Expert Crowdsourcing with Flash Teams

Daniela Retelny, Sébastien Robaszkiewicz, Alexandra To, Walter Lasecki*, Jay Patel, Negar Rahmati, Tulsee Doshi, Melissa Valentine, Michael S. Bernstein
Stanford University, University of Rochester*
{dretelny, robi, ato1120, jayhp9, negar, tdoshi, mav, msb}@cs.stanford.edu, wlasecki@cs.rochester.edu

ABSTRACT
We introduce *flash teams*, a framework for dynamically assembling and managing paid experts from the crowd. Flash teams advance a vision of expert crowd work that accomplishes complex, interdependent goals such as engineering and design. These teams consist of sequences of linked modular tasks and handoffs that can be computationally managed. Interactive systems reason about and manipulate these teams' structures: for example, flash teams can be recombined to form larger organizations and authored automatically in response to a user's request. Flash teams can also hire more people elastically in reaction to task needs, and pipeline intermediate output to accelerate completion times. To enable flash teams, we present Foundry, an end-user authoring platform and runtime manager. Foundry allows users to author modular tasks, then manages teams through handoffs of intermediate work. We demonstrate that Foundry and flash teams enable crowdsourcing of a broad class of goals including design prototyping, course development, and film animation, in half the work time of traditional self-managed teams.

Author Keywords
Crowdsourcing; expert crowd work; flash teams

ACM Classification Keywords
H.5.3 Information Interfaces and Presentation (e.g. HCI): Group and Organization Interfaces

INTRODUCTION
Crowdsourcing systems coordinate large groups of people to solve problems that a single individual could not achieve at the same scale. Microtasking systems typically use highly-controlled workflows to manage paid, non-expert workers toward expert-level results (e.g., [6, 28, 10, 25, 38]). While these crowdsourcing approaches are effective for simple independent tasks, many real-world tasks such as software development and design remain largely out of reach. Such tasks require deep domain knowledge that is difficult to decompose into independent microtasks anyone can complete [26, 42, 40]. Unlocking these capabilities is critical to the vision of universal, on-demand crowd support for end users. For example, could we enable anybody with a napkin sketch of a design idea to ask the crowd to follow the user-centered design process and create a user-tested, high-fidelity prototype of that idea within twenty-four hours?

In this paper, we explore the feasibility of solving these complex, interdependent problems via *structured collaborations between experts from the crowd*. We aim to gather experts from online marketplaces such as oDesk [1] into small teams that can, for example, follow the user-centered design process to transform a napkin sketch into a fully user-tested prototype, create an animated movie from a prompt, or develop an entire online class platform complete with video content and quizzes — all in roughly one day.

Experts from the crowd, however, tend to work as isolated contractors, and microtask crowdsourcing techniques (e.g., [2, 30, 6, 25, 43]) cannot coordinate these experts because they do not effectively leverage participants' diverse skills and expertise. To enable end users to reliably crowdsource complex work, we take inspiration from organizational behavior research that suggests even temporary groups can coordinate complex work effectively, if they have an enabling team structure [41] to encode who is working together and who is responsible for which tasks [8, 5]. Our goal is to create such structures with far less manual effort and more scalable replicability than traditional organizations.

To achieve this goal, we frame expert crowd work around *sequences of linked tasks*. We hypothesize that these chains of focused work could maintain the coordinating strength of lightweight team structures while providing a representation that interactive systems can leverage to support collaboration, create new teams automatically, grow and shrink teams on demand, and combine teams into larger organizations.

We thus present *flash teams*, a framework for dynamically assembling and managing crowdsourced expert teams. Flash teams are a sequence of modular tasks that draw on paid experts from the crowd. Each task in a flash team requires an input (e.g., low-fidelity prototype) and produces an output of another type (e.g., heuristic evaluation). End users create workflows by linking these modular tasks together to pass each task's output as input to the next task. They then use a web application to launch and manage the team, monitoring the workflow if desired. For example, when creating a user-centered design flash team, the user might begin with a task where a UI designer creates a low-fidelity mockup of the

UIST '14, October 5–8, 2014, Honolulu, HI, USA.

ACM 978-1-4503-3069-5/14/10$15.00.
http://dx.doi.org/10.1145/2642918.2647409

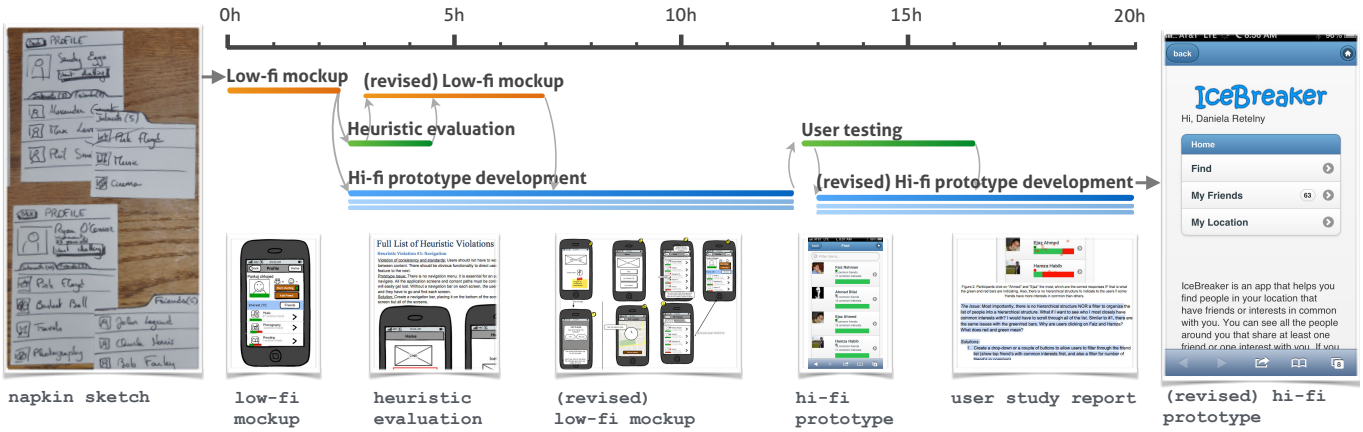


Figure 1: A flash team is a linked set of modular tasks that draw upon paid experts from the crowd, often three to six at a time, on demand. The napkin sketch design team follows the user-centered design process to create a series of prototypes and iterate based on feedback to produce a user-tested software prototype within a day. Multiple arrows indicate the beginning and end of pipelining; lighter bars indicate possible elastic growth.

user's idea. The designer would pass the mockup on to a UX researcher for a heuristic evaluation. Tasks would proceed through the chain of a revised mockup, a high-fidelity prototype, a user study and a revised software prototype, crossing diverse expertise from one or more UI designers, UX researchers, and software developers.

