)
	Verification	AMT	Unskilled	< 10		-	> 0.25 per HIT	Current ad-hoc labours are not well-suited for verification.	

[a] RALIC is the acronym of 'Replacement Access, Library and ID Card', which is an access control system developed by University College London.
[b] A Human Intelligent Task (HIT) is a single, self-contained task that workers can perform.
[c] AMT and CF refer to Amazon Mechanical Turk and CrowdFlower, respectively.
[d] Tribler is a peer-to-peer file-sharing client. KDE and Xfce are two desktop environments for Unix-like platforms.

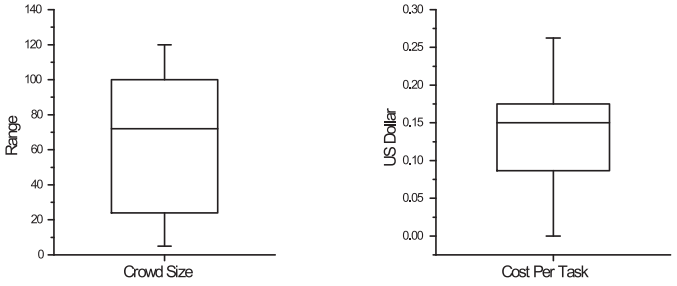

Fig. 12. Crowd size and cost per task in the surveyed studies.

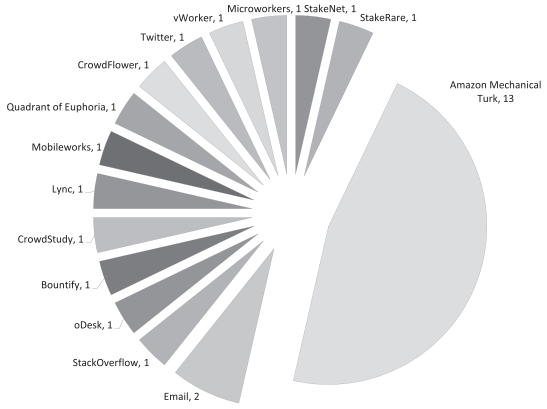

Fig. 13. Platforms used in the surveyed studies.

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 system of systems projects. The tool is publicly available online[26].

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 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.

[26] http://www.cs.ucl.ac.uk/research/StakeSource.

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. By carefully designing the coding duels, the platform can serve for software construction purpose by leveraging the best solutions from the crowd. Also, by recording the crowd developers' duel solving process, the multiple attempts with evolving code versions can serve for education purpose.

(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; 2013b; Wightman, 2013; Barzilay et al., 2013; Ponzanelli et al., 2014a; 2014b; 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 reparation 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 construc-

tion examples by collecting code from the web. According to a reported user study, this plugin can help developers to increase their completion rate by 40%.

Example Overflow (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 function 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; 2013b) 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.

de 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, 2011. 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[27] 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[28] and Firefox's[29] built-in telemetries (performance testing frameworks) (Akhawe 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).

Vliegendhart 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 Mob4Hire: 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; 2014b).

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 pre-defined 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

[27] http://office.microsoft.com/lync.
[28] http://www.chromium.org/developers/telemetry.
[29] http://telemetry.mozilla.org.

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; Jiau 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., 2015; 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. (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., 2015).

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. (2015) 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 (Ebner et al., 2008; Krcmar et al., 2009; Jayakanthan and Sundararajan, 2011a; 2011b).

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 et al. (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. Papamartzivanos 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); Chilana (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; Stolee 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.

Table 9
Crowdsourced evaluation for software engineering research.

Reference	SE Task	Size	Crowd	Platform	Effort
(Fry and Weimer, 2010)	Fault localisation	65	Developers	AMT	1830 judgements
(Stolee and Elbaum, 2010)	Code smell impact evaluation	50	End users programmers	AMT	160 HIT responses
(Hartmann et al., 2010)	IDE enhancement	13	Students	Workshop	39 person-hours
(Fry et al., 2012)	Patch maintainability	157	Developers	Campus, AMT	2100 judgements
(Afshan et al., 2013)	Readability evaluation on test input strings	250	Developers	CrowdFlower	8 questions per task, 250 responses
(Stolee and Elbaum, 2013)	Code smell impact evaluation	61	End user programmers	AMT	366 task responses
(Stolee et al., 2014)	Survey on code search habits	99	Developers	Campus, AMT	10 questions per survey
(Fast et al., 2014)	Code annotation	-	Developers	oDesk	500 code snippets' evaluation

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., 2013b; Hu and Wu, 2014; Xu and Wang, 2014b). 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.

6.2. Task decomposition

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 *VeriWeb*, which can decompose the verifiable specifications task into manageable sub-problems.

6.3. Planning and scheduling

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 ε-first algorithm works in two stages: First, it explores the estimation of workers' quality by using part of the total budget; Second, it exploits the estimates of workers' quality to maximise the overall utility with the remaining budget. The evaluation of the proposed algorithm was based on empirical data collected from oDesk. The results indicated that the algorithm was able to outperform related state-of-the-art crowdsourcing algorithms by up to 300%.

Tung and Tseng (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 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 Itkonen (2013) studied how the crowd size and allocated time can affect the performance of software testing. Their results, conduced on 130 students, indicated that multiple crowd workers under time pressure had 71% higher effectiveness (measured by 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.

6.4. Motivation and remuneration

Motivation is viewed as a critical factor for the success of a software project (Sharp et al., 2009; Beecham et al., 2008; Boehm et al., 1981). For crowdsourced software projects, developers without proper motivation may not be able to make consistent contributions, while inappropriate remuneration may lead to low capital efficiency or task starvation. Varshney (2012) demonstrated that player motivation is essential for driving participation and ensuring a reliable delivery platform. Based on a study from IBM's internal crowdsourced software development system — *Liquid*, several intrinsic, extrinsic, and social motivation factors were identified. Developer participation was found to follow a power-law distribution. A momentum-based generative model and a thermodynamic interpretation were used to describe the observed participation phenomena.

Mao et al. (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.

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

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

This blurring of the distinction between traditional and 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 Fig. 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

Abran, A., Moore, J.W., et al., 2004. Guide to the Software Engineering Body of Knowledge (SWEBOK®). 2004 Version, IEEE CS Professional Practices Committee.

Adamopoulos, K., Harman, M., Hierons, R.M., 2004. Mutation testing using genetic algorithms: A co-evolution approach. In: Proc. 6th Annual Genetic and Evolutionary Computation Conference, pp. 1338–1349.

Adepetu, A., Ahmed, K., Abd, Y.A., 2012. CrowdREquire: A Requirements Engineering Crowdsourcing Platform. Technical Report. AAAI.

Afshan, S., McMinn, P., Stevenson, M., 2013. Evolving readable string test inputs using a natural language model to reduce human oracle cost. In: Proc. 6th IEEE International Conference on Software Testing, Verification and Validation, pp. 352–361.

Agarwal, Y., Hall, M., 2013. ProtectMyPrivacy: Detecting and mitigating privacy leaks on iOS devices using crowdsourcing. In: Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services, pp. 97–110.

