Crowdsourcing in Software Engineering: Models, Motivations, and Challenges

Thomas D. LaToza and André van der Hoek

Abstract
Almost surreptitiously, crowdsourcing has entered software engineering practice. In-house development, contracting, and outsourcing still dominate, but many a software development project today uses crowdsourcing for a variety of purposes, whether it is to squash bugs, test their software, or gather alternative designs for a new user interface. While the overall impact has been mundane thus far, crowdsourcing has the potential to lead to fundamental and disruptive changes in how software will be developed in the future. This paper explores the models of crowdsourcing that have been applied to software development to date, outlines the exciting opportunities that exist, and articulates a series of challenges that must be overcome for crowdsourcing software development to truly reach its potential.

Keywords
Programming teams, staffing, software management

Introduction

Imagine the following headlines:

> "Over 1000 developers build new web browser from scratch in a weekend"
> "Major software company fixes core vulnerability across 100 systems in two hours"
> "Brilliantly creative approach to improve web accessibility designed by individuals across the world"

While these may seem futuristic, even fantastic (and all are fictional), consider the following real examples of crowdsourcing:

- Players of FoldIt, a puzzle game which has over 57,000 users, solved a protein folding problem in three weeks that had stumped researchers for years [7];
- Ten red balloons, scattered across the entire United States, were located in less than nine hours by a team that recruited and coordinated thousands to help in its quest;[1] and
- An encyclopedia of over 35 million articles in 290 languages was created and maintained by a crowd of over 70,000 active contributors.[2]

All were once considered equally fantastic, or at least clearly impossible.

This article is about software engineering, the crowd, and whether similar advances to these can be had in software. While our introductory examples are pure fiction today, it is clear that

[1] en.wikipedia.org/wiki/DARPA_Network_Challenge
[2] en.wikipedia.org/wiki/Wikipedia:About

crowdsourcing is already penetrating software development practice. Topcoder[3] has hosted over 4,000 software design, development, and data science competitions, awarding over $25,000 a day to developers who compete in those competitions. Over 100,000 testers freelance on uTest[4], testing new apps for compatibility with devices, performing functionality testing, and conducting usability inspections and studies. Over 16,000,000 answers have been provided to programming questions on StackOverflow[5], now the 69th most trafficked web site in the US.[6] Bug bounties, particularly concerning security vulnerabilities, are being offered on a regular basis by major software companies such as Netflix, Microsoft, Facebook, and Google.[7] New platforms for crowdsourcing software engineering are emerging with regularity, offering different specialized services (e.g., Bountify[8], AppStori[9], Pay4Bugs[10]).

Models

Many approaches for bringing crowds to software development work exist. To examine those models, it is useful to return to the original definition of crowdsourcing as provided by Howe [1]:

> *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.*

Mao et al. [8] echo this closely in their definition of crowdsourced software engineering:

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

Each definition hones in on the three factors that distinguish crowdsourcing from other outsourced work: (1) the work being solicited through an open call to which basically anyone can respond, (2) the resulting engagement of workers who are unknown to the organization needing the work done, and (3) the potential for the group of workers to be large. These factors delineate crowdsourcing, though the exact nature of the open call and how it is issued, how an overall task is or is not broken down into smaller tasks, if and how workers collaborate, and other such factors remain unspecified. Through variations in such aspects, a number of crowdsourcing models have emerged that have become common in performing software development work.

Before we discuss the models, it is useful to observe that crowdsourcing itself is a form of *collective intelligence*, the more general idea that information processing can emerge from the actions of

[3] www.topcoder.com
[4] www.utest.com
[5] stackoverflow.com
[6] www.alexa.com
[7] bugcrowd.com/list-of-bug-bounty-programs
[8] bountify.co
[9] appstori.com
[10] www.pay4bugs.com

groups of individuals. Several collective intelligence approaches have been applied to software development work. For instance, companies sometimes turn to *open innovation* internally as a mechanism to generate ideas, soliciting input from employees on areas beyond their normal work assignments in order to get large numbers of ideas [2]. Systems that mine and reuse the work of others are also a form of collective intelligence, for example code search engines that offer code examples or autocomplete tools that mine and surface common coding idioms. We do not consider these systems crowdsourcing, as the work is not solicited through an open call to undefined individuals outside the boundaries of an organization.

