Crowdsourcing Software Engineering

SWE 795, Spring 2017
Software Engineering Environments

GEORGE MASON UNIVERSITY

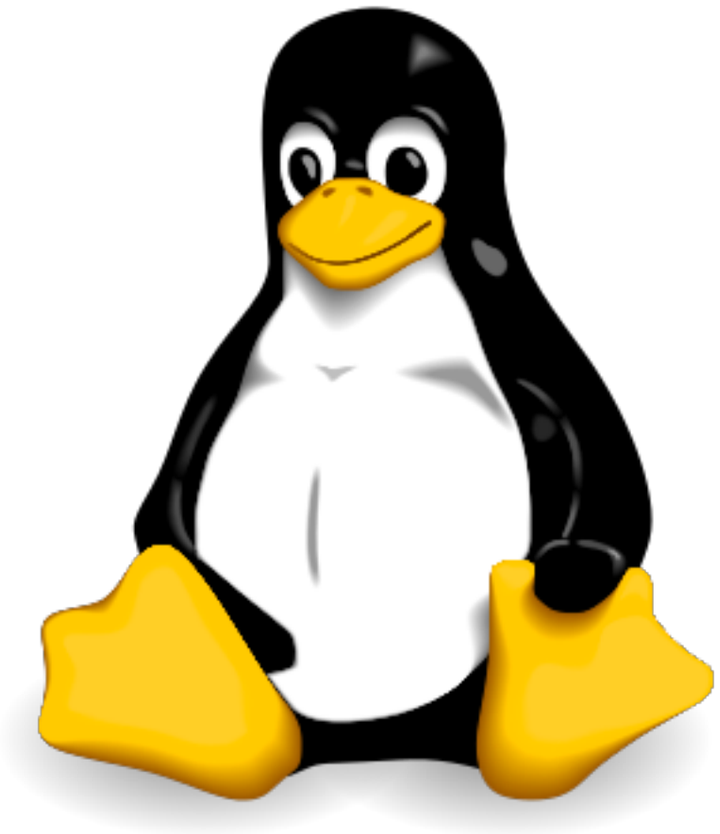
Today

- Part 1 (Lecture)(~80 mins)

- Break!

- Part 2 (Discussion)(~60 mins)
  - Discussion of readings

Open Source Software Development



Linux

Firefox

Dinnaco - Exchange Calculator - iPad App Module Architecture

CHALLENGE TYPE: Architecture

1 REGISTER FOR THIS CHALLENGE

2 SUBMIT YOUR ENTRIES

1st PLACE	2nd PLACE
$1,600	$800
Reliability Bonus $320	DR Points 720

CURRENT PHASE
Submission
2 Days 12 Hours 56 Mins
View all deadlines +

Details | Registrants (18) & Submissions (0) | Results

Challenge Overview

Challenge Overview

Welcome to the **Dinnaco - Exchange Calculator - iPad App Module Architecture** challenge! In this challenge, we are looking to define the module architecture for the new Private Exchange Sustainability Calculator iPad application.

Project Overview

The client for this project has an existing Excel-based application that helps organizations determine what solution is best for their healthcare needs. They have decided to convert this Excel-based application into a new iPad application!

For this challenge, **we need you to design the architecture for the new Private Exchange Sustainability Calculator iPad application**. A design concept challenge has already been completed that shows how the new application will work.

We look forward to seeing your architecture designs!

Tips for success!

- Asking questions early and getting PM's feedback is very important for the success of this competition.
- Raise questions if you feel anything is confusing, or if you have any questions on the provided documentation.

Project Description:

The client for this project has an existing Excel-based application that helps organizations determine what solution is best for their healthcare needs. This application helps organizations determine what solution is best for their healthcare needs. Through a series of questions a user of an organization is able to quickly determine if his/her organization is a good fit for a Private Healthcare Exchange. Dinnaco provides this tool as a service to its clients and is looking to improve the user experience and refresh the overall look and feel.

DOWNLOADS:

None

REVIEW STYLE:

Final Review: Community Review Board ?

Approval: User Sign-Off ?