Akhawe, D., Felt, A.P., 2013. Alice in warningland: A large-scale field study of browser security warning effectiveness. In: Proc. 22Nd USENIX Conference on Security, pp. 257–272.

Akiki, P., Bandara, A., Yu, Y., 2013. Crowdsourcing user interface adaptations for minimizing the bloat in enterprise applications. In: Proc. 5th ACM SIGCHI symposium on Engineering interactive computing systems, pp. 121–126.

Ali, R., Solis, C., Omoronyia, I., Salehie, M., Nuseibeh, B., 2012. Social adaptation: when software gives users a voice. In: Proc. 7th International Conference Evaluation of Novel Approaches to Software Engineering.

Ali, R., Solis, C., Salehie, M., Omoronyia, I., Nuseibeh, B., Maalej, W., 2011. Social sensing: When users become monitors. In: Proc. 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, pp. 476–479.

Allahbakhsh, M., Benatallah, B., Ignjatovic, A., Motahari-Nezhad, H., Bertino, E., Dustdar, S., 2013. Quality control in crowdsourcing systems: Issues and directions. IEEE Internet Comput. 17 (2), 76–81.

Almaliki, M., Ncube, C., Ali, R., 2014. The design of adaptive acquisition of users feedback: An empirical study. In: Proc. 9th International Conference on Research Challenges in Information Science.

Alonso, O., Rose, D.E., Stewart, B., 2008. Crowdsourcing for relevance evaluation. In: ACM SigIR Forum, Vol. 42. ACM, pp. 9–15.

Amann, S., Proksch, S., Mezini, M., 2014. Method-call recommendations from implicit developer feedback. In: Proc. 1st International Workshop on CrowdSourcing in Software Engineering, pp. 5–6.

Aparicio, M., Costa, C.J., Braga, A.S., 2012. Proposing a system to support crowdsourcing. In: Proc. 2012 Workshop on Open Source and Design of Communication, pp. 13–17.

Archak, N., 2010. Money, glory and cheap talk: analyzing strategic behavior of contestants in simultaneous crowdsourcing contests on TopCoder.com. In: Proc. 19th international conference on World wide web, pp. 21–30.

Arcuri, A., White, D.R., Clark, J.A., Yao, X., 2008. Multi-objective improvement of software using co-evolution and smart seeding. In: Proc. 7th International Conference on Simulated Evolution and Learning, 5361, pp. 61–70.

Arellano, C., Díaz, O., Iturrioz, J., 2010. Crowdsourced web augmentation : a security model. In: Proc. 11 International Conference on Web Information Systems Engineering, pp. 294–307.

Auler, R., Borin, E., Halleux, P.D., 2014. Addressing JavaScript JIT engines performance quirks : a crowdsourced adaptive compiler. In: Proc. 23rd International Conference on Compiler Construction, pp. 218–237.

Bacchelli, A., Ponzanelli, L., Lanza, M., 2012. Harnessing Stack Overflow for the IDE. In: Proc. 3rd International Workshop on Recommendation Systems for Software Engineering, pp. 26–30.

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

Ball, T., Burckhardt, S., de Halleux, J., Moskal, M., Tillmann, N., 2014. Beyond Open Source: The TouchDevelop Cloud-based Integrated Development and Runtime Environment. Technical Report.

Barr, E.T., Harman, M., McMinn, P., Shahbaz, M., Yoo, S., 2015. The oracle problem in software testing: a survey. IEEE Trans. Software Eng. 41 (5), 507–525.

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

Beecham, S., Baddoo, N., Hall, T., Robinson, H., Sharp, H., 2008. Motivation in software engineering: a systematic literature review. Inf. Software Technol. 50 (9), 860–878.

Begel, A., Bosch, J., Storey, M.-A., 2013. Social networking meets software development: Perspectives from GitHub, MSDN, Stack Exchange, and TopCoder. IEEE Software 30 (1), 52–66.

Begel, A., DeLine, R., Zimmermann, T., 2010. Social media for software engineering. In: Proc. FSE/SDP Workshop on Future of Software Engineering Research, pp. 33–38.

Bergvall-Kå reborn, B., Howcroft, D., 2013. The apple business model: crowdsourcing mobile applications. Accounting Forum 37 (4), 280–289.

Bernstein, M.S., 2010. Crowd-powered interfaces. In: Proc. 23nd annual ACM symposium on User interface software and technology, pp. 347–350.

Bishop, J., Horspool, R.N., Xie, T., Tillmann, N., de Halleux, J., 2015. Code hunt: experience with coding contests at scale. In: Proc. 37th International Conference on Software Engineering - JSEET.

Blanco, R., Halpin, H., Herzig, D.M., Mika, P., Pound, J., Thompson, H.S., Tran Duc, T., 2011. Repeatable and reliable search system evaluation using crowdsourcing. In: Proc. 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 923–932.

Boehm, B.W., et al., 1981. Software engineering economics, Vol. 197. Prentice-hall Englewood Cliffs (NJ).

Bozkurt, M., Harman, M., (2011). Automatically generating realistic test input from web services. Proc. 6th IEEE International Symposium on Service Oriented System Engineering.

Brabham, D.C., 2008. Crowdsourcing as a model for problem solving an introduction and cases. Convergence 14 (1), 75–90.

Brabham, D.C., Sanchez, T.W., Bartholomew, K., 2009. Crowdsourcing public participation in transit planning: preliminary results from the next stop design case. Transportation Research Board.

Breaux, T.D., Schaub, F., 2014. Scaling requirements extraction to the crowd: Experiments with privacy policies. In: Proc. 22nd IEEE International Requirements Engineering Conference, pp. 163–172.

Bruce, B., Petke, J., Harman, M., 2015. Reducing energy consumption using genetic improvement. In: Proc. 17th Annual Genetic and Evolutionary Computation Conference.

Bruch, M., 2012. IDE 2.0: Leveraging the Wisdom of the Software Engineering Crowds. Technische Universität Darmstadt.

Bruch, M., Bodden, E., Monperrus, M., Mezini, M., 2010. IDE 2.0: Collective intelligence in software development. In: Proc. FSE/SDP Workshop on Future of Software Engineering Research, pp. 53–58.

Burguera, I., Zurutuza, U., Nadjm-Tehrani, S., 2011. Crowdroid: Behavior-based malware detection system for Android. In: Proc. 1st ACM workshop on Security and Privacy in Smartphones and Mobile Devices, pp. 15–26.

Challiol, C., Firmenich, S., Bosetti, G.A., Gordillo, S.E., Rossi, G., 2013. Crowdsourcing mobile web applications. In: Proc. ICWE 2013 Workshops, pp. 223–237.

Chatfield, A.T., Brajawidagda, U., 2014. Crowdsourcing hazardous weather reports from citizens via twittersphere under the short warning lead times of EF5 intensity tornado conditions. In: Proc. 47th Hawaii International Conference on System Sciences. IEEE, pp. 2231–2241.

Chen, C., Zhang, K., 2014. Who asked what: Integrating crowdsourced FAQs into API documentation. In: Proc. 36th International Conference on Software Engineering (ICSE Companion), pp. 456–459.

