The Dos and Dont's of Crowdsourcing Software Development

Brian Fitzgerald and Klaas-Jan Stol

Lero—The Irish Software Engineering Research Centre
University of Limerick, Ireland
{bf,klaas-jan.stol}@lero.ie

1 Introduction

In 1957, the eminent computer scientist, Edsger W. Dijkstra, sought to record his profession as "Computer Programmer" on his marriage certificate. The Dutch authorities, although probably more progressive than most, refused on the grounds that there was no such profession. Ironically, just a decade later, the term "software crisis" had been coined, as delegates at a NATO Conference in Garmisch [1] reported a common set of problems, namely that software took too long to develop, cost too much to develop, and the software which was eventually delivered did not meet user expectations. Despite the advances in technology over the past 50 years, this remains problematic, as evidenced by the following quote from the US President's Council of Advisors on Science & Technology (PCAST) in 2012.

> "The problem of predictable development of software with the intended functionality that is reliable, secure and efficient remains one of the most important problems in [ICT]"

A number of initiatives have emerged over the years to address the software crisis. Outsourcing of the software development activity has been on the increase in recent years according to US[1] and European[2] reports. However, in many cases outsourcing of software development has not been successful [2,3]. The success of the open source movement which has proven surprisingly successful at developing high quality software in a cost effective manner [4] has been an inspiration for a number of specific forms of software outsourcing, including opensourcing [5], innersourcing [6] and crowdsourcing [7].

2 Open-Source-Inspired Outsourcing

The conventional wisdom of software engineering suggests that given the inherent complexity of software, it should be developed using tightly co-ordinated, centralized teams, following a rigorous development process. In recent times, the Open Source Software

[1] IT Outsourcing Statistics: 2012/2013.
[2] European IT Outsourcing Intelligence Report.

G.F. Italiano et al. (Eds.): SOFSEM 2015, LNCS 8939, pp. 58–64, 2015.

(OSS) phenomenon has attracted considerable attention as a seemingly agile, practice-led initiative that appears to address these three aspects of the so-called "software crisis": cost, time-scale and quality. In terms of costs, OSS products are usually freely available for public download. From the point of view of development speed, the collaborative, parallel efforts of globally-distributed co-developers has allowed many OSS products to be developed much more quickly than conventional software. In terms of quality, many OSS products are recognized for their high standards of reliability, efficiency and robustness, and the OSS phenomenon has produced several "category killers" (i.e., products that remove any incentive to develop any competing products) in their respective areas—Linux and Apache spring to mind. The OSS model also seems to harness the most scarce resource of all—talented software developers, many of whom exhibit a long-standing commitment to their chosen projects. It is further suggested that the resulting peer review model helps ensure the quality of the software produced.

This brief synopsis illustrates why the OSS topic would be of such interest to the software engineering community, and also provides a hint as to why it would have greater research appeal and interest, particularly in an outsourcing context where companies seek to take advantage of resources beyond co-located developers on a single site. As mentioned, the OSS phenomenon has inspired other forms of outsourcing, of which crowdsourcing is one. This work will focus on crowdsourcing in software development.

3 Crowdsourcing Software Development

Software engineering no longer takes place in small, isolated groups of co-located developers, all working for the same employer, but increasingly takes place in a globalized context across organizations and communities involving many people. One emerging approach to getting work done is crowdsourcing, a sourcing strategy that has emerged since the nineties [8]. Driven by Web 2.0 technologies, organizations can tap into a workforce consisting of anyone with an Internet connection. Customers, or requesters, can advertise chunks of work, or tasks, on a crowdsourcing platform, where suppliers (i.e., individual workers) select those tasks that match their interests and abilities [9].

Crowdsourcing has been adopted in a wide variety of domains, such as design and sales of T-shirts [10] and pharmaceutical research and development [11] and there are numerous crowdsourcing platforms through which customers and suppliers can find each other [12]. One of the best known crowdsourcing platforms is Amazon Mechanical Turk (AMT) [13]. On AMT, chunks of work are referred to as Human Intelligence Tasks (HIT) or micro-tasks. Typical micro-tasks are characterized as self-contained, simple, repetitive, short, requiring little time, cognitive effort and specialized skills, and crowdsourcing has worked particularly well for such tasks [14]. Examples include tagging images, and translating fragments of text. Consequently, remuneration of work is typically in the order of a few cents to a few US dollars.

In contrast to micro-tasks, software development tasks are often interdependent, complex, heterogeneous, and can require significant periods of time, cognitive effort and various types of expertise. Yet, there are cases of crowdsourcing complex tasks; for instance, InnoCentive deal with problem solving and innovation projects, which may yield payments of thousands of US dollars [10]. A number of potential benefits may

arise through the use of crowdsourcing in general, and these would also be applicable in the context of software development specifically:

- *Cost reduction* through lower development costs for developers in certain regions, and also through the avoidance of the extra cost overheads typically incurred in hiring developers;
- *Faster time-to-market* through accessing a critical mass of necessary technical talent who can achieve follow-the-sun development across time zones, as well as parallel development on decomposed tasks, and who are typically willing to work at weekends, for instance;
- *Higher quality* through broad participation: the ability to get access to a broad and deep pool of development talent who self-select on the basis that they have the necessary expertise, and who then participate in contests where the highest quality 'winning' solution is chosen;
- *Creativity and open innovation*: there are many examples of "wisdom of crowds" creativity whereby the variety of expertise available ensures that more creative solutions can be explored, which often elude the fixed mindset that can exist within individual companies, a phenomenon known as 'near-field repurposing of knowledge.'