CHALLENGE LINKS:

- Screening Scorecard
- Review Scorecard

GET THE UML TOOL:

- Github source code repository
- Mac disk image
- Java installer

SHARE:

0

CODEFIGHTS Challenges Tourneys 1 Leaders Badges Store 98,490 0 Tigran

Round 2 Score 20 Time 1:47

```
/*
 * @Input: an integer
 * @Output: abeolute value of @Input
 */
function myAbs(value) {
  if (value  0) {
    return -value;
  }
  return value;
}
```

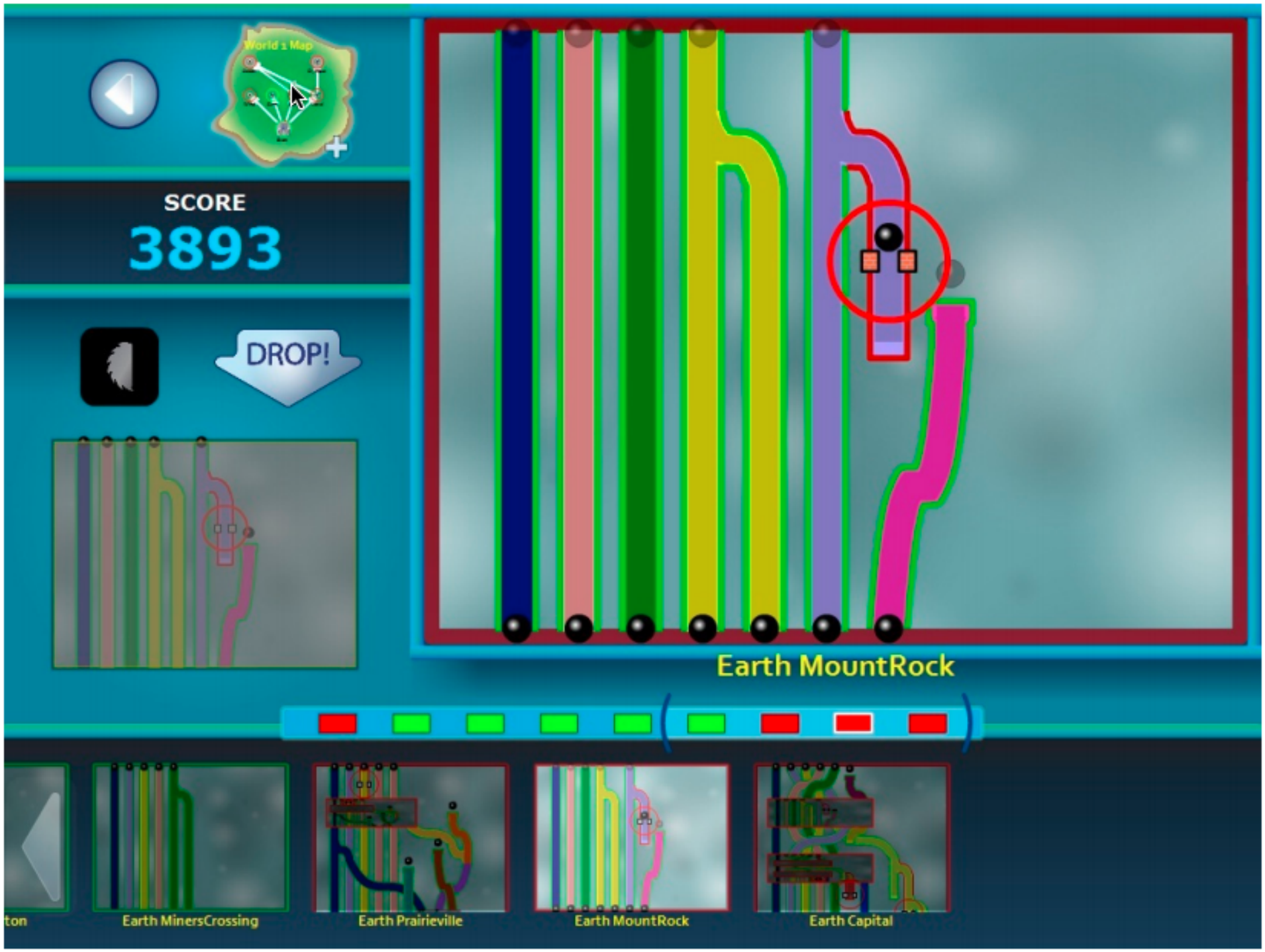
>> Each run will decrease round earnings

Run (3 left)

Submit (29 points) ctrl+enter

Exit / Rules

Games with a Purpose (Pipejam)



Crowdsourcing - definition

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.