Chen, F., Kim, S., 2015. Crowd debugging. In: Proc. 10th Joint Meeting on Foundations of Software Engineering, pp. 320–332.

Chen, K.-t., Chang, C.-j., Sinica, A., Wu, C.-c., Chang, Y.-c., Lei, C.-l., 2010. Quadrant of Euphoria: a crowdsourcing platform for QoE assessment. IEEE Netw. (April) 28–35.

Chen, N., Kim, S., 2012. Puzzle-based automatic testing: bringing humans into the loop by solving puzzles. In: Proc. 27th IEEE/ACM International Conference on Automated Software Engineering, pp. 140–149.

Chen, N., Lin, J., Hoi, S.C., Xiao, X., Zhang, B., 2014. AR-Miner: mining informative reviews for developers from mobile app marketplace. In: Proc. 36th International Conference on Software Engineering, pp. 767–778.

Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (Eds.), 2008. Software engineering for self-adaptive systems (Dagstuhl Seminar). Dagstuhl Seminar Proceedings 08031.

Chilana, P.K., 2013. Supporting users after software deployment through selection-based crowdsourced contextual help. University of Washington.

Chilana, P.K., Ko, A.J., Wobbrock, J.O., 2012. LemonAid: selection-based crowdsourced contextual help for web applications. In: Proc. SIGCHI Conference on Human Factors in Computing Systems, pp. 1549–1558.

Chilana, P.K., Ko, A.J., Wobbrock, J.O., Grossman, T., 2013. A multi-site field study of crowdsourced contextual help : usage and perspectives of end users and software teams. In: Proc. 31st Annual CHI Conference on Human Factors in Computing Systems.

Cleland-Huang, J., Shin, Y., Keenan, E., Czauderna, A., Leach, G., Moritz, E., Gethers, M., Poshyvanyk, D., Hayes, J.H., Li, W., 2012. Toward actionable, broadly accessible contests in software engineering. In: Proc. 34th International Conference on Software Engineering, pp. 1329–1332.

Cochran, R.A., D'Antoni, L., Livshits, B., Molnar, D., Veanes, M., 2015. Program boosting: program synthesis via crowd-sourcing. In: Proc. 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 677–688.

Cook, C., Visconti, M., 1994. Documentation is important. CrossTalk 7 (11), 26–30.

Cooper, S., Khatib, F., Treuille, A., Barbero, J., Lee, J., Beenen, M., Leaver-Fay, A., Baker, D., Popović, Z., et al., 2010. Predicting protein structures with a multiplayer online game. Nature 466 (7307), 756–760.

Cox, L.P., 2011. Truth in crowdsourcing. IEEE J. Secur. Privacy 9 (5), 74–76.

Dibbern, J., Goles, T., Hirschheim, R., Jayatilaka, B., 2004. Information systems outsourcing: a survey and analysis of the literature. Data Base Adv. Inf. Syst. 35 (4), 6–102.

Dietl, W., Dietzel, S., Ernst, M.D., Mote, N., Walker, B., Cooper, S., Pavlik, T., Popović, Z., 2012. Verification games: Making verification fun. In: Proc. 14th Workshop on Formal Techniques for Java-like Programs, pp. 42–49.

Dolstra, E., Vliegendhart, R., Pouwelse, J., 2013. Crowdsourcing GUI tests. In: Proc. 6th IEEE International Conference on Software Testing, Verification and Validation, pp. 332–341.

Ebner, W., Leimeister, M., Bretschneider, U., Krcmar, H., 2008. Leveraging the wisdom of crowds: Designing an IT-supported ideas competition for an ERP software company. In: Proc. 41st Annual Hawaii International Conference on System Sciences. 417–417

Ernst, M.D., Popović, Z., 2012. Crowd-sourced program verification. Technical Report. University OF Washington.

Esselink, B., 2000. A practical guide to localization, Vol. 4. John Benjamins Publishing.

Estellés-Arolas, E., González-Ladrón-De-Guevara, F., 2012. Towards an integrated crowdsourcing definition. J. Inf. Sci. 38 (2), 189–200.

Exton, C., Wasala, A., Buckley, J., Schäler, R., 2009. Micro crowdsourcing: A new model for software localisation. Localisation Focus 8 (1), 81–89.

Farrell, J., Rabin, M., 1996. Cheap talk. The Journal of Economic Perspectives 10 (3), 103–118.

Fast, E., Steffee, D., Wang, L., Brandt, J.R., Bernstein, M.S., 2014. Emergent, crowd-scale programming practice in the IDE. In: Proc. 32nd annual ACM conference on Human factors in Computing Systems, pp. 2491–2500.

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

Fried, D., 2010. Crowdsourcing in the software development industry. Nexus of Entrepreneurship and Technology Initiative.

Fry, Z.P., Landau, B., Weimer, W., 2012. A human study of patch maintainability. In: Proc. 2012 International Symposium on Software Testing and Analysis, pp. 177–187.

Fry, Z.P., Weimer, W., 2010. A human study of fault localization accuracy. In: Proc. 26th IEEE International Conference on Software Maintenance.

Gardlo, B., Egger, S., Seufert, M., Schatz, R., 2014. Crowdsourcing 2.0: Enhancing execution speed and reliability of web-based QoE testing. In: Proc. 2014 IEEE International Conference on Communications, pp. 1070–1075.

Goldman, M., 2011. Role-based interfaces for collaborative software development. In: Proc. 24th Annual ACM Symposium Adjunct on User Interface Software and Technology, pp. 23–26.

Goldman, M., 2012. Software development with real-time collaborative editing. Massachusetts Institute of Technology.

Goldman, M., Little, G., Miller, R.C., 2011. Real-time collaborative coding in a web IDE. In: Proc. 24th annual ACM symposium on User interface software and technology, pp. 155–164.

Gomide, V.H.M., Valle, P.A., Ferreira, J.O., Barbosa, J.R.G., da Rocha, A.F., Barbosa, T.M.G.d.A., 2014. Affective crowdsourcing applied to usability testing. Int. J. Comput. Sci. Inf. Technol. 5 (1), 575–579.

Greenwood, P., Rashid, A., Walkerdine, J., 2012. UDesignIt: Towards social media for community-driven design. Proc. 34th International Conference on Software Engineering 1321–1324.

Gritti, A., 2012. Crowd outsourcing for software localization. Universitat Politécnica de Catalunya.

Guzman, E., Maalej, W., 2014. How do users like this feature? A fine grained sentiment analysis of app reviews. In: Proc. 22nd International Conference on Requirements Engineering, pp. 153–162.

Hamidi, S., Andritsos, P., Liaskos, S., 2014. Constructing adaptive configuration dialogs using crowd data. In: Proc. 29th ACM/IEEE International Conference on Automated Software Engineering, pp. 485–490.

Harman, M., Jia, Y., Langdon, W.B., Petke, J., Moghadam, I.H., Yoo, S., Wu, F., 2014. Genetic improvement for adaptive software engineering (keynote). In: Proc. 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, pp. 1–4.