Flash teams' modular task structure is transparent to computational systems, enabling those systems to leverage the structure to orchestrate crowd dynamics. For example, teams can be *combined*: since teams are modular, their component tasks can be composed like Lego blocks to form larger organizations that persist only for a day or an afternoon. Moreover, by leveraging automated path search through the space of previous teams' intermediate inputs and outputs, end users can *assemble a flash team on-demand by providing only the desired input and output*. During runtime, flash teams are *elastic*: they grow and shrink to meet task demands by hiring additional members from the crowd. Finally, their work can be *pipelined*: when in-progress results are enough for downstream tasks to begin work, the system passes in-progress results along to accelerate completion times.

We embody this approach in *Foundry*, a platform for end users to create flash teams and a runtime environment to manage them. Foundry allows users to create flash teams by directly authoring each task or forking a team that other users authored and then recruit from the oDesk online labor marketplace [1]. For team members, Foundry takes on managerial responsibilities to walk team members through the workflow, notify them of any schedule shifts, scaffold the handoff process, and provide members with shared awareness [14]. For the end user, Foundry abstracts away low-level management effort while allowing for high-level oversight and guidance.

This research advances two important ideas. First, flash teams are the first to leverage the *scale of paid crowdsourcing for expert work*. This approach pushes beyond volunteer peer production systems with a vision of elastic and on-demand organizations that manage teams of paid experts from the crowd. Second, this paper offers a method of scaling expert crowd work through *computational management of an elastic workforce*. Flash teams enable complex work at crowd scale by automating the structures of traditional organizations. We introduce specific affordances such as structured handoffs, directly-responsible individuals (DRIs), and elastic growth, embedded within the Foundry platform. Empirical results from our field experiment demonstrate that flash teams can accomplish efficient and high-quality work.

RELATED WORK

In this section, we connect two threads of research: crowdsourcing and organizational behavior.

Crowdsourcing with experts

Crowdsourcing research aims to integrate large-scale, API-friendly labor marketplaces into software through open calls and task decomposition. Crowdsourcing has largely focused on tasks any individual can complete: many crowdsourcing platforms are built to accomplish tasks that require little training (e.g., Amazon Mechanical Turk) and recruit amateurs (e.g. FoldIt [10]). Consequently, most crowdsourcing workflows and algorithms [32] aim to structure non-expert contributions to produce expert-level performance. For example, MapReduce frameworks can guide crowds to write simple encyclopedia entries [25], and clustering workflows can produce expert-level taxonomies [9]. These workflows can even be authored by crowd members themselves [27]. Examples of microtask crowds pursuing expert performance include document editing [6], translation [21], and visual question answering [7]. AI techniques optimize these workflows [13]. Prior work that has gone beyond microtasks has still focused on recreating the performance of a single expert [29].

Prior work suggested that expertise might play a role in crowdsourcing [26]: flash teams instantiate a first step toward that goal. Other crowdsourcing campaigns already recruit experts, for example using workflow support tools [33] or helping solve planning problems [43]. Often these campaigns focus on a single expertise area such as math [12]. To date, these expert crowdsourcing efforts have been one-offs, needing to create an online presence to gather participants for each new goal. Our intent is to create a general approach and

platform that can support a large number of tasks on demand. In doing so, we open the door to a range of higher-level workflows that can assume expert knowledge.

This work raises opportunities for platform design in expert crowdsourcing. Foundry draws inspiration from visual workflow tools [24, 27] and management tools such as Gantt charts to structure teams using a visual timeline language. It is possible to design for worker interest, honesty, and motivation (e.g., [3, 15]); Foundry thus aims to make collaborators visible and make the larger goal clear. Software concepts such as class hierarchies can integrate with business processes [34]: Foundry instantiates this vision with team structures.

Organizational behavior

Organizational behavior (OB) research has identified the obstacles to team coordination as well as the conditions and structures that enable effective team coordination. Obstacles to effective team coordination include geographic dispersion, technology-mediated communication, and fluid (or changing) team membership [39, 19, 11, 22]. These conditions impede team coordination because team members lack a shared understanding of their overall task, and each person's responsibilities within the task [36]. Because their communication is all electronically-mediated, teams also struggle to engage in the rich communication necessary to build a shared understanding [11]. Teams of experts who are pulled from the crowd will face these same obstacles in the extreme.

Foundry's design is inspired by specific OB studies that show how to support coordination under the challenging working conditions confronted by crowdsourced expert teams.

First, Foundry's composeable team structures are motivated by OB studies of the purpose and functioning of team structures. Team structures enable coordination by encoding who is responsible for what work and which team members are interdependent [8]. Even in the absence of ongoing relationships, lightweight team structures can help even relative strangers coordinate effectively, when the structures set up shared space and shared work around strictly defined work roles [41]. Second, Foundry's modular design is influenced by management modularity theory which shows how system components must be "loosely coupled" and have a standardized interface so that they can connect and interact in a variety of configurations [4]. Third, Foundry's design includes several integration mechanisms such as pipelining, structured handoffs and DRIs that address key points of coordination (which OB research shows are often neglected) [18].

We argue that for teams of crowdsourced experts to be most successful, they need to quickly understand their shared work, interdependencies and respective roles in completing that work [37]. Our aim is to provide the value of team structures, modularity, and coordinating mechanisms (as identified in OB research), but to do so in ways that leverage automation, computation, and the scale and flexibility of the crowd.

FLASH TEAMS

In this section we introduce *flash teams*, which are computationally-guided teams of crowd experts supported by lightweight, reproducible and scalable team structures. Flash teams aim to embed the techniques of high performing offline teams within a model that can take advantage of computation's ability to abstract, scale, and visualize progress. We take a position that expert crowd work can succeed via a linked set of modular tasks that interface with each other by publishing intermediate results. Each set of tasks is machine-understandable, which means that interactive systems can manage and manipulate the team structures so that flash teams can grow, adapt, and recombine into larger organizations.

Flash team composition

To structure expert crowd work, flash teams require small atomic actions called *blocks*. Each block represents one or more experts performing a task. It requires an input, instructions for what one or more members of the crowd should do, and specifies an output type (Figure 2). Blocks can then be connected to other blocks that can accept compatible inputs or outputs. Blocks aim to be self-contained so that they can be reused in other contexts. Inputs and outputs are represented as tags (e.g., `heuristic evaluation report`, `character art`, `voiceover script`, `voiceover audio files`), and any blocks that interface along the same input or output use the same tag. For example, suppose the user is building a team to execute the user-centered design process. Then, they might create a heuristic evaluation block to take `low fidelity prototype` as input and produce `heuristic evaluation report` as output. That user could then connect the heuristic evaluation block to any other block that takes `heuristic evaluation report` as input, for example a block that revises a mockup based on feedback.

Flash teams exhibit distributed leadership [33] where the user (requester) may be coordinating with different workers in each block. To avoid a diffusion of responsibility, for each block, the project requester specifies a *directly responsible individual* [31], or DRI, who takes on a temporary management position for the duration of that task. For example, if a task involves a collaboration between three software developers, one of those developers acts as the DRI and coordinates workers, brokers agreements, and ensures the task is completed on time and at high quality.