Peer production. One of the oldest and most well-known models of software crowdsourcing is open source. Tens of thousands contribute to software projects such as Linux, Apache, Rails, and Firefox. Open source software development is an example of *peer production*, a model in which control is decentralized and contributions are made without monetary reward [3]. Key to peer production is that control is distributed and contributors themselves, rather than a paying client, ultimately make the decisions about the scope and goals of the project. Contributors are typically motivated by the opportunity to gain experience with new technologies, bolster their reputation, and contribute to a good cause. To do so, they must first get up to speed – learning about the project's conventions, architecture, designs, and social norms in a process that can take days or weeks – which may dissuade those casually interested from ever making a contribution. Yet, many contributors continue to make open source a success today.

Beyond open source, other forms of peer production exist. In StackOverflow, for instance, developers share hard-earned expertise by answering questions (one might argue that this is not peer production, as goals are set by the question askers; however, the question askers themselves are members of the crowd rather than a distinguished class). Expert questions on StackOverflow receive an answer in a median of just 11 minutes [4], another example of the power of crowdsourcing.

Competitions. A second crowdsourcing model, the *competition*, has recently gained significant attention in software development. The competition has similarities to traditional outsourcing, in which a client requests work and pays for its completion, but differs in treating workers as contestants rather than collaborators. Pioneered for software by TopCoder, projects are first proposed by a client and are then decomposed by a co-pilot (an experienced worker paid directly for their work as a coordinator) into a series of competitions that may cover requirements, architecture, user interface design, implementation, and testing, dividing each into tasks that can be completed in a number of days. Contestants each provide a competing solution; from these, the co-pilot selects a winning entry and runner up and the corresponding workers are paid. Competitions give clients access to diverse solutions, which some believe leads to higher quality results. At the same time, additional costs may arise that are not immediately obvious [5].

Competitions are particularly popular for software tasks in which diversity of input is most valuable. For example, sites such as 99designs[11] let clients crowdsource visual design tasks,

[11] www.99designs.com

reviewing alternative icons, logos, or website designs produced by the crowd to select the best option. Bug bounties, too, fall in this category: different workers may identify different bugs, increasing the likelihood a bug is found.

Microtasking. Another model of crowdsourcing is *microtasking*, in which work is decomposed into a set of self-contained 'microtasks', which each can be completed in minutes and which together compose into a solution to a more complex task. Microtasking is best typified by Amazon's Mechanical Turk (AMT), a general platform in which clients post batches of microtasks (often automatically generated) that workers ('Turkers') complete one-at-a-time. To ensure quality, microtasks are often completed by multiple workers, with voting and other mechanisms used to select the best solution. The primary advantage of this model is its extreme scalability: by making tasks small and self-contained, work can be distributed to arbitrarily large crowds, enabling large tasks to be completed quickly. In software development, this model has found success in testing. Hundreds of thousands of testers participate in labor markets such as UserTesting.com[12], TryMyUI[13], TestBats[14], and uTest[15], testing, for instance, small pieces of functionality or individual aspects of usability. These services offer clients the benefits of a fluid labor force and speed: a workforce can be located and contracted quickly, with worker screening and payment handled through the platform. This enables a labor force to be contracted and complete a task in less time than it might take to even post a traditional job advertisement. For example, UserTesting.com promises to provide usability feedback in less than an hour. For small organizations looking to find skilled help quickly, this provides an enormous benefit.

[12] www.usertesting.com
[13] www.trymyui.com
[14] www.testbats.com
[15] www.utest.com

While peer production, competitions, and microtasking are all forms of crowdsourcing, there are important differences between them. To compare crowdsourcing models and provide a sense of the overall space in which different such models exist, we identify eight foundational and orthogonal dimensions along which crowdsourcing models vary, building on other models of the use of crowdsourcing more generally [10][11]. The following table briefly explains these dimensions.

Dimension	Brief explanation	Scale
crowd size	size of the crowd necessary to effectively tackle the problem	small to large
task length	amount of time a worker spends completing an individual task	minutes to weeks
expertise demands	level of domain familiarity required for a worker to make a contribution	minimal to extensive
locus of control	ownership over the creation of new (sub)tasks	client to workers
incentives	motivational factors that cause workers to engage with the task	intrinsic to extrinsic
task interdependence	degree to which tasks within the overall workflow build on each other	low to high
task context	amount of system information a worker must know to contribute	none to extensive
replication	the number of times the same task may be redundantly completed	none to many

With these dimensions, then, it becomes possible to describe a wide range of models for crowdsourcing software engineering. The table below does so for a concrete example system of each of the three models discussed (peer production, competitions, and microtasking). Other such systems can be similarly captured. Verification games, for instance, transform formal software verification problems into a game-like experience to which non-experts can nonetheless contribute. A game such as PipeJam [9] can be described as (medium, minutes, minimal, client, intrinsic, medium, none, none). It is clear, even from these few examples, that a wide diversity of crowdsourcing systems are possible which vary significantly in approach.