Given that the first three benefits above (cost, time and quality) directly address the three central problematic areas of the so-called 'software crisis' [15], it is not surprising that a number of authors have argued that crowdsourcing may become a common approach to software development [16,17].

We conducted a case study of a major multinational company who commissioned a crowdsourcing software development initiative using the TopCoder platform [7]. Below we present a number of lessons learned in the form of Dos and Dont's.

4 The Dos in Crowdsourcing Software Development

4.1 Do Familiarize Yourself with the Crowdsourcing Process

The software development approach in crowdsourcing can be significantly different from that which organizations use for their internal development. For example, the crowdsourcing software development process at TopCoder is a waterfall process and it is not trivial to integrate this with the agile type approach which characterizes the majority of in-house development today. It is important to become familiar with the crowdsourcing process at the outset, so that architects, developers and project managers can prepare and discuss internally what needs to be done for a smooth interaction with the crowd.

There are several new roles which emerge when crowdsourcing software development. For example at TopCoder, the interaction with crowd contestants is mediated by co-pilots who are experienced members of the crowd community and platform specialists who interact with customer companies. Also, while the concept of first and second prizes is clear, concepts such as Reliability Bonus and Digital Run points are not so obvious but have significant financial implications for the customer. The level at which

prize money should be pitched for competitions, and the preparing of specifications and reviewing of competitions is also something which needs to be understood in the crowdsourcing software development process.

The warranty periods for crowdsourced work can also be problematic. For example, TopCoder operate a five day warranty period after a competition winning entry has been selected, during which the customer has to accept or reject the submission. This requires discipline at the customer end to ensure that submissions can be internally reviewed. There is also a 30-day warranty period during which problems can be reported. However, it can be difficult to operate this longer warranty period usefully as some much additional interdependent development work would have been done in the intervening 30 days and this would make it difficult to roll back the crowdsourcing element.

4.2 Do Provide Clear Documentation to the Crowd

Documentation clearly plays an important role, as this is the key channel through which crowd developers will know what to develop. The documentation that specifies the context and the requirements for the software development task at hand must be easy to understand and provide sufficient information for crowd developers to do their task. Finding the right balance is important; giving either too little or too much information will result in a deliverable that is likely to be unacceptable. Overwhelming the crowd with information is likely to scare them off, resulting in few or even no submissions. Also, the crowd tend not to have a recurrent relationship with customers. Thus the kind of tacit organizational knowledge that one can take for granted for in-house development does not occur in crowdsourcing. Consequently, far more comprehensive and explicit documentation of requirements is necessary.

4.3 Do Assign a Special Person to Answer Questions from the Crowd

Interacting with the crowd can be a very time-consuming activity. In the case study we conducted, the single point of contact had to have both technical and project management skills, and consequently such a liaison ended up being a senior resource. However, a significant amount of time of this senior person was taken up by answering technical questions on the Q&A forum through which crowd developers asked for clarification. Therefore, a better approach would be to allocate a person who would be well informed about the technical intricacies of the project but who would not have a senior role, and hence be a cheaper resource. The nature of interaction which takes place episodically, perhaps once per day, through the rather narrow Q&A forum also requires quite a lot of discipline on the part of the person charged with that responsibility.

5 The Don'ts in Crowdsourcing Software Development

5.1 Don't Stay Anonymous

A crowdsourcing customer may be concerned about potential IP "leaking," and giving away the company's "Secret Sauce." As a result, a customer may choose to disguise

their participation by staying anonymous, using a pseudonym in contest descriptions. However, a significant downside of that tactic is that such contests may attract very little interest and participation from the crowd. For crowd developers, it can be particularly interesting to work for blue chip companies as doing so allows them build their resumes. It is not uncommon for developers to use their TopCoder 'rating' on their resume as evidence of their technical skills and know-how. By staying anonymous, however, a customer may be much less appealing to work for. Also, the anonymity may offer an inadequate level of protection anyway in that the specifications for a competition may effectively reveal the company's identify anyway.

5.2 Don't Reject Submissions If They Can Easily be Fixed

Once a contest is over, the customer may have five days to accept the 'winning' submission. This means that there is only limited time for a customer to fully analyze and test the deliverable before an accept or reject decision must be made. If a customer decides the deliverable is not of the expected quality, it may be rejected. However, a possible negative side effect is that crowd developers may not participate in future contests of this customer, as doing so involves a risk of spending time and not getting paid for it. If a customer is not yet ready to handle the incoming deliverable, the customer can, of course, just accept the deliverable. After accepting, there is an additional warranty period of 30 days during which identified defects can be reported and fix without additional cost. However, taking this route can pose significant overhead in receiving, checking and integrating the fixed deliverable. Therefore, a customer is probably better off to fix minor defects internally rather than using the warranty period.