Users assemble and manage a flash team by chaining blocks like interlocking puzzle pieces to transform a starter input into a desired final output. Blocks can execute in parallel or in serial but they typically must wait on all their inputs to begin work. The complete set of blocks determines which crowd experts need to be recruited — for example, a UI designer,

Figure 2: The basic unit of a flash team is a *block*, which gathers one or more paid experts from the crowd to perform a focused task. Blocks can be connected via shared inputs and outputs.

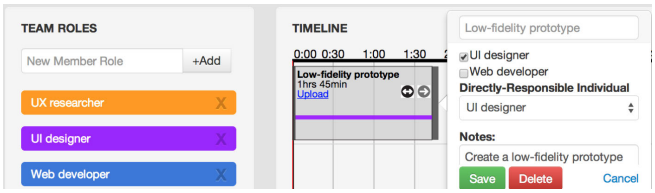


Figure 3: Foundry allows end users to author and manage flash teams. In authoring mode, the user can specify the expertise for each block and enter details about each block's requirements.

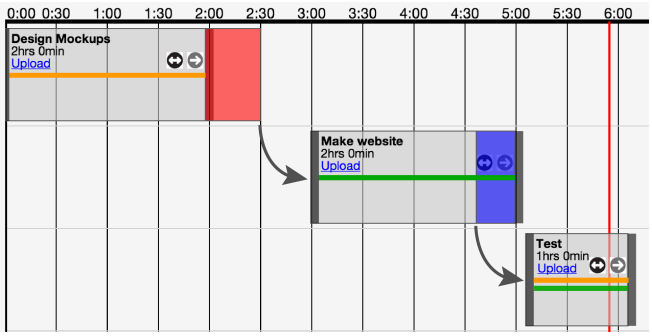


Figure 4: Foundry's runtime mode: the design mockups finished 30 minutes late but implementation finished early. The current user test block, marked by a red playhead, involves both (orange and green) experts.

UX researcher, and two software engineers — and when they are likely to be needed.

Foundry. Foundry (Figures 3–4) functions as both a flash team authoring environment with strongly typed handoffs and support for diverse expertise, and a runtime management platform. Users add blocks to the timeline as they would on a calendaring interface. Each block requires a title, a target length of time, input and output type tags, a description of the task (ideally with an example), and one or more of roughly 2,500 skills available on oDesk. To accelerate team creation, and based on an observation that many teams share some components, Foundry makes available a library of all blocks that previous teams have used for drag-and-drop. With Foundry, therefore, the requester can create new workflows from scratch using the library of task modules, fork a previously created workflow, or use Foundry's planner to formulate a workflow.

Most coordination between team members occurs via input/output handoffs. Foundry focuses the user on explicitly marking handoffs via arrows that connect tasks: this helps make dependencies visible. When each block is complete, worker(s) upload the output materials for their block to the team's shared folder, which Foundry automatically creates in Google Drive. When two blocks occur at the same time (e.g., a logo design block should be coordinating with the homepage layout block), users indicate that the blocks should be in synchronous collaboration.

Runtime and coordination

Users either author a team from scratch, or borrow from a list of existing teams and edit or run them. Workers are recruited from oDesk using the skills described in the blocks. oDesk provides a worker rating system that makes filtering relatively simple. We have recruited workers for teams in as little as fifteen minutes, and this number could decrease as such platforms become more popular and optimized. When the user is happy with their team, they launch the team by clicking "Start" in the Foundry interface. Foundry switches to runtime mode and adds a timer playhead that starts advancing along the timeline on the first block. Each worker receives a unique link that logs them in to the Foundry runtime with their tasks highlighted in the timeline.

Foundry embeds the basic affordances of computer-supported cooperative work systems for distributed teams. While the flash team is working, Foundry's goal is to remove the requester from as much management responsibility as possible by leveraging what it knows about the flash team's structure. Requesters can monitor progress, provide feedback, and answer questions through the chat. They can also pause the process, tweak the team structure and resume.

Foundry always displays the overall progress through the timeline, the currently active block details, and when the logged-in worker's next task begins. When a worker marks a block as complete, Foundry notifies workers for the next blocks that their task is active. Additionally, Foundry maintains awareness through a chat window that allows workers to coordinate out-of-band and the user to provide feedback on the process as it occurs.

It is not uncommon for teams to require tweaks as they go: for example, a task may finish early or run late. If a task finishes early, Foundry recalculates the starting times for all downstream tasks and emails all impacted workers with a new estimated start time for their next task. Likewise, if a task runs late, Foundry emails the DRI with a link to a form that requests the estimated time to completion. When the worker fills out the form, downstream tasks are again recalculated and workers are notified of the new start times.

Computationally-enhanced flash teams

Computation can leverage the machine-readable workflow representation of flash teams. For example, we can use a block's output tag to recommend a list of blocks that use the same tag as an input. In this section, we introduce mechanisms for Foundry to react, optimize and author flash teams on demand.

Modular combinations of teams

Because the blocks are modular and have clear boundaries, flash teams can be *combined* to create larger organizations. Entire teams can be abstracted into a block, much like one function in a computer program might contain many other function calls inside it. For example, to create a new platform for a massive open online course (MOOC), one flash team might combine several teams that can design and implement software with additional teams that can develop educational materials and animated videos. Each team might be forked from previously used teams in Foundry and tweaked for the particular needs of the project.

Through combination, users can create not just teams but the equivalent of entire small organizations for an afternoon or a day. These combinations draw on the elastic nature of the

crowd by instantiating multiple copies of a flash team, and multiple types of teams, with little downtime.

Path search support for team authoring

So far, Foundry assumes that end users are fluent with creating teams (e.g., authoring tasks and handoffs). Most people, however, are neither natural programmers nor managers. Instead, they have a goal in mind that they want to accomplish, but may not have a strong idea of how it might get completed.

For example, suppose a user wanted to embed themselves as a character in their favorite video game. The user might have no experience with art or video game modding, but have a `photograph` of themselves and knows that they want a `3D model` to replace the default game avatar with. Using this, the system explores the set of existing teams and realizes that the `photograph` can be used as `source material` for a fashion designer from a theatre production task who creates a `clothing sketch` according to the world's style. The `photograph` and `clothing sketch` can then be passed to an illustrator from an eBook task, who in turn creates a `2D drawing` of character art. The `2D drawing` is then used by a 3D modeling expert originally from a product mock-up team to create a `3D model` which can be imported into the game.

By leveraging blocks' shared input/output tags, Foundry can search through the space of novel team combinations. If the user knows what they want, they can tell Foundry their starter input (e.g., `character sketch`) from the list of previously-observed tags and the output they want (e.g., `3D model`). If there is a path through the known blocks that can solve the user's problem, Foundry will find it using an AI planner. The resulting teams are not whole components of previous teams, but subsets of blocks from many other teams that come together to enable a new goal. The user can review the team to ensure that all team blocks make sense in this new context.

This team was automatically generated on demand to fill in details that the user may not have been able to specify themselves. The members of the team were not drawn from prior teams with a focus on video games, but Foundry identified that previous blocks shared an input/output vocabulary and could thus interface with each other. It brought them together to effectively complete the desired task.