[Howe 2006]

Crowdsourcing for software engineering - definition

The act of undertaking any **external software engineering tasks** by an undefined, potentially large group of online workers in an open call format.

[Mao+ 2015]

Some crowdsourcing models

Wisdom of the crowds (e.g., prediction markets)
aggregating information from diverse participants

Peer production (e.g., **Wikipedia)**
decentralized goal setting

Microtasking (**Mechanical Turk)**
short, self contained parallelizable tasks

Human computation / Games with a purpose
dual-sided value creation through play

Online labor markets
fluid labor forces, recruiting specialists

Dimensions of software crowdsourcing

Dimension	Explanation	Range
crowd size	**size** of the crowd necessary to effectively tackle the problem	small to large
task length	amount of **time** a worker spends completing an individual task	minutes to weeks
expertise demands	level of **domain** familiarity required for a worker to make a contribution	minimal to extensive
locus of control	**ownership** over the creation of new (sub)tasks	client to workers
incentives	**motivational** factors that cause workers to engage with the task	intrinsic to extrinsic
task interdependence	degree to which tasks within the overall workflow build on each other	low to high
task context	amount of **system information** a worker must know to contribute	none to extensive
replication	the number of times the same task may be **redundantly** completed	none to many

LaToza and van der Hoek. Crowdsourcing for Software Engineering: Models, Opportunities, Challenges. IEEE Software, Jan/Feb 2016.

