

# Crowdsourcing in Software Engineering

## Models, Motivations, and Challenges

Thomas D. LaToza, George Mason University

André van der Hoek, University of California, Irvine

*// Crowdsourcing is changing how software is being developed, through several leading models of organizing work. This article surveys the landscape of crowdsourcing in software development, discusses key motivations for engaging, and highlights several key challenges ahead. //*



### IMAGINE THESE HEADLINES:

*More than 1,000 Developers Build Web Browser from Scratch in One Weekend*

*Major Software Company Fixes Core Vulnerability across 100 Systems in Two Hours*

*Individuals Worldwide Design Brilliantly Creative Approach to Improve Web Accessibility*

Although those headlines might seem futuristic, even fantastic (and all are fictional), consider these real examples of crowdsourcing:

- Players of FoldIt, a puzzle game with more than 57,000 users, solved a protein-folding problem in three weeks that had stumped researchers for years.<sup>1</sup>
- Ten red balloons scattered across the US were located in less than nine hours by a team that

recruited and coordinated thousands to help in its quest.<sup>2</sup>

- A crowd of more than 70,000 contributors created and have maintained an encyclopedia with more than 35 million articles in 290 languages.<sup>3</sup>


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

This article is about software engineering, the crowd, and whether advances like these can be had in software. Although our introductory examples are pure fiction today, crowdsourcing clearly is already penetrating software development practice. Topcoder ([www.topcoder.com](http://www.topcoder.com)) has hosted more than 427,500 software design, development, and data science competitions, awarding more than US\$25,000 a day to competitors. More than 100,000 testers freelance on uTest ([www.utest.com](http://www.utest.com)), testing new apps for compatibility with devices, performing functionality testing, and conducting usability inspections and studies.

In addition, more than 16,000,000 answers to programming questions have been provided on StackOverflow ([stackoverflow.com](http://stackoverflow.com)), now the 69th-most-trafficked website in the US (as determined through [www.alexa.com](http://www.alexa.com)). Major software companies such as Netflix, Microsoft, Facebook, and Google regularly offer bug bounties, particularly concerning security vulnerabilities.<sup>4</sup> New platforms for crowdsourcing software engineering are emerging regularly, offering different specialized services—for example, Bountify ([bountify.co](http://bountify.co)), AppStori ([appstori.com](http://appstori.com)), and Pay4Bugs ([www.pay4bugs.com](http://www.pay4bugs.com)).

### Models

To examine crowdsourcing models for software development, it's useful



to return to Jeff Howe's original definition:

*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.*<sup>5</sup>

Ke Mao and his colleagues closely echoed that definition:

*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.*<sup>6</sup>

Each definition homes in on the three factors that distinguish crowdsourcing from other outsourced work:

- The work is solicited through an open call to which basically anyone can respond.
- The workers who volunteer are unknown to the organization needing the work done.
- The group of workers can be large.

What the open call's exact nature is and how it's issued, how an overall task is or isn't broken down into smaller tasks, if and how workers collaborate, and other such factors remain unspecified. Through variations in such aspects, various crowdsourcing models have emerged that have become common in software development.

Crowdsourcing is a form of collective intelligence, the general idea being that information processing can emerge from the actions of groups of individuals. Several

collective-intelligence approaches have been applied to software development. For example, companies sometimes turn to internal *open innovation*, soliciting employee input on areas beyond their normal work assignments to get many ideas.<sup>7</sup> Another example involves systems that mine and re-

which might dissuade those casually interested from ever contributing. Yet, many contributors continue to make open source a success.

Other forms of peer production exist. For instance, in StackOverflow, developers share hard-earned expertise by answering questions.

Most large software companies have used crowdsourcing, whether to gather alternative UI designs, test, or fix bugs.

use others' work—for example, code search engines that offer code examples or autocomplete tools that mine and surface common coding idioms. We don't consider these systems crowdsourcing because they don't solicit work through an open call to undefined individuals outside an organization's boundaries.