Because of teams' modularity, the number of possible teams grows combinatorially with the number of teams seen by the system. Foundry thus uses a planning algorithm to find a set of blocks that connect from original input to final output, if such a path exists. We represent all known blocks in the STRIPS planning language [17]. Each block becomes a STRIPS action with all required inputs combined via a logical conjunction as the precondition, and all outputs listed as postconditions. An automated planner searches through transitions that connect a given starting state to a desired end state in order to find the shortest path. The general planning problem is NP-complete; however, boolean satisfiability solvers can scale these systems to thousands of actions [23].

Elasticity

Simple workflows cannot react to changing requirements: for example, a project may require additional help or a new skill. Because crowdsourcing platforms allow quick recruitment, the crowd can be an *elastic* resource, allowing teams to grow and shrink as needed. Elastic growth allows a single team abstraction to encompass a wide variety of actual runtime needs.

To enable elastic flash teams, the user specifies growth parameters for a block. The block may, for example, now allow up to two additional experts to join this task. If the user or the DRI feel that growth would be helpful, they can request the expansion and the user can approve it. One type of elasticity is replication: for example, the DRI for a software development task may ask for an additional developer to help accomplish the job on time. The second type of elasticity is specialization: if a team needs a specific skill, the DRI could recruit a team member with that skill (e.g., logo design). For instance, in one of the design teams, the team requested a Facebook API expert because the application required it. Users need to balance the temptation of growing a team against the increased coordination costs of larger groups.

Pipelining

Flash teams have been described as blocking operations: downstream tasks must wait for upstream tasks to complete before they can start. However, in many scenarios, early results may be enough for a downstream task to begin.

Flash teams can thus be *pipelined* by streaming intermediate results as they are ready, rather waiting for the entire task to complete. To pipeline a handoff, the user indicates that a block can stream its output — for example, that the heuristic evaluation block can stream each violation as the worker identifies them. The user also indicates that the downstream task can accept streamed results — for example, that the revision of the low-fidelity mockup can fix each violation as it arrives. The DRI for the upstream block uses Foundry to indicate when the task is ready to start streaming, and Foundry launches the downstream task.

Pipelining has two beneficial effects: less waiting and increased collaboration. There is a risk that downstream tasks might need to change course when they see more of the input, but in our experience pipelining scenarios allow team members to work productively in parallel. In addition, instead of communicating through a single handoff, pipelining encourages synchronous feedback between team members, reducing the risk of communication failures or misunderstandings.

EXAMPLE FLASH TEAMS

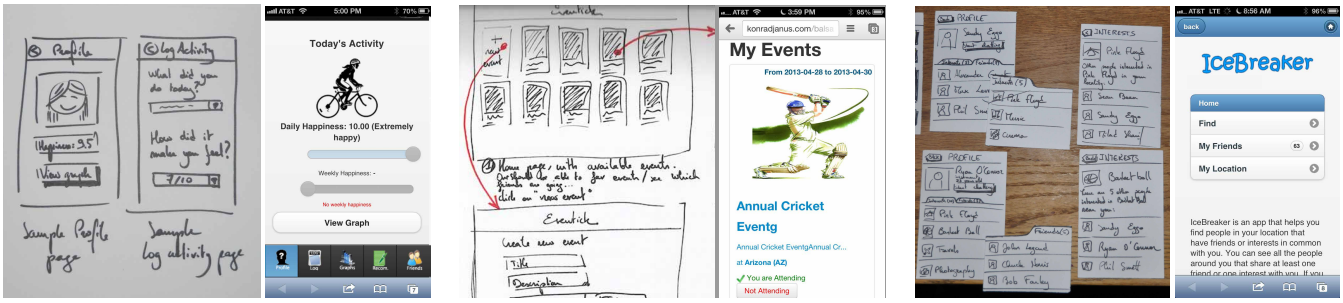
In this section, we present several flash teams and their results. We demonstrate how flash teams execute complex and creative tasks beyond the reach of paid microtasking and surpass the capabilities of traditional organizations via their scalable and flexible structure.

Napkin Sketch Design Team

Rapid iteration is a core tenet of the user-centered design process, and prototyping many ideas in parallel leads to improved design outcomes [16]. Flash teams could enable a designer to generate many ideas as sketches in a notebook, then get them prototyped in parallel — from low-fidelity prototypes, through a heuristic evaluation and live user testing, to

Task	Type of Team	Roles	Time [*hh:mm*]	Median Wage	Total Cost
Happily App	Napkin Sketch	UX, UI, Dev	31:30	$26.85	$744.48
Eventick App	Napkin Sketch	UX, UI, Dev1, Dev2, Dev3	18:00	$28.78	$1,270.28
Icebreaker App	Napkin Sketch	UI, UX1, UX2, Dev1, Dev2	23:10	$31.38	$1,200.97
Animation	Animation	Director, Voiceover, Animator, Illustrator, Scriptwriter	44:40	$56.94	$2,381.32
Diaphragm MOOC	Education Content, Animation	Education, Content, Director, Animator, Actor, Voiceover	19:20	$30.14	$1,597.32
Photography MOOC	Education Content, Animation	Education, Content, Director, Animator, Actor, Voiceover	19:00	$21.77	$741.58
Tower of Hanoi MOOC	Education Content, Animation	Education, Content, Director, Animator, Voiceover	11:30	$18.52	$446.49
MOOC Platform	Napkin Sketch 1, Napkin Sketch 2, Napkin Sketch 3	UX1, UI1, Dev1, UX2, UI2, Dev2, UX3, UI3, Dev3	13:00	$29.14	$1,015.80

Table 1: Eight flash teams created mobile web applications, video animations, and an online course platform with three courses.



(a) Happily, an emotion-tracking site. **(b)** Eventick, a local event billboard site. **(c)** Icebreaker: a local social network site.

Figure 5: Sketched inputs and high-fidelity prototype results from the napkin sketch design teams. This team iterates from a sketch through low fidelity prototypes to produce a working, user-tested software prototype within a day.

an iterated high-fidelity prototype — in less than a day. This rapid, parallel cycle could allow end users to quickly iterate toward high-quality ideas.

The *napkin sketch design team* carries out the entire user-centered design process starting from a napkin sketch of the idea by combining techniques from HCI practice. Rather than acting as a turn-crank design solution or replacing designer insight, it enables designers to experiment and reflect on ideas quickly. The complete napkin sketch design team blocks are:

- *Low-fidelity prototype*. Produce a mockup of the sketched interface using a low-fidelity wireframing tool. Input: `napkin sketch`. Output: `low-fi mockup`. Role: UI designer. Target time: 2.5hr.
- *Heuristic evaluation*. Report the violations of Nielsen's heuristics for a design. Input: `low-fi mockup OR high-fi prototype`. Output: `heuristic evaluation`. Role: UX researcher. Target time: 2hr. Can pipeline output.
- *Revised low-fidelity prototype*. Iterate on a mockup based on usability feedback. Input: `low-fi mockup AND (heuristic evaluation OR user study report)`. Output: `low-fi mockup`. Role: UI designer. Target time: 4hr. Can pipeline input.
- *Software prototype*. Create a hosted, high-fidelity web prototype to instantiate a mockup design. Input: `low-fi mockup`. Output: `high-fi prototype`. Role: Web developer (elastic 1-3).
- *User study*. Run at least three users through a prototype and create a document that summarizes the results and recommendations. Input: `high-fi prototype OR low-fi mockup`. Output: `user study report`. Role: UX researcher. Can pipeline output.
- *Revised software prototype*. Fix problems with the prototype that were identified by the user study. Input: `high-fi prototype AND user study report`. Output: `high-fi prototype`. Role: Web developer (elastic 1-3). Can pipeline input.

