

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

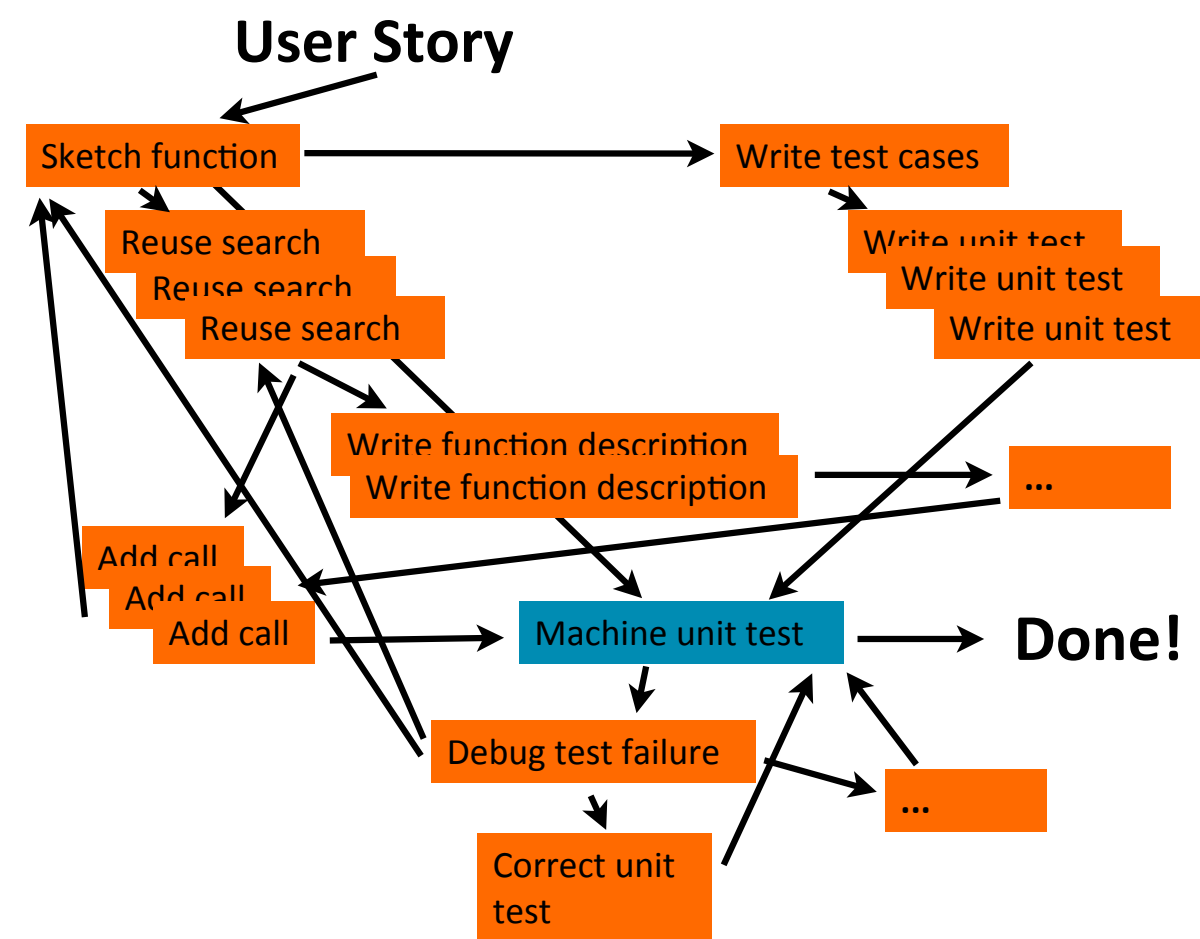
³ Carnegie Mellon University

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 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.

Let's get started!



CrowdCode
15 lines of code 1 functions written 8 microtasks completed 1 Bob

Your score ★
80 points

Leaders
80 Bob

Ask the Crowd
Bob how do I check if a variable is a valid number?

Edit a function 10 pts
Implement the function below.
If you're not yet exactly sure how to do something, indicate a line or portion of a line as **pseudocode** by beginning it with **///**. If you'd like to call a function, describe what you'd like it to do with a pseudocall - a line or portion of a line beginning with **///**. Update the description and header to reflect the function's actual behavior - the crowd will refactor callers and tests to match the new behavior. (Except if you are editing the function "main" - you can't change this function's name or number of parameters, but you can still change its description).

```
1 /**
2  Computes the amount of a traffic ticket for a car
3  moving at [speed] mph in a [speedLimit] zone.
4
5  @param { speed: NUMBER, speedLimit: NUMBER } input
6  @return NUMBER - ticket $ amount
7  */
8  function main(input)
9  {
10     /// check if speed and speedLimit are valid numbers
11
12     if (input.speed > input.speedLimit)
13     {
14         return computeTicket(input.speed - input.speedLimit);
15     }
16     return 0;
17 }
18
```

Recent Activity
You earned 10 points for editing a function!
You earned 10 points for writing test cases!
You earned 10 points for adding a call!
You earned 10 points for describing a function!
You earned 10 points for writing a test!
You earned 10 points for conducting a reuse search!
You earned 10 points for writing test cases!
You earned 10 points for editing a function!

