Worker-Centric Design for Software Crowdsourcing: Towards Cloud Careers

Dave Murray-Rust, Ognjen Scekic and Donghui Lin

Abstract Crowdsourcing is emerging as a compelling technique for the cost-effective creation of software, with tools such as ODesk and TopCoder supporting large scale distributed development. From the point of view of the commissioners of software, there are many advantages to crowdsourcing work—as well as cost, it can be a more scalable process, as there is the possibility of selecting from a large pool of expertise. From the point of view of workers, there is a different set of benefits, including choice of when and how to work, providing a means to build a portfolio, and a lower level of commitment to any particular employer. The crowdsourcing of software development—in common with some other activities such as design—represents an alternative to existing mechanisms that require skilled workers. However, if crowdsourcing were to replace traditional employment for a significant proportion of software developers, the reduced levels of commitment between workers and commissioners could prove problematic for workers over time. In this paper, explore three areas of interest: (i) trust and reputation development; (ii) team selection and team building; (iii) contextualisation of the work carried out. By drawing together work in these areas from the point of view of *workers* rather than *commissioners*, we highlight some of the incipient issues with the growth of crowdsourced labour. We also explore ways in which crowdsourcing of software development—and other skilled practices—differers from microtasking.

D. Murray-Rust (✉)
CISA, School of Informatics, University of Edinburgh, Edinburgh, UK
e-mail: d.murray-rust@ed.ac.uk
URL: http://www.cisa.inf.ed.ac.uk

O. Scekic
Distributed Systems Group, Vienna University of Technology, Vienna, Austria
e-mail: oscekic@dsg.tuwien.ac.at
URL: http://dsg.tuwien.ac.at

D. Lin
Department of Social Informatics, Kyoto University, Kyoto, Japan
e-mail: lindh@i.kyoto-u.ac.jp
URL: http://www.i.kyoto-u.ac.jp

W. Li et al. (eds.), *Crowdsourcing*, Progress in IS,
DOI 10.1007/978-3-662-47011-4_3

1 Introduction

Crowdsourcing is emerging as a powerful tool for carrying out many different tasks, and commissioning work of different kinds. Organisations accrue many benefits from crowdsourcing work, typically including: cost, quality, network effects, lower commitment, greater pool of expertise, scalability and on-demand labour [8, 10, 22].

Most analyses of crowdsourcing take the point of view of the commissioners of work rather than the workers themselves: how is it possible to get work done better, more cheaply, more robustly or faster. When considered as an "outsider" technology, this need to prove value to commissioners of work is completely understandable. However, crowdsourcing is no longer a niche activity, however. In 2009, it was estimated that cloudworkers had been paid up to \$2Bn over the preceding decade [9]; the number of participants has grown by over 100 % per year, and there are now over 6 million cloudworkers worldwide.

Mechanical Turk is seeing a shift from casual, spare time work carried out by Americans, to Indian workers who derive essential income from the work that they do [21]. This has led to some analysis of the ethics of "professional crowdsourcing", where the monetary rewards have a significant effect on the people carrying out the work. Silberman et al. [25] discuss several problems faced by Mechanical Turk workers (Turkers), such as employers who don't pay, or reject work; conning naïve users into downloading malware or participating in scams; and poorly defined or structured tasks. Bederson and Quinn [2] discuss wage-based issues, and call for hourly rates for crowdworkers, or at least an expected hourly rate to be published, along with clear quality metrics which stop employers being able to arbitrarily reject work after it has been done. On the positive side, Horton [12] finds some evidence that online employers are seen as more trustworthy than local employers. Of particular interest is Felstiner's discussion of the crowdsourcing industry in the context of labour laws [8].

Software crowdsourcing is markedly different to "Turking" and other similar microtasking activities, for a number of reasons. Frei [9] divides crowd labour into *micro tasks, macro tasks, small projects* and *complex projects*. While much of the crowdsourcing industry focusses on microtasks, software creation tends to fall into the small- or complex-project brackets, requiring workers to bring in existing skills, and some degree of coordination or direct worker contact.

The software crowdsourcing industry is arguably older than micro-tasking: RentACoder, Guru, LiveOps and Elance all began before Mechanical Turk was introduced, and TopCoder and oDesk appeared prior to the explosion of crowd labour platforms (2006 onwards [9, p. 4]). As such, software crowdsourcing can be seen as a natural evolution of a freelancer-based industry: it is very common for developers to work on a short term basis, being brought in for particular projects without expecting a continuing relationship with the client. The crowdsourcing aspect is largely a technological addition to simplify existing practices.