Depending on whether the same workers participate in multiple blocks or new workers come online for each step, this team involves three to ten experts in UI design, UX research and web programming.

We ran the napkin sketch design team three times with inputs of varying complexity, all for prototypes of mobile web applications (Figure 5). We obtained the original application ideas and sketches from actual student projects in an Introduction to HCI course. The first design sketch was for Happily, a web application that helps users track their happiness throughout the day. The second was for Eventick, a mobile application where users can post local events and hear about events in their area. The third was for Icebreaker, a social networking site to help users find nearby individuals with common interests. These prototypes each contained roughly four working pages, a database backend, and content-specific requirements (e.g., Facebook API integration for Icebreaker).

Each flash team took advantage of more of flash teams' computational support than the previous iteration. Happily used a basic, sequential workflow, where one task occurred at a time and the next task could not begin until the previous one was completed. Eventick enabled pipelining: both heuristic evaluation and development began as soon as the first version of the mockup was complete. It also enabled elastic growth for replication, allowing the web development block to grow from one to three developers. Icebreaker was the most com-

plex, enabling elastic growth for specialization: the DRI developer hired a developer specialized in Facebook API integration, and the team brought on a designer to create a logo for the homepage. Icebreaker also pipelined the user study results into the revised software prototype block.

These flash teams completed their prototypes in as little as eighteen hours of active work (Table 1). Workers were typically paid $25–$30 per hour, and costs ranged from roughly $750 to $1270 for up to five participants. The prototypes strongly reflected the designs articulated in the original napkin sketches, and many major design and usability problems had been ironed out through the heuristic evaluation and user study revisions. The initial napkin sketches and final prototypes are shown in Figure 5. We did not optimize the team to hand off across timezones, so actual wall clock time was often longer as team members needed to sleep and take breaks.

Animation Team

A quick review of YouTube videos is enough to realize that many people who have excellent ideas are not quite so excellent at communicating them. Unfortunately, for most of these people, hiring an expensive firm to bring their ideas to fruition is the only alternative. The animation flash team aims to make high quality video animations more broadly accessible. It also exercises flash teams in a creative, non-engineering domain. The animation team takes a high-level script outline as input and produces a short animated movie as output. The core blocks of the animation team (Figure 6) included:

- *Script*. Write the video script. Input: `script idea`. Output: `script`. Roles: director, scriptwriter. Target time: 2hr.
- *Storyboard*. Use the script to create a shot-by-shot storyboard for the video. Input: `script`. Output: `storyboard`. Roles: director, animator. Target time: 2hr.
- *Character design*. Design the characters' visual appearance and develop their art assets. Input: `script`. Output: `character assets`. Role: illustrator (elastic 1-3). Target time: 4hr. Can pipeline output (draft/final designs).
- *Character animation*. Import and animate character assets from the storyboard. Input: `storyboard AND character assets`. Output: `animation character layer`. Role: animator (elastic 1-3). Target time: 7hr. Can pipeline input.

To develop the flash team structure, we collaborated with a movie director from oDesk. The animation flash team consists of five roles: director, scriptwriter, illustrator, animator, and voiceover artist. Unlike the decentralized leadership of the napkin sketch team, the director role participates actively across the entire workflow. In the animation team we ran, the task was to create an animation that captures a memory from HCI researcher Terry Winograd's early life, where Terry builds a computer in his garage (Figure 7).

Education team

While many people step into the role of teacher at times, most do so without aid or feedback from trained educators. In addition, while the web can teach a wide variety of concepts, there are always concepts that are not well covered or not explained clearly where a user might want some explanation. The education flash team allows anyone to generate a short curriculum to learn a topic. It takes as input the educational goal (e.g., "teach the basic mechanics of singing from the diaphragm") and outputs a script that teaches the subject and multiple-choice test questions to self-test the content. The main blocks of the education team are:

- *Script*. Brainstorm and write out a script for the lesson. Input: `lesson idea`. Output: `lesson script`. Roles: educator, topic area expert. Target time: 3hr.
- *Quiz*. Generate a quiz based on the content of the lesson script. Input: `lesson script`. Output: `lesson quiz`. Roles: educator, topic area expert. Target time: 1hr.
- *Entertainment value script improvements*. Edit the script so it is as engaging as possible. Input: `lesson script`. Output: `edited script`. Role: director. Target time: 1hr.

The team draws together three sets of skills: expertise in the topic area of the lesson, curriculum development, and multimedia direction. The director takes a final pass over the script to ensure that it is as engaging as possible.

An example script snippet from a flash team generating a lesson on singing from the diaphragm: *"When you breathe in, or inhale, your back muscles engage and expand. At the same time, your diaphragm lowers so your lungs can expand."*

Composite team: online course platform

The napkin sketch, animation and education teams are relatively focused in their goals. Could flash teams enable users to pursue more ambitious goals that combine multiple domains of expertise? Flash teams' modularity promises to allow for organizations composed of many smaller teams.

We created a composite flash team that combined and replicated all three other teams. Its goal was to create a prototype platform for online education of long-tail skills such as photographing portraits, singing from the diaphragm and solving the Tower of Hanoi puzzle (Figure 8).

The composite team used abstracted versions of existing teams as single blocks, then tweaked their outputs and inputs. The blocks of the composite team included:

- *Homepage*. Design homepage for the site. Input: `napkin sketch`. Output: `high-fi prototype (homepage) AND design language`. Roles: napkin sketch team. Target time: 12hr. Can pipeline output (design language).
- *Class page*. Design the page to display a course. Input: `design language AND napkin sketch`. Output: `high-fi prototype (course)`. Roles: napkin sketch team. Target time: 12hr. Can pipeline input (design language).
- *Course content*. Write a script and quiz for the video. Input: `lesson idea`. Output: `multimedia-edited script`. Roles: education team. Target time: 4hr.
- *Course video*. Produce the video for the lesson. Input: `multimedia-edited script`. Output: `animation`. Roles: animation team. Target time: 16hr.