5.3 Don't Underestimate the Cost

The cost of crowdsourcing software should not be underestimated. Using the TopCoder platform, for example, the cost of a single contest can be much higher than merely the prize money for the First Prize. Assuming a first place prize of 1,000 USD, the prize money for the second place is 500 USD. Add to that a Reliability Bonus of 200 USD, a Digital Run contribution of 450 USD, a specification review of 50 USD, a review board of 800 USD, and co-pilot fees of 600 USD, a single contest would cost 3,600 USD.

5.4 Don't Expect Miracles

Finally, it is important to stress that crowdsourcing software development does not represent the much sought after 'silver bullet.' Expected benefits from crowdsourcing include high-quality and innovative solutions in a faster time-scale and low cost. Indeed, given TopCoder's workforce of around 700,000 developers, one would expect a significant number of participants for each contest, and consequently, high-quality of innovative deliverables. However, our findings suggest quite a different picture. For the 53 contests held by our case study company, there were a total of 720 registrants, and a total of only 84 submissions, less than two on average. Furthermore, there were more

than 500 issues reported with these submissions. The case company was also quite disappointed by the level of innovation–rather than the expected high-quality HTML5 code (HTML5 has many novel features compared to HTML4), few HTML5 features were actually used due to the portability constraints set forth by the customer company.

6 Conclusion

Crowdsourcing software development is not as straightforward as crowdsourcing micro-tasks found on platforms such as Amazon Mechanical Turk. Given the complexity of software development, we should not be surprised that the difficulties in 'common' (in-house) software development settings are exacerbated when outsourced to a crowd. Yet, little is known about crowdsourcing software development, and our suggested dos and don'ts are based on a single case study. More research is necessary–to that end, we developed a research framework that identifies the key perspectives and concerns [18].

Acknowledgments. This work was supported by Science Foundation Ireland grant 10/CE/I1855 to Lero—The Irish Software Engineering Research Centre, Enterprise Ireland grant IR/2013/0021 to ITEA2-SCALARE, and the Irish Research Council.

References

1. Naur, P., Randell, B. (eds.): Report on a conference sponsored by the NATO SCIENCE COMMITTEE (1968)
2. Nakatsu, R., Iacovou, C.: A comparative study of important risk factors involved in offshore and domestic outsourcing of software development projects: A two-panel delphi study. Information & Management 46, 57–68 (2009)
3. Tiwana, A., Keil, M.: Control in internal and outsourced software projects. Journal of Management Information Systems 26, 9–44 (2009)
4. Feller, J., Fitzgerald, B., Hissam, S., Lakhani, K.: Perspectives on Free and Open Source Software. MIT Press (2005)
5. Ågerfalk, P.J., Fitzgerald, B.: Outsourcing to an unknown workforce: Exploring opensourcing as a global sourcing strategy. MIS Quarterly 32 (2008)
6. Stol, K., Avgeriou, P., Babar, M., Lucas, Y., Fitzgerald, B.: Key factors for adopting inner source. ACM Trans. Softw. Eng. Methodol., 23 (2014)
7. Stol, K., Fitzgerald, B.: Two's company, three's a crowd: A case study of crowdsourcing software development. In: Proc. 36th Int'l Conf. Software Engineering, pp. 187–198 (2014)
8. Greengard, S.: Following the crowd. Communications of the ACM 54, 20–22 (2011)
9. Hoffmann, L.: Crowd control. Communications of the ACM 52 (2009)
10. Howe, J.: Crowdsourcing: Why the Power of the Crowd is Driving the Future of Business. Crown Business (2008)
11. Lakhani, K.R., Panetta, J.: The principles of distributed innovation. Innovations: Technology, Governance, Globalization 2 (2007)
12. Doan, A., Ramakrishnan, R., Halevy, A.: Crowdsourcing systems on the world-wide web. Communications of the ACM 54 (2011)
13. Ipeirotis, P.: Analyzing the amazon mechanical turk marketplace. XRDS 17, 16–21 (2010)
14. Kittur, A., Smus, B., Khamkar, S., Kraut, R.: Crowdforge: Crowdsourcing complex work. In: Proceedings of the ACM Symposium on User Interface Software and Technology (2011)

15. Fitzgerald, B.: Open source software: Lessons from and for software engineering. IEEE Computer 44, 25–30 (2011)
16. Begel, A., Herbsleb, J.D., Storey, M.A.: The future of collaborative software development. In: Proceedings of the ACM Symposium on Computer-Supported Collaborative Work (2012)
17. Kazman, R., Chen, H.M.: The metropolis model: A new logic for development of crowdsourced systems. Communications of the ACM 52 (2009)
18. Stol, K., Fitzgerald, B.: Researching crowdsourcing software development: perspectives and concerns. In: Proc. 1st Int'l Workshop on Crowdsourcing in Software Engineering (2014)