Harman, M., Jia, Y., Zhang, Y., 2012. App store mining and analysis: MSR for App Stores. In: Proc. 9th Working Conference on Mining Software Repositories, pp. 108–111.

Harman, M., Jones, B.F., 2001. Search-based software engineering. Inf. Software Technol. 43 (14), 833–839.

Harman, M., Langdon, W.B., Jia, Y., 2014. Babel pidgin: SBSE can grow and graft entirely new functionality into a real world system. In: Proc. 6th Symposium on Search Based Software Engineering, pp. 247–252.

Hartmann, B., Macdougall, D., Brandt, J., Klemmer, S.R., 2010. What would other programmers do? suggesting solutions to error messages. In: Proc. 28th ACM Conference on Human Factors in Computing Systems, pp. 1019–1028.

He, H., Ma, Z., Chen, H., Shao, W., 2014. How the crowd impacts commercial applications: A user-oriented approach. In: Proc. 1st International Workshop on Crowd-based Software Development Methods and Technologies, pp. 1–6.

Hetmank, L., 2013. Components and functions of crowdsourcing systems-a systematic literature review.. Wirtschaftsinformatik 4.

Hosseini, M., Phalp, K., Taylor, J., Ali, R., 2013. Towards crowdsourcing for requirements engineering. In: Proc. 20th International working conference on Requirements engineering: foundation for software quality (Empirical Track).

Hosseini, M., Shahri, A., Phalp, K., Taylor, J., Ali, R., Dalpiaz, F., 2015. Configuring crowdsourcing for requirements elicitation. In: Proc. 9th International Conference on Research Challenges in Information Science.

Hossfeld, T., Keimel, C., Hirth, M., Gardlo, B., Habigt, J., Diepold, K., 2014. Best practices for QoE crowdtesting : QoE assessment with crowdsourcing. IEEE Transactions on Multimedia 16 (2), 541–558.

Hossfeld, T., Keimel, C., Timmerer, C., 2014. Crowdsourcing quality-of-experience assessments. Computer 98–102.

Howe, J., 2006a. Crowdsourcing: a definition. http://crowdsourcing.typepad.com/cs/2006/06/crowdsourcing_a.html.

Howe, J., 2006. The rise of crowdsourcing. Wired magazine 14 (6), 1–4.

Hu, Z., Wu, W., 2014. A game theoretic model of software crowdsourcing. In: Proc. 8th IEEE International Symposium on Service Oriented System Engineering, pp. 446–453.

Huang, Y.-C., Wang, C.-I., Hsu, J., 2013. Leveraging the crowd for creating wireframe-based exploration of mobile design pattern gallery. In: Proc. Companion Publication of the 2013 International Conference on Intelligent User Interfaces Companion, pp. 17–20.

Hughes, J. M., 2010. Systems and methods for software development. US Patent 7778866 B2.

Huhns, M.N., Li, W., Tsai, W.-T., 2013. Cloud-based software crowdsourcing (dagstuhl seminar 13362). Dagstuhl Reports 3 (9), 34–58.

Ipeirotis, P.G., Provost, F., Wang, J., 2010. Quality management on amazon mechanical turk. In: Proc. 2010 ACM SIGKDD Workshop on Human Computation, pp. 64–67.

Ismail, Q., Ahmed, T., Kapadia, A., Reiter, M.K., 2015. Crowdsourced exploration of security configurations. In: Proc. 33rd Annual ACM Conference on Human Factors in Computing Systems. ACM, pp. 467–476.

James, S., 2004. The wisdom of the crowds. New York: Randome House.

Jayakanthan, R., Sundararajan, D., 2011. Enterprise crowdsourcing solution for software development in an outsourcing organization. In: Proc. 11th International Conference on Web Engineering, pp. 177–180.

Jayakanthan, R., Sundararajan, D., 2011. Enterprise crowdsourcing solutions for software development and ideation. In: Proc. 2nd international workshop on Ubiquitous crowdsouring, pp. 25–28.

Jiau, H.C., Yang, F.-P., 2012. Facing up to the inequality of crowdsourced API documentation. ACM SIGSOFT Software Eng. Notes 37 (1), 1–9.

Johnson, R., 2014. Natural products: Crowdsourcing drug discovery. Nature Chem. 6 (2), 87–87.

Kajko-Mattsson, M., 2005. A survey of documentation practice within corrective maintenance. Empirical Software Eng. 10 (1), 31–55.

Kallenbach, M., 2011. HelpMeOut-Crowdsourcing suggestions to programming problems for dynamic, interpreted languages. RWTH Aachen University.

Kazman, R., Chen, H.-M., 2009. The metropolis model a new logic for development of crowdsourced systems. Commun. ACM 52 (7), 76–84.

Kazman, R., Chen, H.-M., 2010. The metropolis model and its implications for the engineering of software ecosystems. In: Proc. 2010 FSE/SDP workshop on Future of software engineering research, pp. 187–190.

Khatib, F., DiMaio, F., Cooper, S., Kazmierczyk, M., Gilski, M., Krzywda, S., Zabranska, H., Pichova, I., Thompson, J., Popović, Z., et al., 2011. Crystal structure of a monomeric retroviral protease solved by protein folding game players. Nature Struct. Molecular Biol. 18 (10), 1175–1177.

Kittur, A., Smus, B., Khamkar, S., Kraut, R.E., 2011. CrowdForge: Crowdsourcing complex work. In: Proc. 24th Annual ACM Symposium on User Interface Software and Technology, pp. 43–52.

Kogut, B., Metiu, A., 2001. Open-source software development and distributed innovation. Oxford Rev. Economic Policy 17 (2), 248–264.

Krcmar, H., Bretschneider, U., Huber, M., Leimeister, J.M., 2009. Leveraging crowdsourcing: Activation-supporting components for IT-based ideas competition. J. Manage. Inf. Syst. 26 (1), 197–224.

Lakhani, K.R., Boudreau, K.J., Loh, P.-R., Backstrom, L., Baldwin, C., Lonstein, E., Lydon, M., MacCormack, A., Arnaout, R.A., Guinan, E.C., 2013. Prize-based contests can provide solutions to computational biology problems. Nature Biotechnol. 31 (2), 108–111.

Lakhani, K.R., Garvin, D.A., Lonstein, E., 2010. TopCoder(A): developing software through crowdsourcing. Harvard Business School Case.

Lakhotia, K., McMinn, P., Harman, M., 2009. Automated test data generation for coverage: Haven't we solved this problem yet? In: Proc. 4th Testing Academia and Industry Conference — Practice And Research Techniques, pp. 95–104.

Langdon, W.B., Harman, M., 2015. Optimising existing software with genetic programming. IEEE Trans. Evol. Comput. 19 (1), 118–135.

Langdon, W.B., Lam, B., Petke, J., Harman, M., 2015. Improving CUDA DNA analysis software with genetic programming. In: Proc. 17th Annual Genetic and Evolutionary Computation Conference.

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

LaToza, T.D., Ben Towne, W., van der Hoek, A., Herbsleb, J.D., 2013. Crowd development. In: Proc. 6th International Workshop on Cooperative and Human Aspects of Software Engineering, pp. 85–88.