As well as commercial crowdsourcing, there is a grand tradition of free crowdsourcing exemplified by the Free and Libre Open Source Software (FLOSS) movement, which has created many high profile, high quality complex pieces of software (e.g. the Linux kernel). A side effect of this is the profusion of tools for carrying out distributed programming tasks, such as chat applications, distributed version control systems (DVCSs), bug and issue trackers, unit test frameworks, continuous deployment systems etc. This means that the software community as a whole has greater literacy with the techniques and practices that allow and support distributed working than many other areas.

Just as the entry requirements for creating software are higher than those for microtasking, the monetary incentives are greater too: as of 2010, on average, a Turker earns \$1.25/h, which is less than the minimum wage in India. In contrast, an average developer on oDesk earns \$15/h—double the US minimum wage, and higher than designers (\$10/h) or technical writers (\$8/h) [8].

On the topic of payment, a concern for software creators is the cannibalisation of their market; the average yearly wage for a software developer in the US is \$92 k [27], approximately four times the oDesk average, and which also includes benefits and a sense of job security; this economic advantage is one of the chief motivations for firms to crowdsource work (although there are others, such as innovation and competitiveness [10]). An early article about crowdsourcing [13] examines the disruptive effect that crowdsourced stock photo sites (e.g. iStockphoto.com) had on the professional stock photography market. The article also hints that a large part of professional photographer's ability to charge for their work is dependent on access to professional quality equipment (although this is a position professional photographers may disagree with). Software development is different here: low-end computers can be used to create high quality code, and programmer selection is more likely to be carried out on the basis of a laundry list of technologies mastered than computational hardware owned.

There is the question of whether this commoditisation of the low end really disrupts the lives of working professionals. In the design world, companies such as 99designs produce give access to design at lower prices than were previously available; however, it is an open question whether this represents lost sales for high end design houses, or simply the opening up of the market to a new audience. Software development already has a highly diverse ecosystem, with pricey, local "boutique" consultancies pitted against low cost, low quality outsourcing houses. Software crowdsourcing has the potential to impact on both of these communities, as (a) the price is low enough to be competitive with the cheaper providers and (b) the quality can be high enough to make some mid- to high-level providers worried.

Software development is typically seen as a long term career, with potential for high earnings and a transition into management. Most jobs would include benefits, job security and would provide necessary equipment. This can be contrasted with crowdworking which is fraught with asymmetric power relationships and poor conditions:

> Depending on a firm's quality standards, crowdsourcing can be astoundingly cheap. Crowd workers receive low wages, no benefits, no job security, and have not much prospect at present of organizing to change these conditions. Employers do not need to provide facilities and support for a workforce, nor do they need to pay overhead fees to an outside contractor. [8]

In this paper, we set out our opinions around the question of what it would take to make software crowdsourcing a *sustainable* industry. This means being able to attract intelligent, motivated individuals, who can make enough money to satisfy themselves. Essentially, we ask the question "What would we want from a crowdsourcing marketplace", or, more eloquently:

> Can we foresee a future crowd workplace in which we would want our children to participate? [16]

2 Themes of Interest

As noted previously, there are many issues faced by most crowdworkers, whether performing relatively unskilled micro tasks or larger complex projects. However, there are some issues that are particular to the situation where existing professional activities are replaced with crowdsourcing. While microtasks create a new labour market, there is already a large population of people who expect to be able to construct a career around software development. Here we attempt to tease out some of these expectations and issues, and highlight where current and emerging technologies and systems can help to support these workers.

These themes are by no means comprehensive; rather they represent a solid backbone around which to start building career ladders for cloud software developers, and combatting the atomisation of workers and information asymmetry which are endemic in the cloud.

2.1 Trust and Reputation as Prerequisite for "Cloud Careers"

Arguably, co-workers are one of the most important factors contributing to a pleasant and productive working environment . In traditional companies workers usually cannot directly select their co-workers. However, since the nature of the employment relationship is a long-lasting one, it gives them time to get to know their colleagues and forge working relationships. The management will actively monitor these relationships in order to achieve a more harmonic, and thus more productive or creative environment.