### Peer Production

One of the oldest and best-known models of software crowdsourcing is open source. Tens of thousands of people contribute to software projects such as Linux, Apache, Rails, and Firefox. Open source development is an example of *peer production*, a model in which control is decentralized and contributions are made without monetary reward.<sup>8</sup> The contributors, rather than a paying client, decide the project's scope and goals.

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. This process can take days or weeks,

(You could argue that this isn't peer production because the question askers set the goals. However, the question askers are themselves members of the crowd rather than a distinguished class.) Expert questions on StackOverflow receive an answer in a median of just 11 minutes,<sup>9</sup> another example of crowdsourcing's power.

### Competitions

A second crowdsourcing model, competitions, has recently gained significant attention in software development. Pioneered for software by TopCoder, competitions are similar to traditional outsourcing, in which a client requests work and pays for its completion. However, they treat workers as contestants rather than collaborators. First, a client proposes a project. Then a copilot (an experienced worker paid for this work) decomposes the project into a series of competitions that might cover requirements, architecture, UI design, implementation, and testing. The copilot divides each competition into tasks that can be completed in a number of days. Contestants each provide a competing solution; from these, the copilot selects a winning entry and runner-up,

TABLE 1

The dimensions of crowdsourcing.

Dimension	Explanation	Scale
Crowd size	The size of the crowd necessary to effectively tackle the problem	Small to large
Task length	The amount of time a worker spends completing a task	Minutes to weeks
Expertise demands	The level of domain familiarity required for a worker to contribute	Minimal to extensive
Locus of control	Ownership of the creation of tasks or subtasks	Client to workers
Incentives	The factors motivating workers to engage with the task	Intrinsic to extrinsic
Task interdependence	The degree to which tasks in the overall workflow build on each other	Low to high
Task context	The amount of system information a worker must know to contribute	None to extensive
Replication	The number of times the same task might be redundantly completed	None to many

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 might arise that aren't immediately obvious.<sup>10</sup>

Competitions are particularly popular for software tasks in which diverse input is most valuable. For example, sites such as 99designs ([www.99designs.com](http://www.99designs.com)) let clients crowdsource visual-design tasks, 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 might identify different bugs, increasing the likelihood a bug is found.

### Microtasking

This model decomposes work into a set of self-contained microtasks, which can each be completed in minutes and which together constitute a solution to a more complex task. Microtasking is typified by Amazon's Mechanical Turk, a general platform in which clients post batches of microtasks (often automatically generated) that workers

("Turkers") complete one at a time. To ensure quality, multiple workers might be asked to complete the same microtask, with the best solution selected by voting and other mechanisms.

This model's primary advantage is its extreme scalability. Because the tasks are small and self-contained, work can be distributed to arbitrarily large crowds, enabling the quick completion of large tasks.

This model has found success in software testing. Hundreds of thousands of testers participate in labor markets such as UserTesting.com ([www.usertesting.com](http://www.usertesting.com)), TryMyUI ([www.trymyui.com](http://www.trymyui.com)), TestBats ([www.testbats.com](http://www.testbats.com)), and uTest ([www.utest.com](http://www.utest.com)). These services offer clients the benefits of a fluid labor force and speed. Clients can quickly locate and contract a workforce, with the platform handling worker screening and payment. So, a client can contract a labor force and obtain the completed task in less time than it might take to 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.

### The Dimensions of Crowdsourcing

Peer production, competitions, and microtasking have important differences. To compare crowdsourcing models and provide a sense of the space in which they exist, we identified eight foundational and orthogonal dimensions along which the models vary (see Table 1), building more generally on other taxonomies of the use of crowdsourcing.<sup>11,12</sup>

These dimensions enable the description of a range of models for crowdsourcing software engineering. Table 2 applies the dimensions to concrete example systems for peer production, competitions, and microtasking.