LaToza, T.D., Chen, M., Jiang, L., Zhao, M., Hoek, A.V.D., 2015. Borrowing from the crowd : a study of recombination in software design competitions. In: Proc. 37nd ACM/IEEE International Conference on Software Engineering.

LaToza, T.D., Chiquillo, E., Ben Towne, W., Adriano, C., van der Hoek, A., 2013. CrowdCode - crowd development. CrowdConf 2013.

LaToza, T.D., van der Hoek, A., 2015. A vision of crowd development. In: Proc. 37th International Conference on Software Engineering, NIER Track.

LaToza, T.D., Towne, W.B., Adriano, C.M., van der Hoek, A., 2014. Microtask programming: building software with a crowd. In: Proc. 27th annual ACM symposium on User interface software and technology, pp. 43–54.

LaToza, T.D., Towne, W.B., Hoek, A.V.D., 2014. Harnessing the crowd : decontextualizing software work. In: Proc. 1st International Workshop on Context in Software Development Workshop, pp. 2–3.

Le Goues, C., Forrest, S., Weimer, W., 2013. Current challenges in automatic software repair. Software Qual. J. 21 (3), 421–443.

Le Goues, C., Nguyen, T., Forrest, S., Weimer, W., 2012. Genprog: a generic method for automatic software repair. IEEE Trans. Software Eng. 38 (1), 54–72.

Lease, M., Yilmaz, E., 2012. Crowdsourcing for information retrieval. In: ACM SIGIR Forum, Vol. 45. ACM, pp. 66–75.

Leone, S., 2011. Information Components as a Basis for Crowdsourced Information System Development. Swiss Federal Institute of Technology in Zurich.

Li, D., Tran, A.H., Halfond, W.G.J., 2014. Making web applications more energy efficient for OLED smartphones. In: Proc. 36th International Conference on Software Engineering, pp. 527–538.

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

Li, W., Huhns, M.N., Tsai, W.-T., Wu, W., 2015. Crowdsourcing: Cloud-Based Software Development. Springer.

Li, W., Seshia, S., Jha, S., 2012. CrowdMine: Towards crowdsourced human-assisted verification. In: Proc. 49th Annual Design Automation Conference, pp. 2–3.

Liang, C.-J.M., Lane, N.D., Brouwers, N., Zhang, L., Karlsson, B.F., Liu, H., Liu, Y., Tang, J., Shan, X., Chandra, R., Zhao, F., 2014. Caiipa : Automated large-scale mobile app testing through contextual fuzzing. In: Proc. 20th Annual International Conference on Mobile Computing and Networking.

Lim, S., Quercia, D., Finkelstein, A., 2010. Stakenet: Using social networks to analyse the stakeholders of large-scale software projects. In: Proc. 32nd ACM/IEEE International Conference on Software Engineering, 2010.

Lim, S.L., 2010. Social Networks and Collaborative Filtering for Large-Scale Requirements Elicitation. University of New South Wales.

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

Lim, S.L., Finkelstein, A., 2012. StakeRare: using social networks and collaborative filtering for large-scale requirements elicitation. IEEE Trans. Software Eng. 38 (3), 707–735.

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

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

Lin, J., 2013. Understanding and Capturing People's Mobile App Privacy Preferences. Carnegie Mellon University.

Lin, J., Amini, S., Hong, J.I., Sadeh, N., Lindqvist, J., Zhang, J., 2012. Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing. In: Proc. 2012 ACM Conference on Ubiquitous Computing, pp. 501–510.

Liu, D., Bias, R.G., Lease, M., Kuipers, R., 2012. Crowdsourcing for usability testing. In: Proc. American Society for Information Science and Technology, 49, pp. 1–10.

Lydon, M., 2012. Topcoder overview. http://www.nasa.gov/pdf/651447main_TopCoder_Mike_D1_830am.pdf. Accessed: 2015-11-23.

Maalej, W., Pagano, D., 2011. On the socialness of software. In: Proc. 9th International Conference on Dependable, Autonomic and Secure Computing, pp. 864–871.

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

Manotas, I., Pollock, L., Clause, J., 2014. SEEDS: A software engineer's energy-optimization decision support framework. In: Proc. 36th International Conference on Software Engineering, pp. 503–514.

Mäntylä, M.V., Itkonen, J., 2013. More testers - the effect of crowd size and time restriction in software testing. Information and Software Technology 55 (6), 986–1003.

Manzoor, J., 2011. A crowdsourcing framework for software localization. KTH Royal Institute of Technology.

Mao, K., Yang, Y., Li, M., Harman, M., 2013. Pricing Crowdsourcing Based Software Development Tasks. In: Proc. 2013 International Conference on Software Engineering (NIER Track), pp. 1205–1208.

Mao, K., Yang, Y., Wang, Q., Jia, Y., Harman, M., 2015. Developer recommendation for crowdsourced software development tasks. In: Proc. 9th IEEE International Symposium on Service-Oriented System Engineering, pp. 347–356.