In crowdsourcing environments, the relationship of workers with the platform and co-workers are irregular and short-lived. This leaves no time to get to know and other workers. Crowdsourced teams are often unique, both time- and composition-wise. Co-workers are often hidden behind digital profiles, creating an atmosphere of distrust and discomfort. Furthermore, such settings provide and ideal environment for attempting fraudulent activities, such as multitasking, rent-seeking or tragedy of the commons style exploitation [19].

Hence, managers and workers must be supported in the task of assembling teams; while there are many different approaches to this, almost all of them rely directly or indirectly on some sort of *trust* or *reputation* metrics.

Trust and reputation are two terms often used incorrectly and interchangeably, as varying definitions for both terms exist, and are used in different contexts by different authors. In this paper, we use a loose and operative description which we feel is in general agreement with the majority of the crowdsourcing community.

Trust is a concept denoting one's *personal* expectation of someone else. Reputation is an aggregated, *communal* expectation of an individual.[1] Trust influences reputation, and vice versa:

- $T(a, b)$—denotes the level of trust which a has for b;
- $R_c(x)$—denotes an aggregate measure of worker x's trustworthiness within community c.

In case of crowdsourcing and other socio-technical systems, this means that a worker (Alice) can trust a co-worker (Bob), if her personal feeling or past experience supports the trust. However, Alice's high opinion of Bob may not be shared, and he could have a low reputation within the community. If the low reputation is simply a result of having few collaborations, then over time as Bob works with people, the community view will change and his reputation will increase.

This small example demonstrates two well-known problems: (a) bootstrapping of trust/reputation; and (b) the dilemma of choosing trust over reputation.

The trust-reputation dilemma is reflected in the fact that although reputation reflects an aggregated community view, depending on the particular collaboration pattern in a team, it may be better to favour trust over reputation as a metric. However, this depends on the confidence level of the trust metric, and a trade-off is usually required.

Trust bootstrapping is related to the fact that the trust emerges only after several interactions between the same two subjects have taken place. Reputation bootstrapping is related to the fact that a subject's good reputation can be established only after the majority of community members (or its most influential members) have interacted with the subject.

In crowdsourcing environments, it is often not realistic to expect enough interactions to build up valid trust and reputation metrics. Teams are formed and dispersed, people join and leave the community, and multiple crowdsourcing platforms exists without pervasive identities.

Hence, a number of techniques and systems have been developed which aid the building and management of trust and reputation [20]—see [15] for a survey, and [6] for recent applications to open collaborative systems. Such systems help workers

[1]For a comprehensive and detailed discussion on different aspect and definitions of trust and reputation, the reader is referred to [26].

estimate their initial trust values for other workers based on evaluations of established authorities or majority votes. The intention is that the assessed trust values will exhibit a selective effect, encouraging workers to engage in interactions that will subsequently allow more precise personal trust assessment. As the accuracy of the trust values improves, so does the accuracy of the reputation metrics.

Trust and reputation systems can be highly context-specific [14, 18] and multi-faceted (see Fig. 1 for an example). Each worker is valued differently by each prior collaborator, in each context where they have worked. To form a full evaluation, the opinions of all of those collaborators should be taken into account. However, experiences are highly context dependant, and (for example) a worker's natural behaviours may align more closely with the norms of one context, leading to a higher perception of their quality in that context than others.

This context-dependency is a barrier to trust and reputation metrics being transferable between different platforms/projects. We use the term *reputation transfer* to denote any set of commonly-agreed and shared metrics, methods and data allowing a unified view of a worker's trust and reputation over different platforms.

Reputation transfer is one of the crucial requirements for emerging crowdsourcing systems and one of the important research questions that still needs to be addressed. Solving this problem would allow workers to maintain the reputation across different platforms and avoid platform lock-ins, thus allowing for a more stable future career as platforms come and go. The bootstrapping problem would be greatly reduced, as when a new platform starts up, an initial set of data is available. Overall, it would make the crowdsourced labour more attractive for skilled workers and complex tasks by allowing the workers to move their careers entirely to a competitive and fair crowdsourcing environment.