Dimensions of PipeJam

Crowd size: medium

Task length: minutes

Expertise demands: minimal

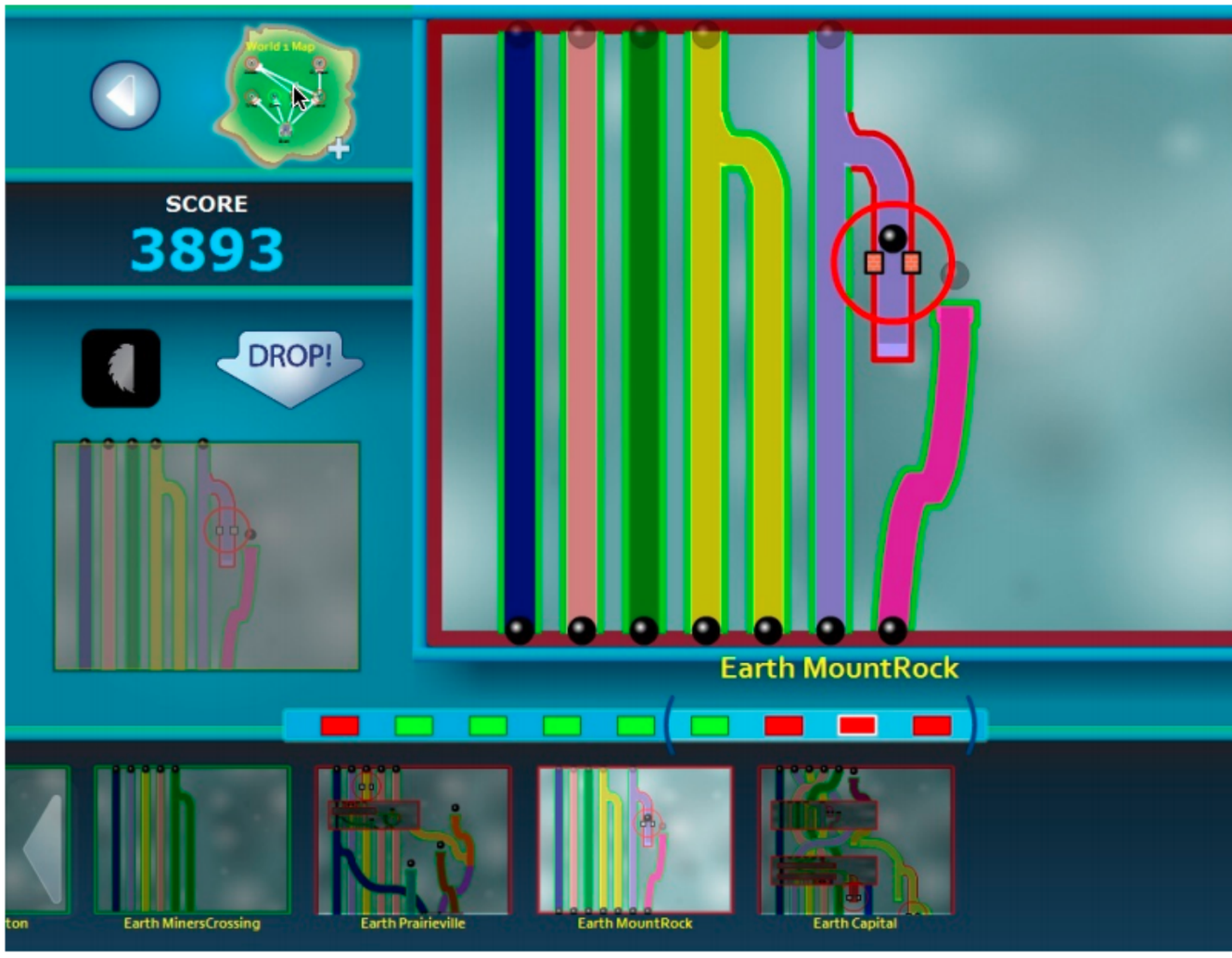
Locus of control: client

Incentives: intrinsic

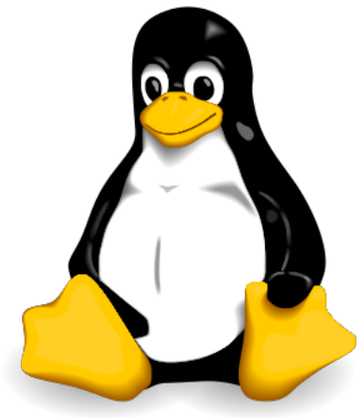
Task interdependence: medium

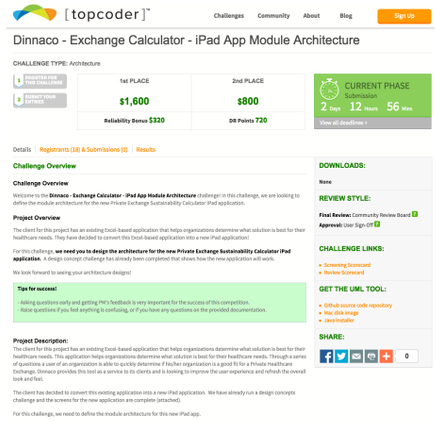
Task context: none

Replication: none



Locus of control & incentives





Crowd & Intrinsic

OSS

Q&A sites (StackOverflow)

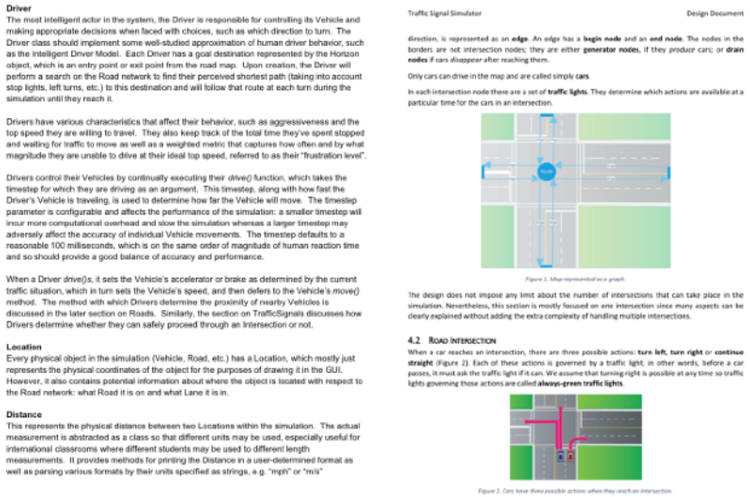
Gamified programming

Client & Extrinsic

Competitions

Labor markets

Replication



Driver
The most intelligent actor in the system, the Driver is responsible for controlling its Vehicle and making appropriate decisions when faced with choices, such as which direction to turn. The Driver class should implement some well-studied approximation of human driver behavior, such as the Intelligent Driver Model. Each Driver has a goal destination represented by the Horizon object, which is an entry point or exit point from the road map. Upon creation, the Driver will perform a search on the Road network to find their perceived shortest path (taking into account stop lights, left turns, etc.) to this destination and will follow that route at each turn during the simulation until they reach it.