Dimension	Open source	TopCoder	UserTesting.com
crowd size	small – medium	small	medium
task length	hours – days	week	minutes
expertise demands	moderate	extensive	minimal
locus of control	workers	client	client
incentives	intrinsic	extrinsic	extrinsic
task interdependence	moderate	low	low
task context	extensive	minimal	none
replication	none	several	many

Sidebar 1. The dimensions of crowdsourcing models.

Opportunities

It is interesting to consider the forces driving the current emergence of crowdsourcing models, platforms, and environments, particularly in terms of their use within software development organizations. Many of the crowdsourcing models are relatively novel, and their long-term benefits and drawbacks remain poorly understood. Businesses, however, clearly must see tangible benefits to adopt and use crowdsourcing – even if experimentally. Below, we review several motives driving software development organizations to adopt crowdsourcing, as well as the forces motivating software developers to participate in crowdsourcing platforms.

Reduced time to market. Increased speed of development is a frequent reason to engage in crowdsourcing. The possibility to usability test a system in a few hours is tantalizing, given that, in-house, such efforts typically can take much longer. The key, of course, is parallelism: many workers contribute with small efforts that together constitute a potent whole. It would be difficult for an individual or small team to put as many eyes searching for bugs or vulnerabilities as a bug bounty could; neither could such an individual or small team reach the breadth of devices and situations covered by a usability test on uTest. The presence of a fluid, dynamic labor force that can engage with new tasks enables parallelism inherent in work to translate into faster time to market.

This does not mean that every crowdsourced effort is faster. Far from it. The argument holds for crowdsourcing models where work can easily be broken down into short tasks, and where each task is mostly self-contained with minimal coordination demands, enabling a worker to quickly make a contribution. The argument is much less clear for the development of wholesale systems.

Generating alternative solutions. Organizing work into self-contained tasks makes it possible for multiple workers to each independently complete the same task. Because workers bring with them diverse perspectives, backgrounds, and experiences, this often leads to the creation of alternative solutions. By selecting the best amongst these alternatives – or, in some cases, requesting further work combining aspects of some of those alternatives – crowdsourcing makes it possible to obtain higher quality solutions. In StackOverflow, developers can compare many alternative answers to their question, explicitly selecting the best for their needs. Similarly, 99designs enables a client to rapidly solicit and compare many alternative user interface designs, increasing the likelihood that new design ideas will be identified that a single person might never have considered.

Not every task is set up to generate alternatives. Sites such as TestBats focus on achieving test coverage by creating a wide range of diverse tasks, and do not attempt the same task repeatedly. Similarly, in open source we see little alternative generation, as individual developers choose themselves what to work on. Of course, alternative solutions are at times extensively discussed on mailing lists and developers sometimes fork whole projects. But it is rare for developers to explicitly develop and consider, for example, alternative architectures or implementations.

Employing specialists. Decomposing large software development tasks into smaller tasks enables greater flexibility in the use of specialist freelancers. Crowdsourcing might make it possible to rely less on in-house developers who are generalists (and thus perhaps less fluent) or on recruiting specialists into the team when so needed (which tends to require a substantial lead time). Bringing in specialist freelancers through crowdsourcing happens today on a limited basis, and often on smaller tasks that are free to the company (e.g., a Firebase expert who is able to explain the source of an exception on StackOverflow). The potential exists, however, for the model to be taken further through the use of paid freelance work – witness the security experts who find a vulnerability through a bug bounty. This may be especially valuable for development tasks that require intricate, detailed knowledge of technologies or frameworks, where specialists can use their deep understanding to build the best possible design, implement critical code, or troubleshoot an existing code base that in-house developers cannot get 'right'.

Freelancers might choose to become specialists in a narrow range of technologies, performing highly specialized tasks for short engagements across many projects, much as a vascular surgeon is specialized and repeatedly performs a small number of related surgeries. At the same time, freelancing is certainly not applicable in every situation. It is important that a suitable task context be available, otherwise more time may be spent understanding what to do than doing the work. Such task contexts may be difficult to create. Moreover, for many organizations, it is important to retain sufficient expertise in-house so as to guide the freelancers in their work, and check it once it is complete. This may itself create significant new costs.