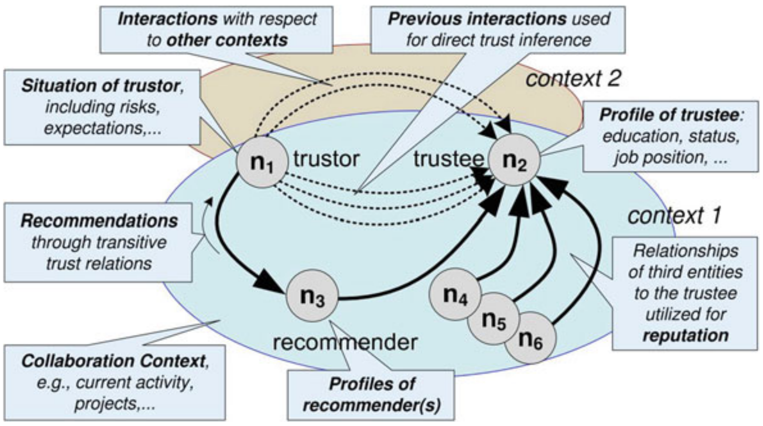
Interactions with respect to other contexts
Previous interactions used for direct trust inference
Situation of trustor, including risks, expectations,...
context 2
Profile of trustee: education, status, job position, ...
n1 trustor
trustee n2
context 1
Recommendations through transitive trust relations
Relationships of third entities to the trustee utilized for reputation
n3
n4
n5
n6
recommender
Collaboration Context, e.g., current activity, projects,...
Profiles of recommender(s)

Fig. 1 Factors influencing trust in socio-technical systems (reproduced from [26]). When evaluating a *trustee* n_2, the *trustor* n_1 must take into account: bilateral interactions in this context; bilateral interactions in other contexts; relationships with others in this context; evaluations of recommenders (and the profile of the recommender)

However, solving the problem of reputation transfer is far from trivial. A general solution requires re-interpreting each worker's past context-dependent performance relative to the current context. The current context may have emerged since the design of the original performance tracking system, so there may be types of data which are unavailable. Additionally, as metrics change over time, behaviour changes to match them, so it becomes unfair to judge past performance on the standards of today.

The proliferation of different metrics and trust models indicate that agreeing on a uniform, context-independent trust and reputation model is practically unfeasible. A new approach and some out-of-the-box thinking will be needed take to address this problem.

2.2 Team Selection

In the crowdsourcing environment, large complex projects require cooperation between a number of crowd workers. As well as the issues of trust and reputation discussed previously, the composition of the team can have an effect on performance, and also the satisfaction of the workers who constitute the teams.

Kittur et al. propose that crowdsourcing labour markets can be regarded as a loosely coupled distributed computing system in with each crowd worker is analogous to a processor [16]. Just as it is important to organize distributed processors for various tasks, team formation and selection for crowdsourced software development is an important issue. Two major areas which should should be dealt with here are the possibility of self-organization of crowd workers, and the manner in which crowd workers are matched to tasks.

2.2.1 Self-Organization of Crowd Workers

Existing crowdsourcing platforms do not have much support for coordination and interaction among crowd workers. In some platforms like Amazon Mechanical Turk, tasks are separated in an atomic manner so that crowd workers do not need to collaborate with each other. Other platforms support complex projects with offline collaboration among crowd workers under the guidance of the work requester. However, creative work like software development requires a large degree of knowledge integration, coordinated effort and interaction among workers.

Self-organization is a process of formation of global order and coordination based on local interaction among individuals/components, which has been previously discussed in natural sciences, distributed computing environments, multi-agent systems, and so on. To deal with organization issues, Crowston et al. have previously studied self-organization of teams in open source software development [4], and illustrate the effectiveness of self-assignment of tasks based on the experiences in several projects. In software crowdsourcing, characteristics such as autonomy, decentralized control, emergence, and adaptation should be considered with respect to the organisation of crowd workers. Methodologies for self-organization in multi-agent systems (e.g. [23]) can be applied to software crowdsourcing, and to some extent a crowd working platform can be conceptualised as a multi-agent system, where reorganisation happens “bottom-up”, with no explicit central control; or, under an internal central control or planning by the work requester.

2.2.2 Task Matching for Crowd Workers

The task allocation problem has been discussed for decades in artificial intelligence and distributed computing circles. In crowdsourcing environments, recent researches focus on how tasks can be decomposed to allow modelling and execution as workflows with iterative tasks for the purpose of quality assurance [5]. However, in creative, complex crowdworking, the matching of tasks to workers is equally important. Two main factors should be considered: the skills possessed by crowd-workers, and the incentives needed to motivate them.