This team was composed of nine flash teams: three napkin sketch teams to create the three pages of the site, three education teams to write the video scripts and quizzes, and three animation teams to produce the videos. The first napkin sketch

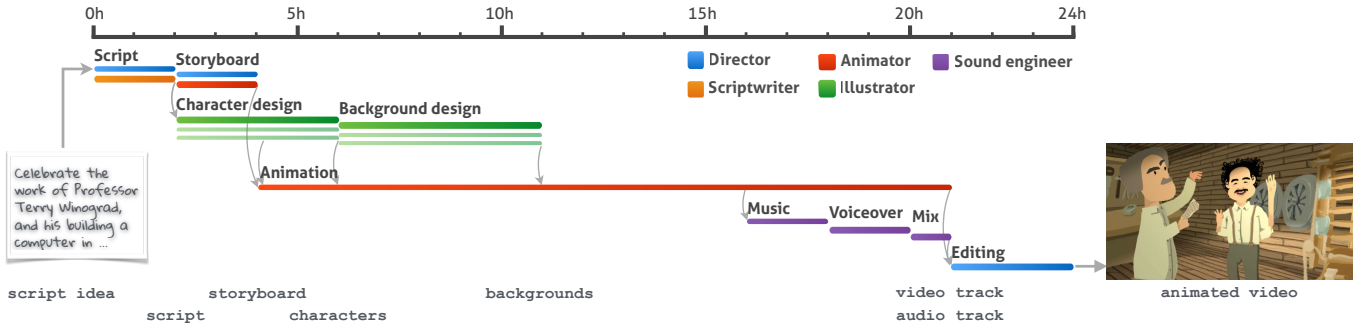


Figure 6: The workflow for the animation flash team, which takes a high-level script outline as input and produces a short animated movie as output.

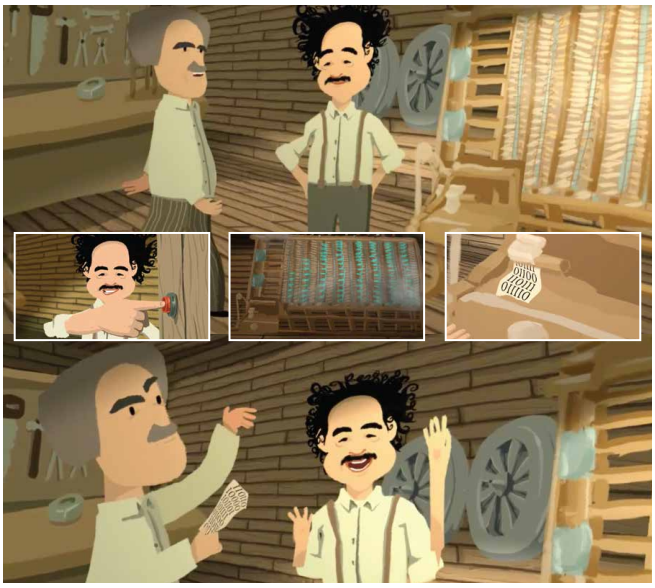


Figure 7: The animation team produced a short movie of a young Terry Winograd building a computer in his garage.

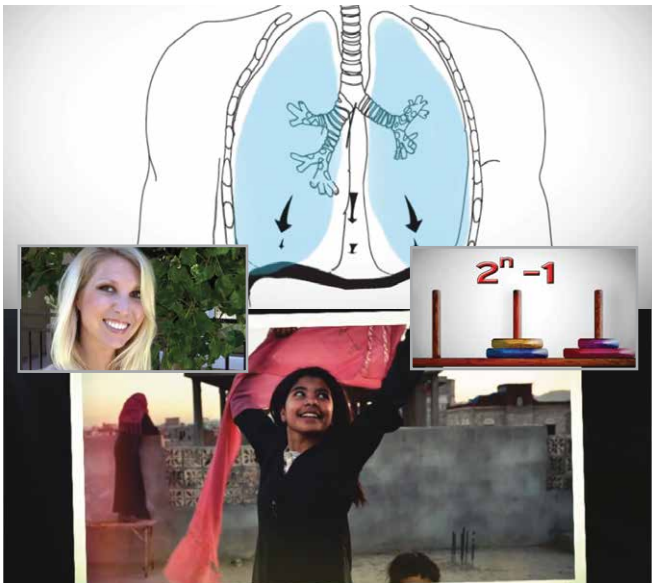


Figure 8: Course videos: singing from the diagram (top), portrait photography (left and bottom), and the Towers of Hanoi puzzle (right).

team pipelined the site design language. The nine teams created the platform and three interactive lessons in *19 hours and 20 minutes* of active work for a budget of $3,801.19 (Table 1). Wall clock time was longer because not all teams began simultaneously. The talent for the singing video included a professional singer from the Frankfurt Opera.

FIELD EXPERIMENT

This paper claims that flash teams enable expert crowd members to come together and complete complex, interdependent tasks in short periods of time. We conducted a controlled experiment to compare flash teams to a self-managed team approach where Foundry brings together the crowd experts and lets them choose their own path. To minimize costs, we focused on collecting data with a small number (3) of teams in each condition, totaling 22 crowd experts.

Method

We recruited six napkin sketch design teams from oDesk by requesting workers with expertise in UI design, UX research, and web application development. Teams were assigned to either the flash team or the control condition, resulting in a total of three teams in each condition. We used the same job postings on oDesk, recruitment process and task for the teams in both conditions. To keep conditions comparable, the same set of Foundry features was available, including a shared folder via Google Drive and a chat window.

To recruit the experts, we conducted brief filtering interviews and accepted applicants with high ratings. All workers were paid their profile's hourly wage. Members of each team had a median of at least 250 working hours on oDesk and ratings over 4.2 out of 5 stars (average 4.66/5). We randomized the 22 workers into teams and tasked them with creating a simple party planning task manager from an input sketch that would allow end users to add, edit and complete tasks. We asked all teams to follow the user-centered design process and requested that the final mobile web app be completed within 13 hours.

The main difference between the two conditions was the existence of the flash team workflow, whose presence indicated a path to the team members, and allowed Foundry to support coordination via handoffs and notifications. For the flash team condition, Foundry clearly specified each block and the details associated with it, including the duration, team member, DRI, input and output. This enabled Foundry's support for handoffs, task status, time until next task, and email noti-

Condition	Roles	Work Time [*hh:mm*]	Wall Clock [*hh:mm*]	Median Wage	Total Cost
Flash	UI, UX1, UX2, Dev	11:02	17:16	$18.00	$211.78
Flash	UI, UX, Dev	13:52	24:10	$22.22	$364.80
Flash	UI1, UI2, UX1, UX2, Dev	14:13	23:51	$14.45	$215.58
Control	UI, UX, Dev1, Dev2	14:30	37:03	$25.00	$322.22
Control	UI, UX, Dev	29:20	12:24	$16.67	$612.79
Control	UI, UX, Dev	27:30	37:01	$12.00	$430.59

Table 2: Time and cost comparison for the flash and control teams. On average, flash teams took half as many work hours than control teams.

fications. In contrast, for the control condition, the workflow on Foundry consisted of one long task and did not provide any details or information on how to complete it. This team was allowed to self-manage and make their own plan to complete the work in 13 hours.

Once each project started, we monitored the chat on Foundry to make sure the team was working and answer any questions that came up. We were equally responsive to the progress and questions of the experimental and control teams. If a team member left, we hired a replacement on oDesk immediately. Though we told teams to complete in 13 hours, we allowed teams as much time as they needed to complete the task.