Other such systems can be similarly captured. For example, verification games transform formal software verification problems into a game-like experience to which even nonexperts can contribute. Using the dimensions, we describe the PipeJam<sup>13</sup> game thusly:

- crowd size: medium
- task length: minutes

Applying the dimensions of crowdsourcing to concrete examples of three crowdsourcing models.

Dimension	Crowdsourcing model and example		
	Peer production (open source)	Competitions (TopCoder)	Microtasking (UserTesting.com)
Crowd size	Small to medium	Small	Medium
Task length	Hours to days	Week	Minutes
Expertise demands	Moderate	Extensive	Minimal
Locus of control	Worker	Client	Client
Incentives	Intrinsic	Extrinsic	Extrinsic
Task interdependence	Medium	Low	Low
Task context	Extensive	Minimal	None
Replication	None	Several	Many

- expertise demands: minimal
- locus of control: client
- incentives: intrinsic
- task interdependence: medium
- task context: none
- replication: none

As these examples show, crowdsourcing systems with significantly varying approaches are possible.

## Motivations

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

## Reduced Time to Market

Increased development speed is a frequent reason for crowdsourcing. The possibility to perform usability testing of a system in a few hours is tantalizing, given that such efforts typically take much longer in-house. The key is parallelism: many workers contribute with small efforts that constitute a potent whole. An individual or small team would have difficulty devoting as many eyes to search for bugs or vulnerabilities as a bug bounty could. Neither could such an individual or small team reach the breadth of devices and situations that a uTest usability test covers. A fluid, dynamic labor force that can engage with new tasks enables the parallelism inherent in work to translate into faster time to market.

This doesn't mean that every crowdsourced effort is faster. Far from it. The argument for crowdsourcing's speed holds for models with two characteristics. First, work must easily be broken down into short tasks. Second, each task must be mostly self-contained with

minimal coordination demands, letting workers quickly make a contribution. These characteristics aren't necessarily true when work scales up—for instance, to the development of wholesale systems or significant aspects thereof.

## Generating Alternative Solutions

Organizing work into self-contained tasks enables multiple workers to independently complete the same task. Because workers have diverse perspectives, backgrounds, and experiences, this often leads to the creation of alternative solutions. By selecting the best alternative—or, in some cases, requesting more work combining aspects of several alternatives—crowdsourcing can produce higher-quality solutions. In StackOverflow, developers can compare many alternative answers to their question, selecting the best for their needs. Similarly, 99designs lets clients rapidly solicit and compare many UI designs, increasing the likelihood of identifying new design ideas that no one person might have considered.

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

## Employing Specialists

Decomposing large 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 in a particular area of work) or on

might be especially valuable for development tasks requiring intricate, detailed knowledge of technologies or frameworks. For that, specialists can use their deep understanding to build the best possible design, implement critical code, or troubleshoot a code base that in-house developers can't 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 repeatedly performs a small number of related surgeries. At the same time, freelancing certainly isn't always applicable. A suitable task context must be available; otherwise, developers might spend more time trying to understand what to do than doing the work. Creating such contexts might be difficult. Moreover, many organizations need to retain sufficient expertise in-house so as to guide the freelancers in their

of control afforded to the crowd. In open source, anyone can voice their opinion and submit contributions for review. In a TopCoder competition, anyone can submit an entry. In a bug bounty, anyone can engage with the code base to find problems. To workers, this can be greatly liberating. They no longer have to work on what they've been 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. In open source, developers must first become familiar with the code base, architecture, build environment, and work practices, which might take days or weeks. In other cases, the barriers might be more subtle. In communities that engage in competitions, established experts sometimes win most of the competitions, discouraging novices and effectively shutting them out of such competitions.