Drivers have various characteristics that affect their behavior, such as aggressiveness and the top speed they are willing to travel. They also keep track of the total time they've spent stopped and waiting for traffic to move as well as a weighted metric that captures how often and by what magnitude they are unable to drive at their ideal top speed, referred to as their "frustration level".

Drivers control their Vehicles by continually executing their *drive()* function, which takes the timestep for which they are driving as an argument. This timestep, along with how fast the Driver's Vehicle is traveling, is used to determine how far the Vehicle will move. The timestep parameter is configurable and affects the performance of the simulation: a smaller timestep will incur more computational overhead and slow the simulation whereas a larger timestep may adversely affect the accuracy of individual Vehicle movements. The timestep defaults to a reasonable 100 milliseconds, which is on the same order of magnitude of human reaction time and so should provide a good balance of accuracy and performance.

When a Driver *drive()*s, it sets the Vehicle's accelerator or brake as determined by the current traffic situation, which in turn sets the Vehicle's speed, and then defers to the Vehicle's *move()* method. The method with which Drivers determine the proximity of nearby Vehicles is discussed in the later section on Roads. Similarly, the section on TrafficSignals discusses how Drivers determine whether they can safely proceed through an Intersection or not.

Location
Every physical object in the simulation (Vehicle, Road, etc.) has a Location, which mostly just represents the physical coordinates of the object for the purposes of drawing it in the GUI. However, it also contains potential information about where the object is located with respect to the Road network: what Road it is on and what Lane it is in.

Distance
This represents the physical distance between two Locations within the simulation. The actual measurement is abstracted as a class so that different units may be used, especially useful for international classrooms where different students may be used to different length measurements. It provides methods for printing the Distance in a user-determined format as well as parsing various formats by their units specified as strings, e.g. "mph" or "m/s"

Traffic Signal Simulator Design Document

direction, is represented as an **edge**. An edge has a **begin node** and an **end node**. The nodes in the borders are not intersection nodes; they are either **generator nodes**, if they *produce cars*; or **drain nodes** if cars *disappear* after reaching them.

Only cars can drive in the map and are called simply **cars**.

In each intersection node there are a set of **traffic lights**. They determine which actions are available at a particular time for the cars in an intersection.

Figure 1. Map represented as a graph.

The design does not impose any limit about the number of intersections that can take place in the simulation. Nevertheless, this section is mostly focused on one intersection since many aspects can be clearly explained without adding the extra complexity of handling multiple intersections.

4.2 Road Intersection

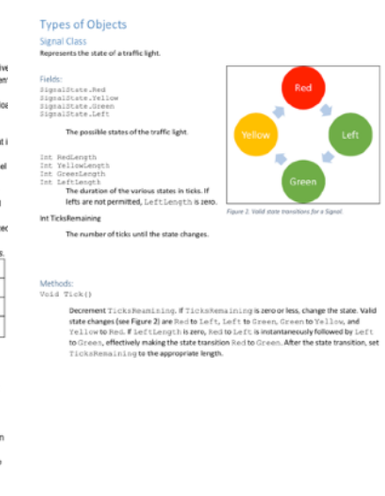
When a car reaches an intersection, there are three possible actions: **turn left, turn right** or **continue straight** (Figure 2). Each of these actions is governed by a traffic light, in other words, before a car passes, it must ask the traffic light if it can. We assume that turning right is possible at any time so traffic lights governing those actions are called **always-green traffic lights**.

Figure 2. Cars have three possible actions when they reach an intersection.

Prototype
The following image shows a high level mock up of how TrafficSim looks like. This is a first cut design trying to capture and show the various elements of the use-case in a holistic view. This will be referred to in the detailed design later.

Figure 1 UI Mockup of TrafficSim

The post-it notes above explain provide more detail about what the mockup is showing. You may have to zoom in further to read it.



2. Design Overview

In this section the overview of suggested design will be described.

2.1. Key architectural drivers and tradeoffs

Based on given problem description, was concluded that the key architectural drive would be usability, because the main purpose of this system is to show and teach studen how the signal timing schemes in different scenarios affect traffic flow. Next key architectural driver would be performance of the system. The system will produce high loa on user laptop/device, because the system will simulate behavior of multiple graphical instances and the system should run on average user laptop/device.

It is not research project in traffic flow theory, the program should do exactly what i supposed to do - demonstrate student how the different timing schemes in different scenarios affect the traffic flow. The priority of such quality attribute as accuracy of model simulation is low.