Democratization of participation. A definitional characteristic of crowdsourcing is the democratization of participation. Rather than assigning work to a team or outsourcing it to a subcontractor, crowdsourced work offers an open call, allowing contributors to determine how, when, and what to contribute, even as crowdsourcing models differ in the degree of control afforded to the crowd. In open source, anyone may choose to voice their opinion and submit contributions for review. In a TopCoder competition, anyone can choose to submit an entry. In a bug bounty, anyone can choose to engage with the code base in an attempt to find problems in the code. To workers, this can be greatly liberating. No longer do they have to work on what they are assigned; rather, they can choose where and when to contribute and be rewarded when their efforts are successful.

Yet, significant barriers often deter prospective contributors today. In open source, developers must first become familiar with the codebase, architecture, build environment, and work practices, which may take days or weeks. In other cases, barriers may be more subtle. In communities that engage in competitions, established experts sometimes win the majority of competitions, discouraging novices and effectively shutting them out of such competitions.

Learning through work. Beyond status and glory, a key reason developers choose to contribute to open source is to learn a new technology. They may want to learn a new framework (e.g., AngularJs) or get up to speed on the style and idioms of a new project by reading some code and contributing a bug fix. While millions use communities such as Codecademy[16] to learn the basics of

[16] www.codecademy.com

new technologies, going beyond requires jumping into a larger, real project. Different models of crowdsourcing allow developers to do so with varying levels of commitment.

Learning in today's crowdsourcing systems, however, first requires software developers to join and acculturate to software projects. While this may be easy for those that have worked on similar projects before, it remains a serious hurdle for those very developers who have the most to learn.

Moving forward

Even as crowdsourcing slowly but surely enters mainstream software development, key challenges remain for crowdsourcing to reach its true potential and for the scenarios sketched in the introduction to become reality – if it is indeed possible to do so. In particular, further realizing many of the benefits of crowdsourcing – reduced time to market through parallelism, involvement of experts for specialized tasks, considering more alternatives – requires further decomposition of tasks and even greater participation in crowdsourcing platforms. In essence, this asks whether it is possible to push – in terms of the dimensions in Sidebar 1 – the size of the crowd larger, the tasks smaller, and the expertise and coordination demands downward.

Several problems must be solved to design such software crowdsourcing models, which must involve new workflows that orchestrate a variety of subtasks to complete more complex tasks. Workflows must address issues of quality, matching work to workers in consideration of their expertise, coordinating the many contributions, sharing project knowledge across the crowd, and other issues. None is a trivial endeavor.

Yet more fundamentally, we observe that it is likely no coincidence that many of the tasks for which crowdsourcing has found the most success to date – testing, creating UI mockups, answering questions – have clear goals and require minimal context. A key tenet of crowdsourcing is that each participant must be precisely informed of the task to be performed. Understanding the degree to which decomposition can create such self-contained tasks for more interdependent software work , and the potential overhead this creates, is a key challenge. Is it possible, for example, for authoring a software architecture to be decomposed into short, self-contained tasks that do not require contributors to understand the entire complexity of the whole? And even if decomposition methods are found, the issue of specification remains: can requirements be crowdsourced and, if not, can they be specified in sufficient detail without imposing onerous overhead? Researchers have begun to study these challenges, examining how new crowdsourcing workflows might be created for a variety of software development tasks [12][8] and examining the challenges that these new workflows bring [5].

Conclusions

Crowdsourcing, in its various forms, has already changed software development. Open source aside, the number of new crowdsourcing platforms, the number of workers who sign up and actively contribute, and the number of organizations actively experimenting with crowdsourcing all are indicative of a phenomenon that in many ways has crept up more than taken the industry by storm. The potential advantages are tangible, and the increasing shift of software development work to fluid labor markets (e.g., over $1 billion annually in freelancing is brokered through Upwork[17] alone) portends the potential for even more dramatic shifts that call into question long-held fundamental beliefs about software development. As in any fundamentally disruptive shift, the ultimate ramifications are far from certain. Will future developers operate as highly-skilled freelancers, choosing microtasks to follow their passion and bolster their skillset [13], or as mindless automatons, their work selected without their interest or consent, as Neal Stephenson envisioned in Snow Crash [6]?

Regardless, serious challenges must be overcome if crowdsourcing is to have the same kind of impact in software development as it has had in other fields. The nature of software has much to do with it. Software is a complex artifact, that is not easily broken down into clearly articulated, self-contained, and rapidly understood and completed tasks. Rather, its intricate and invisible nature poses a challenge to crowdsourcing that we foresee will take many years to address. Even so, it is rare that truly foundational shifts take place in our field, and crowdsourcing has that potential. It is worthwhile for the community to develop a deep understanding of how and when crowdsourced work can be applied within software projects.