The possibility to usability-test a system in a few hours is tantalizing, given that, in-house, that task often takes considerably longer.

recruiting specialists into the team when necessary (which often requires 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 (for example, a Firebase expert who can explain the source of an exception on StackOverflow). However, the model could be taken further by using paid freelance work—witness the security experts who find a vulnerability through a bug bounty. This

work and to check it once it's complete. This might itself create significant new costs.

## The 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, letting contributors determine how, when, and what to contribute, even as crowdsourcing models differ in the degree

## Learning through Work

As we mentioned before, a key reason developers contribute to open source is to learn a new technology. They might want to learn a new framework (for example, AngularJS) or get up to speed on the style and idioms of a new project by reading some code and contributing a bug fix. Whereas millions use communities such as Codecademy ([www.codecademy.com](http://www.codecademy.com)) to learn the basics of new technologies, going beyond requires jumping into a larger, real project. Different crowdsourcing models let developers do so with varying commitment levels.

However, to learn in today's crowdsourcing systems, software developers must first join, and acculturate themselves to, software projects. Although this might be easy for



those who have worked on similar projects, it remains a serious hurdle for the very developers who have the most to learn.

## Challenges

Even as crowdsourcing slowly but surely enters mainstream software development, key challenges remain for it to reach its true potential and for the scenarios sketched in the introduction to become reality—if that's indeed possible. In particular, further realizing many of crowdsourcing's benefits requires further decomposition of tasks and even greater participation in crowdsourcing platforms. Essentially, this asks whether we can increase the crowd's size, shrink the tasks, and lower the expertise and coordination demands.

Designing such software crowdsourcing models will require solving several problems. These models must involve new workflows that orchestrate a variety of subtasks to complete more complex tasks. Workflows must address quality issues; match work to workers, taking into consideration their expertise; coordinate the many contributions; share project knowledge across the crowd, and deal with other issues. None of these endeavors is trivial.

Yet more fundamentally, it's likely no coincidence that many of the tasks for which crowdsourcing has found the most success—testing, creating UI mockups, and 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

## ABOUT THE AUTHORS



**THOMAS D. LATOZA** is an assistant professor in George Mason University's Department of Computer Science. His research focuses on software development's human aspects, with both empirical and design work on tools for programming, software design, and collaboration. He was a cochair of the Second International Workshop on Crowdsourcing in Software Engineering and the Sixth Workshop on Evaluation and Usability of Programming Languages and Tools. LaToza received a PhD in software engineering from Carnegie Mellon University and was a postdoctoral research associate at the University of California, Irvine. He's a member of ACM. Contact him at [tlatoya@gmu.edu](mailto:tlatoya@gmu.edu).



**ANDRÉ VAN DER HOEK** is a professor in and the 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 was a member of the editorial board of *ACM Transactions on Software Engineering and Methodology* and a program cochair of the 2014 International Conference on Software Engineering. Van der Hoek received a PhD in computer science from the University of Colorado Boulder. He's a member of IEEE and an ACM Distinguished Scientist. Contact him at [andre@ics.uci.edu](mailto:andre@ics.uci.edu).

key challenge. For example, can the authoring of a software architecture be decomposed into short, self-contained tasks that don't require contributors to understand the complexity of the whole?

Even if decomposition methods are found, specification will remain an issue. Can requirements be crowdsourced? If not, can they be specified in sufficient detail without onerous overhead? Researchers have begun to study these problems, determining how to create crowdsourcing workflows for a variety of development tasks<sup>6,14</sup> and examining the challenges involved in designing new workflows.<sup>10</sup>

Crowdsourcing in its various forms has already changed software development. Open source aside, the number of new crowdsourcing platforms, the number of workers signing up and actively contributing, and the number of organizations actively experimenting with crowdsourcing all indicate a phenomenon that in many ways has crept up on the industry rather than taking it by storm. The potential advantages are tangible. The increasing shift of development work to fluid labor markets—for example, more than \$1 billion annually in freelancing is brokered through Upwork ([www.upwork.com](http://www.upwork.com))

.com) alone—portends the potential for even more dramatic shifts.