From the business perspective, there are would be limitation by time and human resources, because the project probably will not have a high budget, because it is small non-profit educational project.

Based on the system description and previous conclusions, the following prioritize list of key architectural drives was created:

Table 1. Architectural drivers priorities.

	Architectural driver	Priority
1.	Usability	High
2.	Performance	Medium
3.	Accuracy of model simulation	Low

2.2. Technologies Used

In order to satisfy given architectural drivers, the technology should support developing application with good level of usability, it should graphical engine in order to draw world simulation and should have good performance characteristics. In order to satisfy business constrain the technology should be proven, and have examples of implementation of such application. Considering high priority of usability attribute and audience of the application, can be concluded that the application should run application on different devices: beginning from the PC, laptops, tablets, and ending with smartphones. Also support of modern devices will increase the student's interest for the application and topic. There are two approaches:

Types of Objects

Signal Class

Represents the state of a traffic light.

Fields:

```
SignalState.Red
SignalState.Yellow
SignalState.Green
SignalState.Left
```

The possible states of the traffic light.

```
Int RedLength
Int YellowLength
Int GreenLength
Int LeftLength
```

The duration of the various states in ticks. If lefts are not permitted, `LeftLength` is zero.

Int TicksRemaining

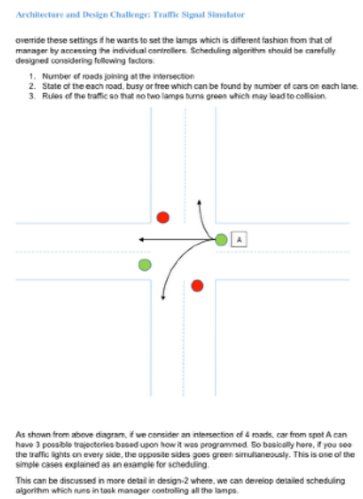
The number of ticks until the state changes.

Figure 2. Valid state transitions for a Signal.

Methods:

```
Void Tick()
```

Decrement `TicksRemaining`. If `TicksRemaining` is zero or less, change the state. Valid state changes (see Figure 2) are `Red` to `Left`, `Left` to `Green`, `Green` to `Yellow`, and `Yellow` to `Red`. If `LeftLength` is zero, `Red` to `Left` is instantaneously followed by `Left` to `Green`, effectively making the state transition `Red` to `Green`. After the state transition, set `TicksRemaining` to the appropriate length.



Architecture and Design Challenge: Traffic Signal Simulator

override these settings if he wants to set the lamps which is different fashion from that of manager by accessing the individual controllers. Scheduling algorithm should be carefully designed considering following factors:

1. Number of roads joining at the intersection
2. State of the each road, busy or free which can be found by number of cars on each lane.
3. Rules of the traffic so that no two lamps turns green which may lead to collision.

As shown from above diagram, if we consider an intersection of 4 roads, car from spot A can have 3 possible trajectories based upon how it was programmed. So basically here, if you see the traffic lights on every side, the opposite sides goes green simultaneously. This is one of the simple cases explained as an example for scheduling.

This can be discussed in more detail in design-2 where, we can develop detailed scheduling algorithm which runs in task manager controlling all the lamps.

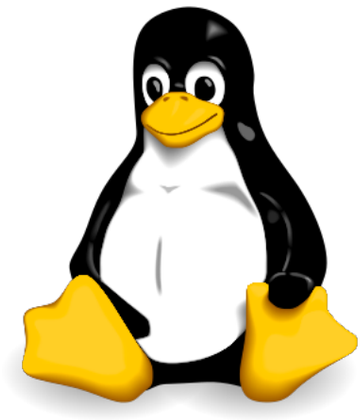
Whole project (OSS)

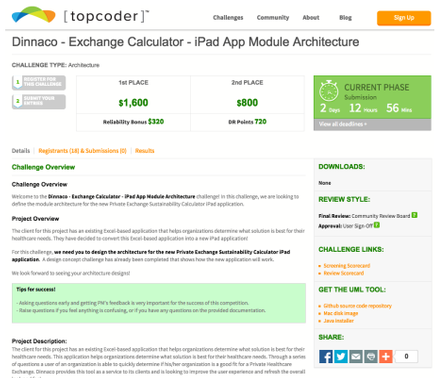
Derivative projects (OSS)

Work items (Competitions)

Design decisions (Q&A sites)