Give us feedback on CrowdCode! What do you like? What don't you like?
Send feedback

Submit **Skip**

Help, I don't know Javascript!

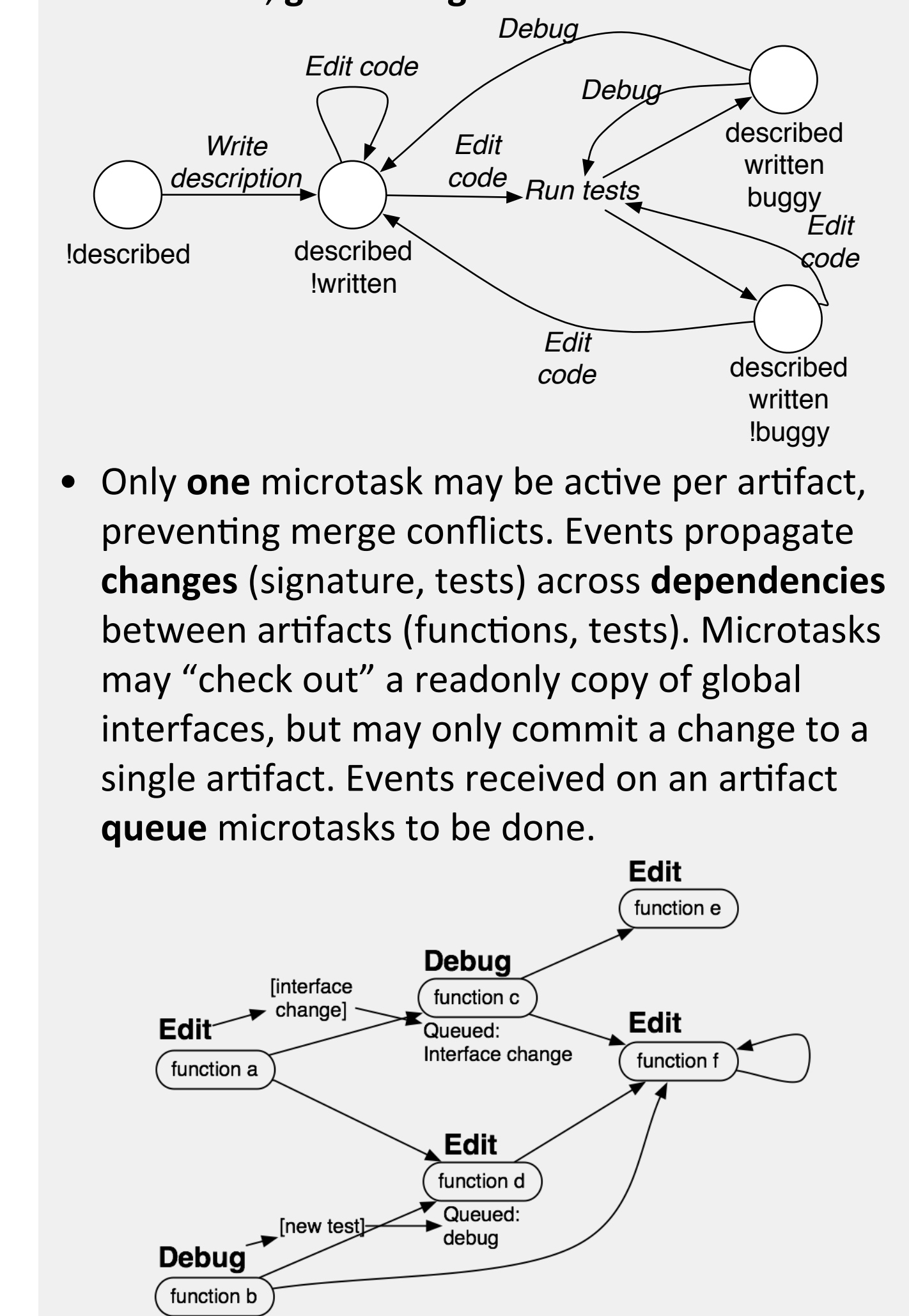
Key Simplifications

- Work begins with a set of **user stories** (scenarios) specified by a **client** which do not change.
 - Each user story can be tested by a set of tests of a main() function.
- Functions are completely specified by their **inputs and outputs**. (e.g., a library)
 - Functions do not mutate global state or interact with environment (e.g., write output).
- All bugs can be detected through unit tests.
- Programs are written in a (basic subset of) Javascript (e.g., no callbacks).
- Programming tasks are to **implement** a feature, **fix** a bug, or write **tests**.
- All **design** is done locally and iteratively (e.g., through refactoring).

=> crowdsourcing the programming of functional Javascript libraries

Generating Microtasks

- 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.