Results

While all six of the teams completed the task and their applications were above-bar for quality, the flash teams took roughly *half as many work hours*, followed the iterative design process more closely, and required less coordination.

By active work time, even the *slowest* flash team completed the task faster than the *fastest* team in the control condition. As shown in Table 2, the cumulative number of hours worked for the flash teams ranged from 11hr2min to 14hr13min (mean = 13hr2min) and had little variance across teams (SD = 1hr45min). This small variance was consistent across all blocks: the average standard deviation across the six blocks' completion times was twelve minutes. In contrast, the cumulative number of hours worked for control teams ranged from 14hr30min to 29hr20min (mean = 23hr47min) and had tremendous variation (SD = 8hr5min). A one-tailed permutation test to compare the completion times in the two conditions was significant ($p = 0.05$), confirming that flash teams are significantly more efficient than self-managed teams of crowd experts. Wall clock time did not always align with total work time, since workers would sleep or take breaks.

The control teams took longer than the flash teams to finish each role's tasks (Table 3). The control teams spent 2.4x the time on UI Design, 1.9x the time on UX Research and 1.4x the time on Development, resulting in an additional 10hr44min in cumulative work. In observing the teams, we noted that team members would often invent work to do while waiting for a collaborator to finish a task. This overeager behavior, which we term *busy waiting*, particularly affected team members such as UI Design who had fewer hours of work that were officially required of them.

By following the sequence of modular tasks on Foundry, flash teams followed the iterative design process more closely than

Role	Flash Team [*hh:mm*]	Control Team [*hh:mm*]
UI Design	2:59 (σ=0:55)	7:20 (σ=3:39)
UX Research	3:42 (σ=0:33)	7:07 (σ=2:52)
Development	6:21 (σ=0:46)	9:20 (σ=2:01)
Total	**13:02 (σ=1:45)**	**23:47 (σ=8:05)**

Table 3: Average work time by role for the flash and control teams. The flash teams finished the tasks for each role faster than the control teams.

the control teams. In theory, since the self-managed teams weren't restricted to a precise workflow, they could have iterated more times than the flash teams. However, these teams produced fewer, if any, iterations on the mockups or developed web app. The control teams inefficiently decomposed the tasks, combining separate tasks into a single task (such as performing the heuristic evaluation and user testing on the same mockup) and in the wrong sequence (for example, the summative user testing was executed on the low-fidelity mockup, and the software prototype was never evaluated).

The flash teams required less coordination than the control teams and were more able to take advantage of on-demand recruiting from the crowd. The control teams' success was largely dependent on individuals' project management skills. For example, one of the control teams did not coordinate at all and found the experience extremely frustrating. Another control team adopted a fully interdependent workflow in which the experts treated Foundry as a collaborative war room environment [35] and established social and informational awareness to support their coordination efforts [37]; they all worked together to produce a high-quality outcome at great financial cost. When workers disappeared due to other commitments, flash teams were robust and could reach out to the crowd for a replacement quickly. In contrast, when members of one of the control teams quit (in multiple occurrences, they grew too frustrated with the experience and left), the entire team's performance suffered. When one of the developers was replaced, for example, the new developer had to start from scratch and was forced to work on his own since the other team members had disappeared by that point.

DISCUSSION AND FUTURE WORK

Flash teams enable users to gather and coordinate paid experts from the crowd to complete complex and interdependent tasks quickly and reliably. Rather than try to recreate the strengths of in-person expert teams, this approach suggests a "beyond being there" vision of expert crowd work [20]. In particular, flash teams afford dynamic recruitment and coordination of on-demand expertise that is extremely difficult in offline scenarios. Problems that might have plagued such teams, such as last minute dropouts, are relatively straightforward to solve in crowd work because another person can join on demand. Furthermore, flash teams can take advantage of timezone differences that could potentially allow them to carry on uninterruptedly for days or weeks.

Flash teams move beyond the typical conception of crowdsourcing as collecting multiple viewpoints on a single question. Rather than treating the crowd as redundant resources that cannot be fully trusted, flash teams view the crowd as an elastic, on-demand set of diverse and high-quality participants. The result is that flash teams often aim to gather

experts with *different expertise* rather than redundant viewpoints. Even more ambitiously, flash teams can be combined to create new types of organizations with completely fluid boundaries — organizations that are composed of many smaller flash teams, each of which are spun up on demand, work in parallel, and disperse when complete.

Flash teams also have several important limits, which we intend to address in future work. While our controlled evaluation of flash teams was limited to napkin sketch design teams, in the future we plan to test other types of flash teams to better understand the types of tasks flash teams are well suited for. Furthermore, similar to traditional collocated and distributed teams, flash teams experience difficulties related to coordination and conflict. During informal interviews after the tasks were completed, flash team members indicated that they enjoyed working as a team, but wanted the opportunity to build more camaraderie — some members would finish their task and then just leave. One approach to improving team motivation would be to allow workers to find trusted colleagues and join into clusters that can be hired together.

It will be important for follow-on evaluations to compare flash teams to the same teams when led by project managers. Our current objective was to test flash teams against self-managing teams, which are the equally scalable alternative and therefore theoretically relevant for our paper. Research in organizational behavior has shown that self-managing teams are prevalent and effective. However, it is likely that many teams in future environments may want to consider human managers, which is a worthwhile tradeoff to understand.

Given that teams do not always go according to plan, future work should also explore issues related to runtime course correction and dispute resolution. When this happened in practice, the flash team tended to work within the existing task structure to resolve the issue. However, Foundry could provide more built-in support for veering off the path if the user allows it, or even algorithmic guides that allow for branching or looping. In addition, flash teams have disagreements like any other team. In one case, miscommunication and disagreement about artistic direction provoked a heated argument between experts on the animation team, causing the director to fire the illustrator and find a replacement.

Currently, the recruitment of each flash team requires a vetting process on oDesk to pre-clear a set of workers who are high-quality and available at the desired time. However, platforms such as oDesk are moving toward more automated hiring procedures. We envision a future where a user could request an expert for a given wage and quality, the platform mediates to provide one either algorithmically or through crowdsourcing consultants. In the meantime, Foundry could build up lists of trusted experts for each flash team structure in order to provide a quick and trusted recruiting experience.

In the future, the crowd scale of flash teams could enable end users to make more data-driven decisions about collaboration and work, and empower the scientific study of teams. As the same flash team gets run multiple times, Foundry could display estimates for how long tasks tend to take in practice. Likewise, Foundry raises the opportunity to randomly perturb team structures at scale to run field experiments and A/B test collaboration structures.

CONCLUSION

This paper offers a vision of how computation and crowdsourcing can shape the future of creative, engineering, and analytical work. We introduce flash teams, which are linked sequences of modular goals for crowd experts that can be represented and interpreted by interactive systems. Flash teams benefit from computational authoring: end users can combine modular team elements to create larger organizations and generate teams on request through automated path search. Flash teams also benefit from computational management: they can grow and shrink on demand via elastic recruitment and pipeline results to accelerate completion times. Offline organizations can embed similar techniques into traditional teams, but flash teams open the door to doing so at a much larger scale than previously.