Task length & Task context





stackoverflow

Long (hours - weeks) & High

OSS

Competitions

Labor markets

(Software development work)

Short (minutes) & Low

Gamified programming

Q&A sites

Labor markets for testing

(Tasks w/ clear goals)

Examples of software crowdsourcing platforms

Dimension	Open source	TopCoder	UserTesting.com
crowd size	small – medium	small	medium
task length	hours – days	days - week	minutes
expertise demands	moderate	extensive	minimal
locus of control	workers	client	client
incentives	intrinsic	extrinsic	extrinsic
task interdependence	moderate	low	low
task context	extensive	minimal	none
replication	none	several	many

Programming is becoming more social and fluid

stackoverflow

share hard-earned expertise

>7M monthly visits

codecademy

learn programming through structured examples

>24M users

github SOCIAL CODING

reputation made visible

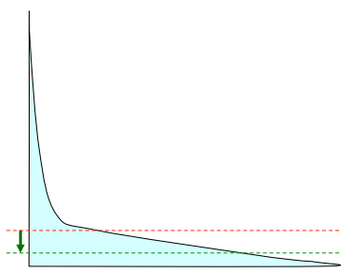
>8M users

competitive programming duels

>100K users

Crowdsourcing SE opportunities

(similar microtask, done for many tasks)



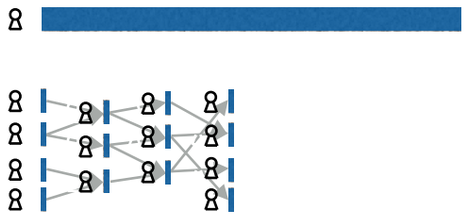
Extreme **specialization** - more effectively match workers to work

Worker might choose microtasks they most enjoy or most experienced.

Increase **participation** by reducing contribution barriers

Expert developer might do a key microtask and contribute key insight.

Microtask design might enable non-programmer to do work.



Let developers **learn** while working

Novice selects microtasks related to learning objective.

Reduce **time** to market

Decomposing tasks into parallel microtasks reduces time.

More **fluid** software teams

Enable teams to hire experts for short engagements

Platform for data-driven SE research

Produces archival data on structure of work inside tasks

Crowdsourcing SE topics today

- Peer production on Q&A sites (Stack Overflow)
- Sharing expertise
- Increasing parallelism in programming
- Replication & competition in software design

Commons-Based Peer Production

- Community with distributed control where contributors, rather than a paying client, make decisions about scope and goals of project.

- Contributors may be motivated by opportunity for experience, reputation, altruism.