Martin, S.F., Falkenberg, H., Dyrlund, T.F., Khoudoli, G.A., Mageean, C.J., Linding, R., 2013. PROTEINCHALLENGE: crowd sourcing in proteomics analysis and software development.. J. Proteomics 88. 41–6
Massolution, 2012. Crowdsourcing industry report. http://www.crowdsourcing.org/editorial/enterprise-crowdsourcing-trends-infographic/18725. Accessed: 2015-03-01.
Meier, F., Bazo, A., Burghardt, M., Wolff, C., 2013. Evaluating a web-based tool for crowdsourced navigation stress tests. In: Proc. 2nd International Conference on Design, User Experience, and Usability: Web, Mobile, and Product Design, pp. 248–256.
Memon, A., Banerjee, I., Nagarajan, A., 2003. GUI ripping: Reverse engineering of graphical user interfaces for testing. In: Proc. 10th Working Conference on Reverse Engineering, pp. 260–269.
Mijnhardt, A., 2013. Crowdsourcing for enterprise software localization. Utrecht University Master thesis.
Minder, P., Bernstein, A., 2011. CrowdLang - First steps towards programmable human computers for general computation. In: Proc. 3rd Human Computation Workshop, pp. 103–108.
Minder, P., Bernstein, A., 2012. CrowdLang: A programming language for the systematic exploration of human computation systems. In: Proc. 4th International Conference on Social Informatics. Springer, Lausanne.
Misra, A., Gooze, A., Watkins, K., Asad, M., Le Dantec, C.A., 2014. Crowdsourcing and its application to transportation data collection and management. Trans. Res. Record 2414 (1), 1–8.
Mooty, M., Faulring, A., Stylos, J., Myers, B.a., 2010. Calcite: Completing code completion for constructors using crowds. In: Proc. 2010 IEEE Symposium on Visual Languages and Human-Centric Computing, pp. 15–22.
Muganda, N., Asmelash, D., Mlay, S., 2012. Groupthink decision making deficiency in the requirements engineering process: Towards a crowdsourcing model. SSRN Electron. J..
Mujumdar, D., Kallenbach, M., Liu, B., Hartmann, B., 2011. Crowdsourcing suggestions to programming problems for dynamic web development languages. In: Proc. 2011 annual conference extended abstracts on Human factors in computing systems, pp. 1525–1530.
Muller, C., Chapman, L., Johnston, S., Kidd, C., Illingworth, S., Foody, G., Overeem, A., Leigh, R., 2015. Crowdsourcing for climate and atmospheric sciences: current status and future potential. Int. J. Climatol. 35 (11), 3185–3203.
Musson, R., Richards, J., Fisher, D., Bird, C., Bussone, B., Ganguly, S., 2013. Leveraging the crowd: How 48,000 users helped improve lync performance. IEEE Software 30 (4), 38–45.
Nag, S., 2012. Collaborative competition for crowdsourcing spaceflight software and STEM education using SPHERES Zero Robotics. Massachusetts Institute of Technology.
Nag, S., Heffan, I., Saenz-Otero, A., Lydon, M., 2012. SPHERES Zero Robotics software development: Lessons on crowdsourcing and collaborative competition. In: Proc. 2012 IEEE Aerospace Conference, pp. 1–17.
Nascimento, P., Aguas, R., Schneider, D., de Souza, J., 2012. An approach to requirements categorization using Kano's model and crowds. In: Proc. 16th IEEE International Conference on Computer Supported Cooperative Work in Design, pp. 387–392.
Nayebi, M., Ruhe, G., 2014. An open innovation approach in support of product release decisions. In: Proc. 7th International Workshop on Cooperative and Human Aspects of Software Engineering, pp. 64–71.
Nebeling, M., Leone, S., Norrie, M., 2012. Crowdsourced web engineering and design. In: Proc. 12th International Conference on Web Engineering, pp. 1–15.
Nebeling, M., Norrie, M.C., 2011. Context-aware and adaptive web interfaces : A crowdsourcing approach. In: Proc. 11th International Conference on Web Engineering, pp. 167–170.
Nebeling, M., Norrie, M.C., 2011. Tools and architectural support for crowdsourced adaptation of web interfaces. In: Proc. 11th International Conference on Web Engineering, pp. 243–257.
Nebeling, M., Speicher, M., Grossniklaus, M., Norrie, M.C., 2012. Crowdsourced web site evaluation with crowdstudy. In: Proc. 12th International Conference on Web Engineering, pp. 494–497.
Nebeling, M., Speicher, M., Norrie, M.C., 2013. CrowdAdapt: Enabling crowdsourced web page adaptation for individual viewing conditions and preferences. In: Proc. 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, pp. 23–32.
Nebeling, M., Speicher, M., Norrie, M.C., 2013. CrowdStudy: General toolkit for crowdsourced evaluation of web interfaces. In: Proc. 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems.
Nguyen, H.D.T., Qi, D., Roychoudhury, A., Chandra, S., 2013. SemFix: program repair via semantic analysis. In: Cheng, B.H.C., Pohl, K. (Eds.), Proc. 35th International Conference on Software Engineering, pp. 772–781.
Norman, T.C., Bountra, C., Edwards, A.M., Yamamoto, K.R., Friend, S.H., 2011. Leveraging crowdsourcing to facilitate the discovery of new medicines. Sci. Transl. Med. 3 (88mr1).
Olson, D.L., Rosacker, K., 2012. Crowdsourcing and open source software participation. Service Business 7 (4), 499–511.
Oreizy, P., Gorlick, M.M., Taylor, R.N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D.S., Wolf, A.L., 1999. An architecture-based approach to self-adaptive software. IEEE Intell. Syst. 14, 54–62.
Orlov, M., Sipper, M., 2011. Flight of the FINCH through the java wilderness. IEEE Trans. Evol. Comput. 15 (2), 166–182.
Pacheco, C., Lahiri, S.K., Ernst, M.D., Ball, T., 2007. Feedback-directed random test generation. In: Proc. 29th International Conference on Software Engineering, pp. 75–84.
Pagano, D., Maalej, W., 2013. User feedback in the appstore: An empirical study. In: Proc. 21st IEEE International Conference on Requirements Engineering, pp. 125–134.
Papamartzivanos, D., Damopoulos, D., Kambourakis, G., 2014. A cloud-based architecture to crowdsource mobile app privacy leaks. In: Proc. 18th Panhellenic Conference on Informatics, pp. 59:1–59:6.
Parnin, C., Treude, C., Grammel, L., Storey, M., 2012. Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow. Technical Report. Georgia Institute of Technology.
Pastore, F., Mariani, L., Fraser, G., 2013. CrowdOracles: Can the crowd solve the oracle problem? In: Proc. 6th IEEE International Conference on Software Testing, Verification and Validation, pp. 342–351.
Pawlik, A., Segal, J., Petre, M., Sharp, H., 2015. Crowdsourcing scientific software documentation: a case study of the NumPy documentation project. Comput. Sci. Eng. 17 (1), 28–36.
Peng, X., Ali Babar, M., Ebert, C., 2014. Collaborative software development platforms for crowdsourcing. IEEE Software 31 (2), 30–36.
Peters, D., Parnas, D., 1998. Using test oracles generated from program documentation. IEEE Trans. Software Eng. 24 (3), 161–173.
Petke, J., Harman, M., Langdon, W.B., Weimer, W., 2014. Using genetic improvement & code transplants to specialise a C++ program to a problem class. In: Proc. 17th European Conference on Genetic Programming, pp. 132–143.
Phair, D., 2012. Open Crowdsourcing: Leveraging Community Software Developers for IT Projects. Colorado Technical University Phd. in computer sci..
Pham, R., Singer, L., Liskin, O., Figueira Filho, F., Schneider, K., 2013. Creating a shared understanding of testing culture on a social coding site. In: Proc. 2013 International Conference on Software Engineering, pp. 112–121.
Pham, R., Singer, L., Schneider, K., 2013. Building test suites in social coding sites by leveraging drive-by commits. In: Proc. 35th International Conference on Software Engineering, pp. 1209–1212.
Ponzanelli, L., 2012. Exploiting crowd knowledge in the IDE. University of Lugano.
Ponzanelli, L., Bacchelli, A., Lanza, M., 2013. Leveraging crowd knowledge for software comprehension and development. In: Proc. 17th European Conference on Software Maintenance and Reengineering, pp. 57–66.
Ponzanelli, L., Bacchelli, A., Lanza, M., 2013. Seahawk: Stack Overflow in the IDE. In: Proc. 35th International Conference on Software Engineering, pp. 1295–1298.
Ponzanelli, L., Bavota, G., Di Penta, M., Oliveto, R., Lanza, M., 2014. Mining stackoverflow to turn the ide into a self-confident programming prompter. In: Proc. 11th Working Conference on Mining Software Repositories, pp. 102–111.
Ponzanelli, L., Bavota, G., Di Penta, M., Oliveto, R., Lanza, M., 2014. Prompter: a self–confident recommender system. In: Proc. 30th IEEE International Conference on Software Maintenance and Evolution, pp. 577–580.
Prikladnicki, R., Machado, L., Carmel, E., de Souza, C.R.B., 2014. Brazil software crowdsourcing: a first step in a multi-year study. In: Proc. 1st International Workshop on CrowdSourcing in Software Engineering, pp. 1–4.
Ramakrishnan, S., Srinivasaraghavan, V., 2014. Delivering software projects using captive university crowd. In: Proc. 7th International Workshop on Cooperative and Human Aspects of Software Engineering, pp. 115–118.
Saengkhattiya, M., Sevandersson, M., Vallejo, U., 2012. Quality in crowdsourcing - How software quality is ensured in software crowdsourcing. Lund University.
Saxe, J., Turner, R., Blokhin, K., 2014. CrowdSource: Automated inference of high level malware functionality from low-level symbols using a crowd trained machine learning model. In: 2014 9th International Conference on Malicious and Unwanted Software, pp. 68–75.
Saxton, G.D., Oh, O., Kishore, R., 2013. Rules of crowdsourcing: models, issues, and systems of control. Inf. Syst. Manage. 30 (1), 2–20.
Schiller, T.W., 2014. Reducing the Usability Barrier to Specification and Verification. University of Washington.
Schiller, T.W., Ernst, M.D., 2012. Reducing the barriers to writing verified specifications. In: Proc. 27th ACM International Conference on Object-Oriented Programming Systems, Languages, and Applications, pp. 95–112.
Schneider, C., Cheung, T., 2011. The power of the crowd: performing usability testing using an on-demand workforce. In: Proc. 20th International Conference on Information Systems Development Cutting edge research on Information Systems.
Sen, K., Agha, G., 2006. CUTE and jCUTE: Concolic unit testing and explicit path model-checking tools. In: Ball, T., Jones, R. (Eds.), Computer Aided Verification. In: Lecture Notes in Computer Science, 4144, pp. 419–423.
Seyff, N., Graf, F., Maiden, N., 2010. Using mobile RE tools to give end-users their own voice. In: Proc. 18th IEEE International Conference on Requirements Engineering, pp. 37–46.
Shah, N., Dhanesha, A., Seetharam, D., 2009. Crowdsourcing for e-Governance: Case study. In: Proc. 3rd International Conference on Theory and Practice of Electronic Governance, pp. 253–258.
Sharifi, M., Fink, E., Carbonell, J.G., 2011. SmartNotes: Application of crowdsourcing to the detection of web threats. In: 2011 IEEE International Conference on Systems, Man, and Cybernetics, pp. 1346–1350.
Sharp, H., Baddoo, N., Beecham, S., Hall, T., Robinson, H., 2009. Models of motivation in software engineering. Inf. Software Technol. 51 (1), 219–233.
Sherief, N., 2014. Software evaluation via users' feedback at runtime. In: Proc. 18th International Conference on Evaluation and Assessment in Software Engineering, pp. 1–4.