Writing Code

Edit a function 10 pts
Can you figure out how this user story should be implemented?
Compute the amount of a traffic ticket for a car moving at [speed] mph in a [speedLimit] zone.
The main function - the entrypoint into the application - is below. Sketch a design of this user story by editing the function's description (the comments above the function header) and sketching an implementation. Note that you should NOT implement everything in main, but instead use pseudocalls (see below) to ask the crowd to create new functions or reuse existing functionality. Try not to break other user stories that may already be implemented. But don't worry too much - it'll all be tested.
If you're not yet exactly sure how to do something, indicate a line or portion of a line as **pseudocode** by beginning it with **///**. If you'd like to call a function, describe what you'd like it to do with a pseudocall - a line or portion of a line beginning with **///**. Update the description and header to reflect the function's actual behavior - the crowd will refactor callers and tests to match the new behavior. (Except if you are editing the function "main" - you can't change this function's name or number of parameters, but you can still change its description).

```
1 /**
2  Computes the amount of a traffic ticket for a car
3  moving at [speed] mph in a [speedLimit] zone.
4
5  @param { speed: NUMBER, speedLimit: NUMBER } input
6  @return NUMBER - ticket $ amount
7  */
8  function main(input)
9  {
10     if (input.speed > input.speedLimit)
11     {
12         /// compute the ticket amount for a car n mph over the limit
13     }
14     return 0;
15 }
16
```

Reuse search 10 pts
Is there a function that does
compute the ticket amount for a car n mph over the limit
Use the search box to see if a function exists to do this. Otherwise, select "No function does this".
Show context
{ if (input.speed > input.speedLimit) { // compute the ticket amount for a car n mph over the limit return 0; }
If you can't find any, click this:
No function does this
Submit **Skip**

Write a function description 10 pts
Can you write a description for a function that
compute the ticket amount for a car n mph over the limit
Show example:
Hide context
{ if (input.speed > input.speedLimit) { // compute the ticket amount for a car n mph over the limit return 0; }
Computes a traffic ticket amount of 10 x mph over limit
returns number
function computeTicket ({ mphOver // Number mph over limit }
Add parameter
Submit **Skip**

Add a call 10 pts
The crowd found the following function for the **pseudocall** below:
Computes a traffic ticket amount of 10 x mph over limit
@param {Number} mphOver - mph over limit
@return {number}
function computeTicket(mphOver)
Can you either replace the pseudocall with a call to this function, or find a different way to do it? Feel free to update the code as necessary.

```
1 /**
2  Computes the amount of a traffic ticket for a car
3  moving at [speed] mph in a [speedLimit] zone.
4
5  @param { speed: NUMBER, speedLimit: NUMBER } input
6  @return NUMBER - ticket $ amount
7  */
8  function main(input)
9  {
10     if (input.speed > input.speedLimit)
11     {
12         return computeTicket(input.speed - input.speedLimit);
13     }
14     return 0;
15 }
16
```

Testing & Debugging

Write test cases 10 pts
Consider the user story
Compute the amount of a traffic ticket for a car moving at [speed] mph in a [speedLimit] zone.
This user story is implemented by the function main (description below). What are some examples of cases where this user story might occur? Are there any unexpected corner cases that might not work?
Computes the amount of a traffic ticket for a car moving at [speed] mph in a [speedLimit] zone.
@param { speed: NUMBER, speedLimit: NUMBER } input
@return NUMBER - ticket \$ amount
function main(input)
going 75 in a 50 zone
Add test case

Write a test 10 pts
Write a simple or advanced test for
going 75 in a 50 zone
Here's the description of the function to test:
Computes the amount of a traffic ticket for a car moving at [speed] mph in a [speedLimit] zone.
@param { speed: NUMBER, speedLimit: NUMBER } input
@return NUMBER - ticket \$ amount
function main(input)
Simple Test **Advanced Test**
Parameter Values
input: { speed: 55, speedLimit: 35 }
Expected Output
200

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 is a problem with the tests, report an issue. You may use the function `printDebugStatement(...)` to print data to the console.
Revert Code

```
1 /**
2  Computes the amount of a traffic ticket for a car
3  moving at [speed] mph in a [speedLimit] zone.
4
5  @param { speed: NUMBER, speedLimit: NUMBER } input
6  @return NUMBER - ticket $ amount
7  */
8  function main(input)
9  {
10     if (input.speed > input.speedLimit)
11     {
12         return computeTicket(input.speed - input.speedLimit);
13     }
14     return 0;
15 }
16
```

Run the Unit Tests
test: going 75 in a 50 zone
Error At: equal(main({ speed: 55, speedLimit: 35 }), 200, 'going 75 in a 50 zone');
Expected
200
Actual
30
Test case description going 75 in a 50 zone
Report Issue In Test
Computes a traffic ticket amount of 10 x mph over limit
@param {Number} mphOver - mph over limit
@return {number}
function computeTicket(mphOver)

Inputs
20
Outputs
30