Such shifts might well 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 skill set?<sup>15</sup> Or, will they be mindless automations, their work selected without their interest or consent, as Neal Stephenson envisioned in *Snow Crash*?<sup>16</sup>

Regardless, serious challenges must be overcome if crowdsourcing is to have the same kind of impact in software development that it has had in other fields. The nature of software has much to do with this. Software is complex and isn't 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 will take years to address. Even so, truly foundational shifts rarely take place in our field, and crowdsourcing has that potential. It's worthwhile for the community to develop a deep understanding of

how and when to apply crowdsourcing in software projects. ☞

## References

1. S. Cooper et al., "Predicting Protein Structures with a Multiplayer Online Game," *Nature*, 5 Aug. 2010, pp. 756–760; [www.nature.com/nature/journal/v466/n7307/full/nature09304.html](http://www.nature.com/nature/journal/v466/n7307/full/nature09304.html).
2. "DARPA Network Challenge," *Wikipedia*, 2015; [https://en.wikipedia.org/wiki/DARPA\\_Network\\_Challenge](https://en.wikipedia.org/wiki/DARPA_Network_Challenge).
3. "Wikipedia:About," *Wikipedia*, 2015; <https://en.wikipedia.org/wiki/Wikipedia:About>.
4. "The Bug Bounty List," Bugcrowd, 2105; <https://bugcrowd.com/list-of-bug-bounty-programs>.
5. J. Howe, "Crowdsourcing: A Definition," blog, 2 June 2006; [http://crowdsourcing.typepad.com/cs/2006/06/crowdsourcing\\_a.html](http://crowdsourcing.typepad.com/cs/2006/06/crowdsourcing_a.html).
6. K. Mao et al., "A Survey of the Use of Crowdsourcing in Software Engineering," research note RN/15/01, Dept. Computer Science, University College London, 2015.
7. M. Klein and G. Convertino, "An Embarrassment of Riches," *Comm. ACM*, vol. 57, no. 11, 2014, pp. 40–42.
8. Y. Benkler and H. Nissenbaum, "Commons-Based Peer Production and Virtue," *J. Political Philosophy*, vol. 14, no. 4, 2006, pp. 394–419.
9. L. Mamykina et al., "Design Lessons from the Fastest Q&A Site in the West," *Proc. 2011 SIGCHI Conf. Human Factors in Computing Systems* (CHI 11), 2011, pp. 2857–2866.
10. K. Stol and B. Fitzgerald, "Two's Company, Three's a Crowd: A Case Study of Crowdsourcing Software Development," *Proc. 36th Int'l Conf. Software Eng. (ICSE 14)*, 2014, pp. 187–198.
11. T.W. Malone, R. Laubacher, and C. Dellarocas, "The Collective Intelligence Genome," *MIT Sloan Management Rev.*, vol. 41, no. 3, 2010, pp. 21–31.
12. J. Surowiecki, *The Wisdom of Crowds*, Random House, 2005.
13. W. Dietl et al., "Verification Games: Making Verification Fun," *Proc. 14th Workshop Formal Techniques for Java-Like Programs (FTfJP 12)*, 2012, pp. 42–49.
14. T.D. LaToza et al., "Microtask Programming: Building Software with a Crowd," *Proc. 27th Ann. ACM Symp. User Interface Software and Technology (UIST 14)*, 2014, pp. 43–54.
15. A. Kittur et al., "The Future of Crowd Work," *Proc. 2013 Conf. Computer Supported Cooperative Work (CSCW 13)*, 2013, pp. 1301–1318.
16. N. Stephenson, *Snow Crash*, Bantam Books, 1992.



Visit the podcast for professional developers to hear in-depth interviews with some of the top minds in software engineering.

[www.se-radio.net](http://www.se-radio.net)

Sponsored by



Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.