Sherief, N., Jiang, N., Hosseini, M., Phalp, K., Ali, R., 2014. Crowdsourcing software evaluation. In: Proc. 18th International Conference on Evaluation and Assessment in Software Engineering, pp. 1–4.
Simon, H.A., 1960. The New Science of Management Decision. Harper & Brothers.
Snijders, R., 2015. Crowd-Centric Requirements Engineering: a Method based on Crowdsourcing and Gamification. Utrecht University.
Snijders, R., Dalpiaz, F., 2014. Crowd-centric requirements engineering. In: Proc. 2nd International Workshop on Crowdsourcing and Gamification in the Cloud.
Sobel, D., 1995. Longitude: The True Story of a Lone genius who Solved the Greatest Scientific Problem of his Time. New York: Walker.
de Souza, L.B.L., Campos, E.C., Maia, M.D.A., 2014. Ranking crowd knowledge to assist software development. In: Proc. 22nd International Conference on Program Comprehension, pp. 72–82.
Standish, G., 1994. The chaos report. http://www.standishgroup.com/sample_research_files/chaos_report_1994.pdf. Accessed: 2015-01-27.
Starov, O., 2013. Cloud platform for research crowdsourcing in mobile testing. East Carolina University.
Stol, K.-j., Fitzgerald, B., 2014a. Research protocol for a case study of crowdsourcing software development. Available from: http://staff.lero.ie/stol/publications, University of Limerick.
Stol, K.-J., Fitzgerald, B., 2014. Researching crowdsourcing software development: Perspectives and concerns. In: Proc. 1st International Workshop on CrowdSourcing in Software Engineering, pp. 7–10.
Stol, K.-J., Fitzgerald, B., 2014. Two's company, three's a crowd: A case study of crowdsourcing software development. In: Proc. 36th International Conference on Software Engineering, pp. 187–198.
Stolee, K., Elbaum, S., 2013. Identification, impact, and refactoring of smells in pipe–like web mashups. IEEE Trans. Software Eng. 39 (12), 1654–1679.
Stolee, K.T., Elbaum, S., 2010. Exploring the use of crowdsourcing to support empirical studies in software engineering. In: Proc. 4th ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 1–4.
Stolee, K.T., Elbaum, S., Dobos, D., 2014. Solving the search for source code. ACM Trans. Softw. Eng. Methodol. 23 (3), 26:1–26:45.
Storey, M.-A., Treude, C., van Deursen, A., Cheng, L.-T., 2010. The impact of social media on software engineering practices and tools. In: Proc. FSE/SDP Workshop on Future of Software Engineering Research, pp. 359–364.
Tajedin, H., Nevo, D., 2013. Determinants of success in crowdsourcing software development. In: Proc. 2013 annual conference on Computers and people research, pp. 173–178.
Tajedin, H., Nevo, D., 2014. Value-adding intermediaries in software crowdsourcing. In: Proc. 47th Hawaii International Conference on System Sciences, pp. 1396–1405.
Teinum, A., 2013. User Testing Tool Towards a tool for crowdsource-enabled accessibility evaluation of websites. University of Agder.
Tillmann, N., de Halleux, J., 2008. Pex-White Box Test Generation for .NET. In: Beckert, B., Hhnle, R. (Eds.), Proc. 2nd International Conference on Tests and Proofs. In: Lecture Notes in Computer Science, 4966, pp. 134–153.
Tillmann, N., Moskal, M., de Halleux, J., Fahndrich, M., 2011. TouchDevelop: Programming cloud-connected mobile devices via touchscreen. In: Proc. 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, pp. 49–60.
Tran-Thanh, L., Stein, S., Rogers, A., Jennings, N.R., 2014. Efficient crowdsourcing of unknown experts using bounded multi-armed bandits. Artificial Intelligence 214, 89–111.
Tsai, W.-T., Wu, W., Huhns, M.N., 2014. Cloud-based software crowdsourcing. IEEE Int. Comput. 18 (3), 78–83.
Tung, Y.-H., Tseng, S.-S., 2013. A novel approach to collaborative testing in a crowdsourcing environment. J. Syst. Software 86 (8), 2143–2153.
Usui, Y., Morisaki, S., 2011. An approach for crowdsourcing software development. In: Proc. Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement, pp. 32–33.
Varshney, L.R., 2012. Participation in crowd systems. In: Proc. 50th Annual Allerton Conference on Communication, Control, and Computing, pp. 996–1001.
Vasilescu, B., Filkov, V., Serebrenik, A., 2013. StackOverflow and GitHub: Associations between software development and crowdsourced knowledge. In: Proc. 2013 International Conference on Social Computing, pp. 188–195.
Vasilescu, B., Serebrenik, A., Devanbu, P., Filkov, V., 2014. How social Q&A sites are changing knowledge sharing in open source software communities. In: Proc. 17th ACM Conference on Computer Supported Cooperative Work and Social Computing, pp. 342–354.
Visconti, M., Cook, C., 2002. An overview of industrial software documentation practice. In: Proc. 22nd International Conference of the Chilean Computer Science Society, pp. 179–186.
Vliegendhart, R., Dolstra, E., Pouwelse, J., 2012. Crowdsourced user interface testing for multimedia applications. In: Proc. ACM multimedia 2012 workshop on Crowdsourcing for multimedia, pp. 21–22.
Vukovic, M., Laredo, J., Rajagopal, S., 2010. Challenges and experiences in deploying enterprise. In: Proc. 10th International Conference on Web Engineering.
Wang, H., Wang, Y., Wang, J., 2014. A participant recruitment framework for crowdsourcing based software requirement acquisition. In: Proc. 9th IEEE International Conference on Global Software Engineering, pp. 65–73.
Warner, J., 2011. Next steps in e-government crowdsourcing. In: Proc. 12th Annual International Digital Government Research Conference on Digital Government Innovation in Challenging Times, pp. 177–181.
Watro, R., Moffitt, K., Hussain, T., Wyschogrod, D., Ostwald, J., Kong, D., Bowers, C., Church, E., Guttman, J., Wang, Q., 2014. Ghost Map: Proving software correctness using games. In: The 8th International Conference on Emerging Security Information, Systems and Technologies.
Watson, C., Li, F.W.B., Godwin, J.L., 2012. BlueFix: Using crowd-sourced feedback to support programming students in error diagnosis and repair. In: Proc. 11th International Conference on Web-Based Learning, pp. 228–239.
Weyuker, E.J., 1982. On testing non-testable programs. Comput. J. 25 (4), 465–470.
White, D.R., Arcuri, A., Clark, J.A., 2011. Evolutionary improvement of programs. IEEE Trans. Evol. Comput. 15 (4), 515–538.
Wightman, D., 2013. Search Interfaces for Integrating Crowdsourced Code Snippets within Development Environments. Queen's University.
Wolfson, S.M., Lease, M., 2011. Look before you leap: legal pitfalls of crowdsourcing. In: Proc. American Society for Information Science and Technology, 48, pp. 1–10.
Wu, F., Harman, M., Jia, Y., Krinke, J., Weimer, W., 2015. Deep parameter optimisation. In: Proc. 17th Annual Genetic and Evolutionary Computation Conference.
Wu, W., Tsai, W.-T., Li, W., 2013. An evaluation framework for software crowdsourcing. Front. Comput. Sci. 7 (5), 694–709.
Wu, W., Tsai, W.T., Li, W., 2013. Creative software crowdsourcing: from components and algorithm development to project concept formations. Int. J. Creative Comput. 1 (1), 57–91.
Xiao, L., Paik, H.-Y., 2014. Supporting complex work in crowdsourcing platforms: A view from service-oriented computing. In: Proc. 23rd Australian Software Engineering Conference, pp. 11–14.
Xie, T., 2012. Cooperative testing and analysis: Human-tool, tool-tool, and human-human cooperations to get work done. In: Proc. 12th IEEE International Working Conference on Source Code Analysis and Manipulation (Keynote).
Xie, T., Bishop, J., Horspool, R.N., Tillmann, N., de Halleux, J., 2015. Crowdsourcing code and process via Code Hunt. In: Proc. 2nd International Workshop on CrowdSourcing in Software Engineering.
Xu, X.L., Wang, Y., 2014. Crowdsourcing software development process study on ultra-Large-Scale system. Advanced Materials Research 989–994, 4441–4446.
Xu, X.L., Wang, Y., 2014. On the process modeling of software crowdsourcing based on competitive relation. Adv. Mater. Res. 989–994, 4708–4712.
Xue, H., 2013. Using Redundancy to Improve Security and Testing. University of Illinois at Urbana-Champaign.
Yan, M., Sun, H., Liu, X., 2014. iTest: Testing software with mobile crowdsourcing. In: Proc. 1st International Workshop on Crowd-based Software Development Methods and Technologies, pp. 19–24.
Yuen, M.-C., King, I., Leung, K.-S., 2011. A survey of crowdsourcing systems. In: Proc. 3rd Inernational Conference on Social Computing, pp. 766–773.
Zagalsky, A., Barzilay, O., Yehudai, A., 2012. Example Overflow: using social media for code recommendation. In: Proc. 3rd International Workshop on Recommendation Systems for Software Engineering, pp. 38–42.
Zogaj, S., Bretschneider, U., 2013. Crowdtesting with testcloud - managing the challenges of an intermediary in a crowdsourcing business model. In: Proc. 21st European Conference on Information Systems.
Zogaj, S., Bretschneider, U., Leimeister, J.M., 2014. Managing crowdsourced software testing: a case study based insight on the challenges of a crowdsourcing intermediary. J. Bus. Econ. 84 (3), 375–405.