[17] www.upwork.com

Biographical Sketches

Thomas D. LaToza is an assistant professor in the Department of Computer Science at George Mason University. His research in software engineering focuses on human aspects of software development, with both empirical and design work on tools for programming, software design, and collaboration. He has served on a variety of program committees, served as co-chair of the Second International Workshop on Crowdsourcing in Software Engineering, and currently serves as co-chair of the Sixth Workshop on the Evaluation and Usability of Programming Languages and Tools. He has degrees in psychology and computer science from the University of Illinois, received a PhD in software engineering from Carnegie Mellon University, and was a postdoctoral research associate at the University of California, Irvine. He is a member of the ACM.

Department of Computer Science
George Mason University
4400 University Drive MSN 4A5
Fairfax, VA 22030 USA
tlatoza@gmu.edu

André van der Hoek is a professor and chair of the Department of Informatics at the University of California, Irvine. His research focuses on understanding and advancing the roles of design, collaboration, and education in software development. He has served on numerous international program committees, was a member of the editorial board of ACM Transactions on Software Engineering and Methodology, and was program co-chair of the 2014 International Conference on Software Engineering. He received joint B.S. and M.S. degrees in business-oriented computer science from the Erasmus University Rotterdam, The Netherlands, and a Ph.D. degree in computer science from the University of Colorado at Boulder. He is a member of the IEEE and an ACM Distinguished Scientist.

Department of Informatics
Donald Bren School Information and Computer Sciences
University of California, Irvine
Irvine, CA 92697
andre@ics.uci.edu

References

[1] J. Howe, “The rise of crowdsourcing.” *Wired*, 14, 2006.

[2] M. Klein and G. Convertino, “An embarrassment of riches.” *Commun. ACM,* vol. 57 (11), October 2014, pp. 40-42.

[3] Y. Benkler and H. Nissenbaum, “Commons-based peer production and virtue.” *The Journal of Political Philosophy*, vol. 14 (4), 2006, pp. 394-419.

[4] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann, “Design lessons from the fastest q&a site in the west.” *Conference on Human Factors in Computing Systems (CHI),* 2011, pp. 2857-2866.

[5] K. Stol and B. Fitzgerald, “Two's company, three's a crowd: a case study of crowdsourcing software development.” *International Conference on Software Engineering (ICSE),* 2014, pp. 187-198.

[6] N. Stephenson, Snow Crash. Bantam Books, 1992.

[7] S. Cooper, F. Khatib, A. Treuille, J. Barbero, J. Lee, M. Beenen, A. Leaver-Fay, D. Baker, Z. Popović, Foldit players, “Predicting protein structures with a multiplayer online game.” *Nature*, vol. 466, 5 August 2010, pp. 756-760.

[8] K. Mao, L. Capra, M. Harman and Y. Jia, “A survey of the use of crowdsourcing in software engineering.” Technical Report RN/15/01, Department of Computer Science, University College London, 2015.

[9] W. Dietl, S. Dietzel, M. D. Ernst, N. Mote, B. Walker, S. Cooper, T. Pavlik, and Z. Popović, “Verification games: making verification fun.” *Workshop on Formal Techniques for Java-like Programs (FTfJP)*, 2012, pp. 42-49.

[10] T. W. Malone, R. Laubacher and C. Dellarocas, “The collective intelligence genome.” MIT Sloan Management Review, vol. 41 (3), 2010, pp. 21-31.

[11] J. Surowiecki, The Wisdom of Crowds. Random House, 2005.

[12] T. D. LaToza, W. B. Towne, C. M. Adriano, and A. van der Hoek, “Microtask programming: building software with a crowd.” *Symposium on User Interface Systems and Technology (UIST),* 2014, pp. 43-54.

[13] A. Kittur, J. V. Nickerson, M. Bernstein, E. Gerber, A. Shaw, J. Zimmerman, M. Lease, and J. Horton, “The future of crowd work.” *Conference on Computer supported cooperative work* (CSCW), 2013, pp. 1301-1318.

Tweets

(1) The majority of large software companies today have used crowdsourcing, whether to gather alternative UI designs, test, or fix bugs.

(2) The possibility to usability test a system in a few hours is tantalizing, given that, in-house, such efforts typically last much longer.

(3) Can software architecting be decomposed into self-contained microtasks that do not require undrstanding the complexity of the whole?

(4) Will future developers work as highly-skilled freelancers or mindless automatons, with microtasks selected for them as in Snow Crash?