Chilton et al. investigate the task search behaviours of crowd workers, and find that workers tend to gravitate towards the newest tasks on offer due to user interface constraints of existing platforms [3]. Therefore, it is important that crowdsourcing platforms for creative complex work support task matching mechanisms to make full use of the skills of crowd workers. Anagnostopoulos et al. propose an optimization solution for team formation in social networks to deal with following requirements [1], which is also necessary in software crowdsourcing:

1. all skills required by the task should be satisfied;
2. communication overhead within the team should be small;
3. workload of tasks should be fairly balanced among people.

Another important factor in task matching is incentive of crowd workers, including both financial incentive and social incentives [24]. Therefore, tasks, skills, and incentives should be appropriately modelled when developing mechanisms for task matching in software crowdsourcing.

Additionally, the *Social Compute Unit* [7] provides a framework for creating teams of people and associated computing resources whose skills and incentives are matched to solving particular problems.

It will become increasingly necessary to provide mechanisms by which software crowdworkers can collaborate with people they know and trust; where they can organise themselves effectively as situations and contexts evolve; and where they are able to utilise—and improve—their skills on a variety of non-monotonous tasks.

2.3 Contextualisation

The context and purpose of software development can be a large motivating factor for workers; Bederson and Quinn [2] call for reduced anonymity on both sides, and provision of task content. Similarly, Zittrain [28] discusses how decontextualized tasks remove the ability of workers to understand the moral valence of their labour, and decide whether the task they are carrying out is morally acceptable to them. Examples included range from spammers attempting to break Captchas to governments outsourcing recognition of persons of interest in photographs. A worker identifying people in CCTV photos or writing reviews of restaurants they have never visited might have a feeling that they are complicit in something ethically questionable:

> Do not do any HITs that involve: filling in CAPTCHAs; secret shopping; test our web page; … If you feel in your gut it's not on the level, IT'S NOT. Why? Because they are scams…
> spamgirl on TurkerNation [25]

Harris [11] presents a taxonomy of ways in which people can be hired to carry out morally ambiguous activities, and while software development is less prone to some of these issues, the strong emphasis on modular design in software makes it harder to divine the purpose of any particular code unit; a worker creating general computational infrastructure may well not give any clue about the intended purpose of the system.

When discussing general collective intelligence situations, Malone [17] describes the reward for taking part as being based on "Money, Love or Glory". The complement of this (leaving aside the pecuniary aspects) is that one should be engaged in a task that one does not hate, and is not ashamed of. Additionally, Malone suggests that commissioners of collective intelligence should engage with the design questions: What is being done? Who is doing it? Why are they doing it? How is it being done? These questions can be reversed to create a list of questions which crowdworkers should be able to ask, both for their own peace of mind and as a way for commissioning entities to engage with the Love and Glory motivations:

- *"What is the overall project?"* At a basic level, it is important to worker to know what the project it—are they building a recommendation system, or a face recognition system, or a social network? This allows the worker to understand their immediate moral or ethical stance with respect to the work, as well as building commitment and fostering pride in work done.
- *"Who is commissioning the work?"* What does the worker feel about the organisation that is asking for the work to be done? By avoiding anonymity, commissioners have the possibility to build loyalty within their atomized, cloud-based workforce.
- *"Why they are commissioning it?"* Beyond the simple specification of what the system to be built is, there is the question of what is it to be used for, and what are the overall goals and intentions of the commissioner. A worker might have different feelings about creating a data integration tool dependant on whether it was going to give people more useful information in their social networks, or be used by the government to catch criminals. Workers should be able to understand whether the goals of the project align with or conflict with their moral and ethical proclivities? Again, this feeds into developing a sense of pride in the work—beyond technical achievement in creating the software artefact, what is its effect on the world at large going to be?

- *"How is it being done?"* In the context of crowdsourced software development becoming a viable, sustainable career choice, the mechanics of the commissioning process are something with which ethically conscious workers will need to engage. As noted previously, the mechanism of outsourcing work to crowds can have a huge effect on the viability of the profession—if programming were to be carried out through competitions where many teams create solutions, but only one team gets paid, that would change the dynamics of the market. When thinking about longer timescales, workers may want to be selective about what kinds of system they engage with.