Ke Mao is pursuing a PhD degree in computer science at University College London, under the supervision of Prof. Mark Harman and Dr. Licia Capra. He received the MSc degree in computer science from the Institute of Software, Chinese Academy of Sciences, China. He worked as a research intern and a software engineer intern at Microsoft and Baidu respectively. He has served as a publicity chair or a PC member for several international workshops on software crowdsourcing. He is currently investigating the application of crowdsourcing in software engineering, with a focus on crowdsourced software testing.

Licia Capra is Professor of pervasive computing in the Department of Computer Science at University College London. Licia conducts research in the area of computer-supported cooperative work. She has tackled specific topics within this broad research field, including crowdsourcing, coordination, context-awareness, trust management, and personalisation. She has published more than 70 papers on these topics, in top venues including SIGSOFT FSE, IEEE TSE, ACM CSCW, SIGIR, SIGKDD, and RecSys.

Mark Harman is Professor of software engineering at University College London, where he is the head of software systems engineering and director of the CREST centre. He is widely known for work on source code analysis and testing and was instrumental in the founding of the field of search-based software engineering, a sub-field of software engineering which is now attracted over 1,600 authors, spread over more than 40 countries.

Yue Jia is a lecturer in the Department of Computer Science at University College London. His research interests cover mutation testing, app store analysis and search-based software engineering. He has published more than 25 papers including one of which received the best paper award at SCAM'08. He co-authored several invited keynote papers at leading international conferences: ICST 2015, SPLC 2014, SEAMS 2014 and ASE 2012 and published a comprehensive survey on mutation testing in TSE. He has served in many programme committees and as program chair for the Mutation workshop and guest editor for the STVR special issue on Mutation.