CrowdCode: A Platform for Crowd Development

Thomas D. LaToza1, Eric Chiquillo2, W. Ben Towne3, Christian Adriano1, André van der Hoek1

1 University of California, Irvine  2 Zynga  3 Carnegie Mellon University

CrowdCode

Build software with a crowd!

CrowdCode organizes work into microtasks, small, self-describing bits like writing pseudocode or brainstorming test cases. After you finish a microtask, CrowdCode figures out what to do next, generating and distributing microtasks to the crowd. So you might write a description for a function one, debug a test failure for another, and then edit the pseudocode the crowd wrote for function one to add a call. As you complete microtasks, you earn points, and can see how you’re doing on the leaderboard.

Writing Code

Edit a function 10 pts
Can you figure out how this user story should be implemented?

The main function is the entry point into the application. Below, sketch a design of this user story by writing the function’s description (the comments above the function header) and adding an implementation. Note that you should **NOT** implement everything in main, just instead use pseudocode (see below) to ask the crowd to create new functions or reuse existing functionality. Try to tease other user stories that may already be implemented. But don’t worry too much - it’s all tested.

If you’re not yet sure how you can do something, include a line or portion of a line as a `pseudocode` by writing a `//`. If you’d like to call a function, describe what you’d like to do with a pseudocode - a line or portion of a line beginning with `//`. Update the description and header to reflect the function’s actual behavior - the crowd will interface colors anything to match the new behavior. (Except if you’re adding the function `main` - you can’t change this function’s name or number of parameters, but you can still change its description)

Write a function description 10 pts
Can you write a description for a function that does this?

Here’s the description of the function to test:

```javascript
// Determine the ticket amount for a car or truck over the limit

function computeTicket(speed, weight) {
  return 100; // Ticket cost per mph over limit
}
```

Write a test case 10 pts
Consider the user story

Compute the amount of a traffic ticket for a car driving at (speed) mph in a (speedLimit) zone.

This user story is implemented by the function `computeTicket` called with a (speedLimit) zone. (speed) is (speed)

Get the expected output for the test case.

Simple Test Advanced Test

Parameter Values

<table>
<thead>
<tr>
<th>Input</th>
<th>Expected Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>speed</td>
<td>70</td>
</tr>
<tr>
<td>speedLimit</td>
<td>50</td>
</tr>
</tbody>
</table>

Testing & Debugging

Test cases

Add a call 10 pts
Can you replace this pseudocode with a call to this function, or find a different way to do it? Test free to update the code as necessary.

Debug a test failure 10 pts
This function has failed its tests. Can you fix it? To check if you’ve fixed it, run the unit tests. If there’s a problem with the tests, report an issue. You may use the function `printDebugStatement` to print data to the console.

Generating Microtasks

Key Simplifications

- Each function has attributes describing its state. Submitting a microtask may change a function’s attributes, generating microtasks.

- Only one microtask may be active per artifact, preventing merge conflicts. Events propagate changes (signature, tests) across dependencies between artifacts (functions, tests). Microtasks may “check out” a readonly copy of global interfaces, but may only commit a change to a single artifact. Events received on an artifact queue microtasks to be done.

- Each microtask resolves a single unit test. Microtasks cannot execute more than one test.

- Microtasks “check in” when the user finishes an artifact, not when the artifact is assigned.

- Only one artifact is active at a time, preventing merge conflicts.

- Workers are motivated by pay or reputation and not malicious.