CrowdCode: A Platform for Crowd Development

Thomas D. LaToza[1], Eric Chiquillo[2], W. Ben Towne[3], Christian M. Adriano[1], André van der Hoek[1]

[1]Department of Informatics
University of California, Irvine
{tlatoza, adrianoc, andre}@ics.uci.edu

[2]Zynga
echiquil@gmail.com

[3]Institute for Software Research
Carnegie Mellon University
wbt@cs.cmu.edu

ABSTRACT

Microtask crowdsourcing organizes complex work into workflows, decomposing large tasks into small, independent tasks performed by workers who are assumed to be transient and possibly unreliable.

Applied to software development, this model might both increase the parallelism in development work and increase participation in open source development by lowering the barriers to contribute, enabling new economic models and allowing software to be constructed dramatically more quickly. However, microtask crowdsourcing typically assumes that the workflow can be specified in advance by the requestor. In software development, this assumption does not hold, as the structure, type, and content of tasks are much more dynamic. How then can such work be decomposed and coordinated as fine-grained microtasks?

Our key insight is to coordinate work through a graph of artifacts, generating microtasks in response to events that occur on artifacts rather than through an explicit workflow. Each microtask asks workers to perform a short well-defined task on a single artifact (e.g., a function or a test), allowing work to proceed on many artifacts in parallel. As workers complete microtasks, events are generated on the artifact, which may then trigger further microtasks to be generated. When an artifact changes, events are sent to artifacts that depend on it, allowing microtask structures to be dynamic and non-hierarchic. For example, when a function changes its signature (e.g., adding a parameter), artifacts that depend on it (callers and tests) are notified, generating microtasks to handle these changes. As artifacts may have many dependencies, artifacts may have multiple pending notifications of changes. To coordinate this work, each artifact has a microtask queue, allowing each change to be performed sequentially and ensuring changes do not conflict. Finally, this model supports iterative workflows. For example, developers editing a function can write *psuedocode*, iteratively generating microtasks until all of the pseudocode has been replaced.

Work first begins with a set of scenarios describing desired application behavior (provided by a requestor), spawning microtasks to edit a function to begin implementing each scenario. As workers edit functions, they may write *pseudocalls*, describing a function call they wish to see. After a microtask that checks to see if an existing function provides the desired behavior, a recursive step may occur, creating a new function and further microtasks. As functions are created, a microtask is generated to enumerate and then implement test cases for the function, introducing constraints between independently created artifacts that can be checked to ensure quality. When functions are completed, the tests are run; if tests fail, microtasks are generated to debug the function (or edit the tests) to enable the function to pass its tests. To allow workers to debug a single function in isolation, even if it calls other functions, workers can view and edit the return values of function calls, recursively creating new tests for the called function.

We have implemented our approach in CrowdCode, a prototype cloud IDE for crowd development. CrowdCode supports the development of Javascript libraries (e.g., Javascript code that takes an object as input and produces an object as output), which can be embedded into a web application. We are currently evaluating our approach by crowdsourcing the construction of a short program with a small crowd.