3 Discussion

The list of points we have raised here is far from complete. There is similar work in the literature, although much of it is aimed at crowd-work in general, and this tends to have a slant towards the Turking, micro-task end of the spectrum, and hence addresses a different set of communities and issues. Some good overviews are: [16], for describing several ways in which cloud work could be improved, and in particular highlighting the need for creating career ladders, through addressing questions of *motivation, job design, reputation* and *hierarchy*. Bederson and Quinn [2] discuss wage issues for cloud workers, but also provide a set of guidelines for improving the system as a whole, both in economic terms—disclosing pay, long-term feedback, price tasks based on time, grievance processes etc.—an non-economically, by providing task *context* and reducing *anonymity*; Harris deals with a related issue, looking at the ease with which nefarious employers can contract out illegal, unethical or just unsavoury activities [11].

Some of the key issues missing from this treatment are:

- We have discussed the need for trust and reputation between crowdworkers; there is also the need for accountability for commissioners of work. Requesters on Mechanical Turk currently are not bound by reputation systems. Silberman et al. [25] have been building systems for workers to track and comment on the qualities of the requesters, so that workers get a fairer deal.
- Traditional workers have the benefit of many organisational structures that support them. Labour laws ensure a safe and healthy environment; employers are tasked with managing the physical space that they inhabit in working hours; they will meet other people in they workplace; advocacy groups and unions may exist to represent the needs of workers. For a crowd working career, something providing some of the properties of these structures would need to be created.

A question which comes to mind given that the discussion is generally concerned with improving labour practices is: *Why are we interested in people who are typically relatively well off, rather than Turkers earning minimum wage?*. Arguably, most people who can participate in software crowdsourcing are quite well off, in that they have been able to become educated, computer literate, and highly skilled. Hence, they are in a relatively strong position when it comes to protecting their rights and careers. However, it is exactly these qualities which make the software crowdsourcing industry worth engaging with: it is an articulate and visible industry, and as such, can lead the way in changing employment practices. There are battles which need to be fought, and entitlements which need to be won, and hopefully fighting the easier battles first can serve as a blueprint for other industries in the future.

Finally, beyond talking about this, what could we do to ensure that these things happen? Is it primarily computational infrastructure? Or are there social organisations that would need to be put in place? In general, in the area of socio-technical systems, it is necessary to program the people as well as the machines; there is a confluence of human behaviour change and computational support needed to create a sustainable, profitable, long-term and above all *humane* marketplace for crowdsourced software.

References

1. Anagnostopoulos, A., Becchetti, L., Castillo, C., Gionis, A., Leonardi, S.: Online team formation in social networks. In: Proceedings of the 21st International Conference on World Wide Web, pp. 839–848. ACM (2012)
2. Bederson, B.B., Quinn, A.J.: Web workers unite! addressing challenges of online laborers. In: Proceedings of the 2011 Annual Conference Extended Abstracts on Human Factors in Computing Systems—CHI EA'11, pp. 97–106. ACM Press, New York (2011). doi:10.1145/1979742.1979606
3. Chilton, L.B., Horton, J.J., Miller, R.C., Azenkot, S.: Task search in a human computation market. In: Proceedings of the ACM SIGKDD Workshop on Human Computation, pp. 1–9. ACM (2010)
4. Crowston, K., Li, Q., Wei, K., Eseryel, U.Y., Howison, J.: Self-organization of teams for free/libre open source software development. Inf. Softw. Technol. **49**(6), 564–575 (2007). http://linkinghub.elsevier.com/retrieve/pii/S0950584907000080
5. Dai, P., Weld, D.S.: Others: decision-theoretic control of crowd-sourced workflows. In: Twenty-Fourth AAAI Conference on Artificial Intelligence (2010)
6. De Alfaro, L., Kulshreshtha, A., Pye, I., Adler, B.T.: Reputation systems for open collaboration. Commun. ACM **54**(8), 81–87 (2011)
7. Dustdar, S., Bhattacharya, K.: The social compute unit. IEEE Internet Comput. **15**(3), 64–69 (2011). doi:10.1109/MIC.2011.68
8. Felstiner, A.: Working the crowd : employment and labor law in the crowdsourcing industry (2010)
9. Frei, B.: Paid Crowdsourcing. Technical Report. www.smartsheet.com. http://www.smartsheet.com/paid-crowdsourcing-current-state-and-progress (2009)
10. Gassenheimer, J.B., Siguaw, J.A., Hunter, G.L.: Exploring motivations and the capacity for business crowdsourcing. AMS Rev. 1–12 (2013). doi:10.1007/s13162-013-0055-8
11. Harris, C.G.: Dirty deeds done dirt cheap. In: 2011 IEEE International Conference on Privacy, Security, Risk and Trust, pp. 1314–1317 (2011)
12. Horton, J.J.: The condition of the turking class: are online employers fair and honest? Econ. Lett. **111**(1), 10–12 (2011). http://www.sciencedirect.com/science/article/pii/S0165176510004398
13. Howe, J.: The rise of crowdsourcing. Wired Mag. **14**(6), 1–4 (2006)
14. Josang, A., Ismail, R., Boyd, C.: A survey of trust and reputation systems for online service provision. Decis. Support Syst. **43**(2), 618–644 (2007). doi:10.1016/j.dss.2005.05.019. http://linkinghub.elsevier.com/retrieve/pii/S0167923605000849