Flash teams offer an opportunity to shift the paid crowdsourcing narrative in both research and practice. Rather than aiming for redundant, independent judgments, flash teams envision a future of crowdsourcing with dynamic collaborations of diverse and interdependent participants. This future would enable traditional organizations to become far more reactive: a few end users with an idea, for example, could temporarily augment their team on demand as the need for certain skills and expertise becomes apparent. Ultimately, we aim to enable experts and amateurs alike to contribute skills they enjoy, on a set of tasks that they find interesting, and at a scale we are just beginning to glimpse.

ACKNOWLEDGMENTS

Thank you to the oDesk workers who participated. This research was supported by the National Science Foundation under grant no. 1351131, the Hasso Plattner Design Thinking Research Program, the Precourt Energy Efficiency Center, and oDesk.

REFERENCES

1. oDesk. URL `www.odesk.com`.
2. Ahmad, S., Battle, A., Malkani, Z., and Kamvar, S. The jabberwocky programming environment for structured social computing. *Proc. UIST '11*, 2011.
3. Antin, J. and Shaw, A. Social desirability bias and self-reports of motivation: a cross-cultural study of Amazon Mechanical Turk in the US and India. *Proc. CHI '12*, 2012.
4. Baldwin, C.Y. and Clark, K.B. Managing in an age of modularity. *Harvard Business Review*, 75(5):84–93, 1997.
5. Bechky, B.A. Gaffers, gofers, and grips: role-based coordination in temporary organizations. *Organization Science*, 17(1):3–21, 2006.
6. Bernstein, M.S., et al. Soylent: a word processor with a crowd inside. *Proc. UIST '10*, 2010.

7. Bigham, J.P., et al. VizWiz: nearly real-time answers to visual questions. *Proc. UIST '10*, 2010.
8. Bunderson, J.S. and Boumgarden, P. Structure and learning in self-managed teams: why bureaucratic teams can be better learners. *Organization Science*, 21(3):609–624, 2010.
9. Chilton, L.B., et al. Cascade: crowdsourcing taxonomy creation. *Proc. CHI '13*, 2013.
10. Cooper, S., et al. Predicting protein structures with a multiplayer online game. *Nature*, 466(7307):756–760, 2010.
11. Cramton, C.D. The mutual knowledge problem and its consequences for dispersed collaboration. *Organization Science*, 12(3):346–371, 2001.
12. Cranshaw, J. and Kittur, A. The polymath project: lessons from a successful online collaboration in mathematics. *Proc. CHI '11*, 2011.
13. Dai, P., Mausam, and Weld, D. Decision-theoretic control of crowd-sourced workflows. *Proc. AAAI '10*, 2010.
14. Dourish, P. and Bellotti, V. Awareness and coordination in shared workspaces. *Proc. CSCW '92*, 1992.
15. Dow, S., Kulkarni, A., Klemmer, S., and Hartmann, B. Shepherding the crowd yields better work. *Proc. CSCW '12*, 2012.
16. Dow, S.P., et al. Parallel prototyping leads to better design results, more divergence, and increased self-efficacy. *TOCHI*, 17(4):18, 2010.
17. Fikes, R.E. and Nilsson, N.J. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3):189–208, 1972.
18. Heath, C. and Staudenmayer, N. Coordination neglect: how lay theories of organizing complicate coordination in organizations. *Research in Organizational Behavior*, 22:153–191, 2000.
19. Hinds, P., Liu, L., and Lyon, J. Putting the global in global work: an intercultural lens on the practice of cross-national collaboration. *The Academy of Management Annals*, 5(1):135–188, 2011.
20. Hollan, J. and Stornetta, S. Beyond being there. *Proc. CHI '92*, 1992.
21. Hu, C., Bederson, B.B., Resnik, P., and Kronrod, Y. Monotrans2: A new human computation system to support monolingual translation. *Proc. CHI '11*, 2011.
22. Huckman, R.S., Staats, B.R., and Upton, D.M. Team familiarity, role experience, and performance: evidence from Indian software services. *Management Science*, 55(1):85–100, 2009.
23. Kautz, H.A., Selman, B., and Others. Planning as satisfiability. *Proc. ECAI '92*, 1992.
24. Kittur, A., Khamkar, S., André, P., and Kraut, R.E. CrowdWeaver: visually managing complex crowd work. *Proc. CSCW '12*, 2012.
25. Kittur, A., Smus, B., Khamkar, S., and Kraut, R.E. Crowdforge: crowdsourcing complex work. *Proc. UIST '11*, 2011.
26. Kittur, A., et al. The future of crowd work. *Proc. CSCW '13*, 2013.
27. Kulkarni, A., Can, M., and Hartmann, B. Collaboratively crowdsourcing workflows with turkomatic. *Proc. CSCW '12*, 2011.
28. Lasecki, W., et al. Real-time captioning by groups of non-experts. *Proc. UIST '12*, 2012.
29. Lasecki, W.S., et al. Real-time crowd control of existing interfaces. *Proc. UIST '11*, 2011.
30. Lasecki, W.S., et al. Chorus: a crowd-powered conversational assistant. *Proc. UIST '13*, 2013.
31. Lashinsky, A. *Inside Apple*. Hachette Book Group, New York, 2012.
32. Little, G., Chilton, L., Goldman, M., and Miller, R.C. Exploring iterative and parallel human computation processes. *Proc. HCOMP '10*, 2010.
33. Luther, K., Fiesler, C., and Bruckman, A. Redistributing leadership in online creative collaboration. *Proc. CSCW '13*, 2013.
34. Malone, T.W., Crowston, K., and Herman, G.A. *Organizing Business Knowledge: The MIT Process Handbook*. MIT Press, Cambridge, MA, USA, 2003.
35. Mark, G. Extreme collaboration. *Communications of the ACM*, 45(6):89–93, 2002.
36. Mathieu, J.E., et al. The influence of shared mental models on team process and performance. *Journal of Applied Psychology*, 85(2):273–83, 2000.
37. Nardi, B.A. and Whittaker, S. The place of face-to-face communication in distributed work. In P.J. Hinds and S. Kiesler (editors), *Distributed Work*, pp. 83–113. MIT Press, Cambridge, 2002.
38. Noronha, J., Hysen, E., Zhang, H., and Gajos, K.Z. Platemate: crowdsourcing nutritional analysis from food photographs. *Proc. UIST '11*, 2011.
39. Olson, G. and Olson, J. Distance Matters. *Human-Computer Interaction*, 15(2):139–178, 2000.
40. Stol, K.J. and Fitzgerald, B. Researching crowdsourcing software development: perspectives and concerns. *ICSE '14 Workshop on Crowdsourcing in Software Engineering*, 2014.
41. Valentine, M. Team scaffolds: how minimal in-group structures support fast-paced teaming. *Academy of Management Proceedings*, 2012.
42. Yu, L. and Nickerson, J.V. Cooks or cobblers? Crowd creativity through combination. *Proc. CHI '11*, 2011.
43. Zhang, H., et al. Human computation tasks with global constraints. *Proc. CHI '12*, 2012.