15. Jøsang, A., Ismail, R., Boyd, C.: A survey of trust and reputation systems for online service provision. Decis. Support Syst. **43**(2), 618–644 (2007)
16. Kittur, A., Nickerson, J.V., Bernstein, M., Gerber, E., Shaw, A., Zimmerman, J., Lease, M., Horton, J.: The future of crowd work. In: Proceedings of the 2013 Conference on Computer Supported Cooperative Work, pp. 1301–1318. ACM (2013)
17. Malone, T.W., Laubacher, R., Dellarocas, C.: The collective intelligence genome. MIT Sloan Manag. Rev. **51**(3), 21–31 (2010). http://raptor1.bizlab.mtsu.edu/S-Drive/KJIH/2010StudyAbroad/JournalArticles/TheCollectiveIntelligenceGenome.pdf
18. Marti, S., Garcia-Molina, H.: Taxonomy of trust: categorizing P2P reputation systems. Comput. Netw. (April 2005), 1–20 (2006). http://www.sciencedirect.com/science/article/pii/S138912860500215X
19. Prendergast, C.: The provision of incentives in firms. J. Econ. Lit. **37**(1), 7–63 (1999). http://www.jstor.org/stable/2564725
20. Resnick, P., Kuwabara, K., Zeckhauser, R., Friedman, E.: Reputation systems. Commun. ACM **43**(12), 45–48 (2000)
21. Ross, J., Irani, L., Silberman, M., Zaldivar, A., Tomlinson, B.: Who are the crowdworkers? Shifting demographics in mechanical turk. In: CHI'10 Extended Abstracts on Human Factors in Computing Systems, pp. 2863–2872 (2010). http://dl.acm.org/citation.cfm?id=1753873
22. Schenk, E., Guittard, C.: Towards a characterization of crowdsourcing practices. J. Innov. Econ. Manag. **1**(7), 93–107 (2011). http://www.cairn.info/revue-journal-of-innovation-economics-2011-1-page-93.htm
23. Serugendo, G.D.M.: Self-organisation and emergence in multi-agent systems. Knowl. Eng. Rev. **20**(2), 165–189 (2005)
24. Shaw, A.D., Horton, J.J., Chen, D.L.: Designing incentives for inexpert human raters. In: Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work, pp. 275–284. ACM (2011)
25. Silberman, M.S., Irani, L., Ross, J.: Ethics and tactics of professional crowdwork. XRDS: Crossroads ACM Mag. Stud. **17**(2), 39 (2010). doi:10.1145/1869086.1869100
26. Skopik, F.: Dynamic trust in mixed service-oriented systems. Ph.D. Thesis (2010). http://hydra.infosys.tuwien.ac.at/Staff/sd/papers/Diss.F.Skopik.pdf
27. Software developer salary in United States. http://www.indeed.com/salary/q-Software-Developer-l-United-States.html. Accessed 21 Nov 2013
28. Zittrain, J.: Ubiquitous human computing. Philos. Trans. Ser. A Math. Phys. Eng. Sci. **366**(1881), 3813–3821 (2008). doi:10.1098/rsta.2008.0116