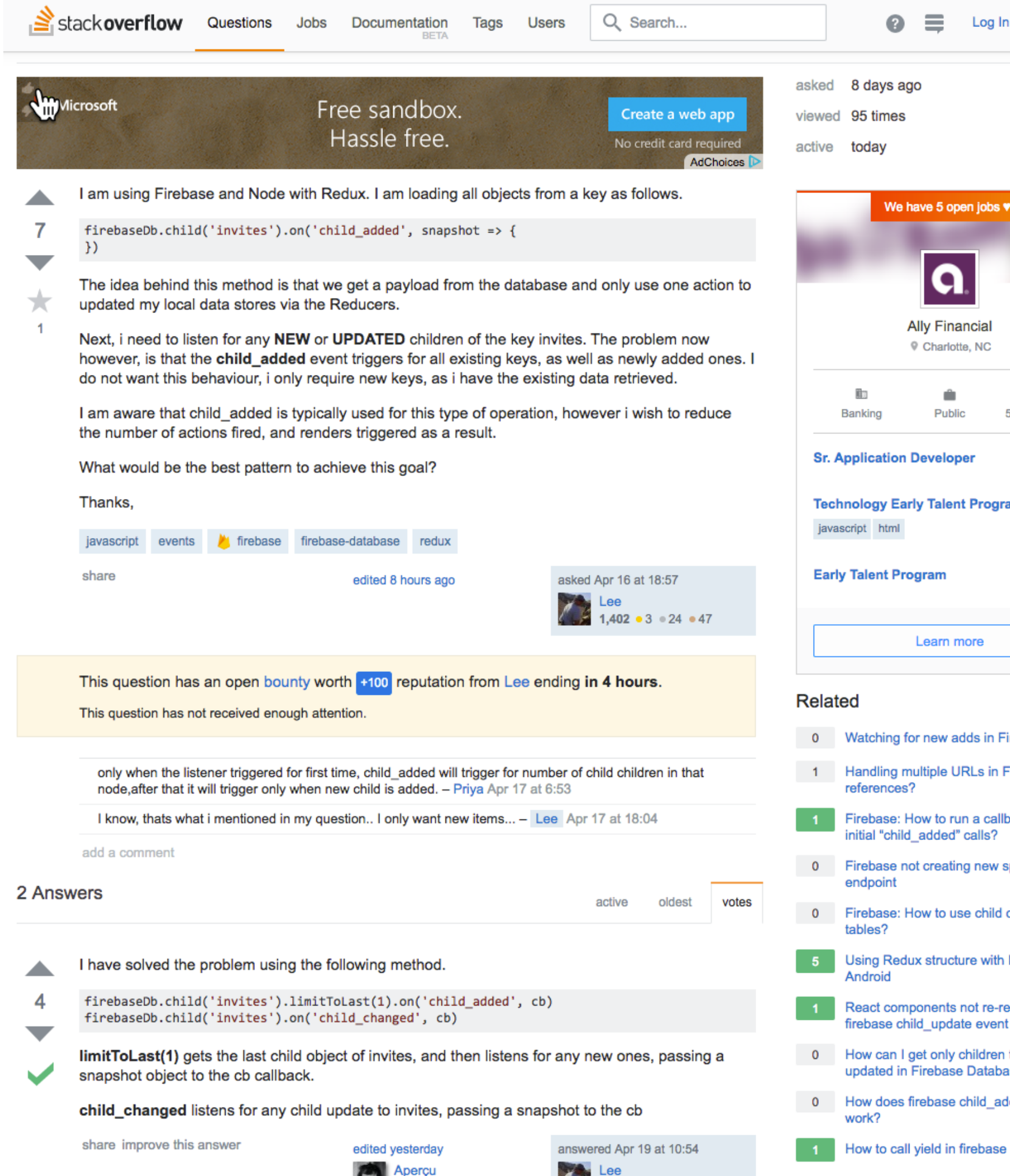
Mechanisms for soliciting contributions

- Key requirements
  - Output is decomposable into separate contribution units
  - Contributions are sufficiently fine-grained to capture contributions from those whose motivation will only sustain short efforts
  - Low cost mechanism defends against incompetent and malicious contributions

Y. Benkler and H. Nissenbaum, "Commons-based Peer Production and Virtue*," *J. Polit. Philos.*, vol. 14, no. 4, pp. 394– 419, 2006.

Stack Overflow

- >100M monthly unique visitors
- >3.9B yearly visits
- >3.7M questions asked



Questions are answered quickly

- Median first answer 11 mins, accepted answer 21 mins

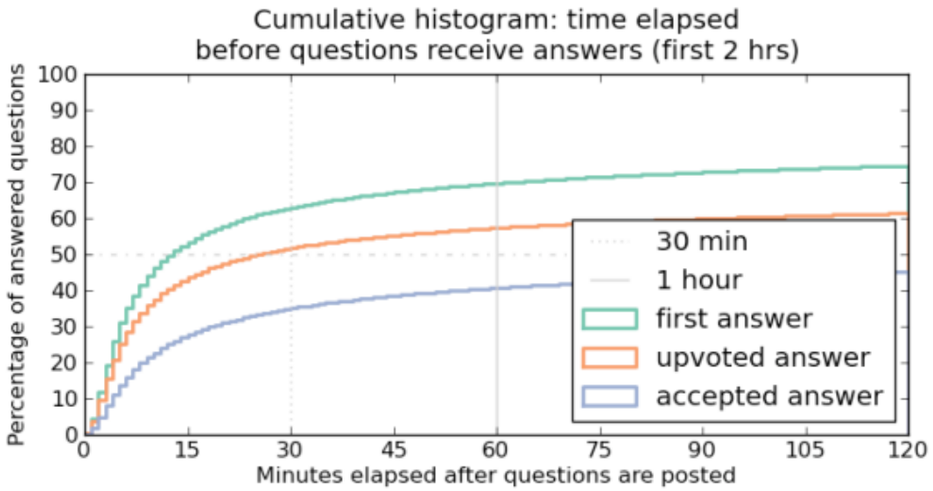


Figure 4: Answers in the 2 hours after questions are posted.

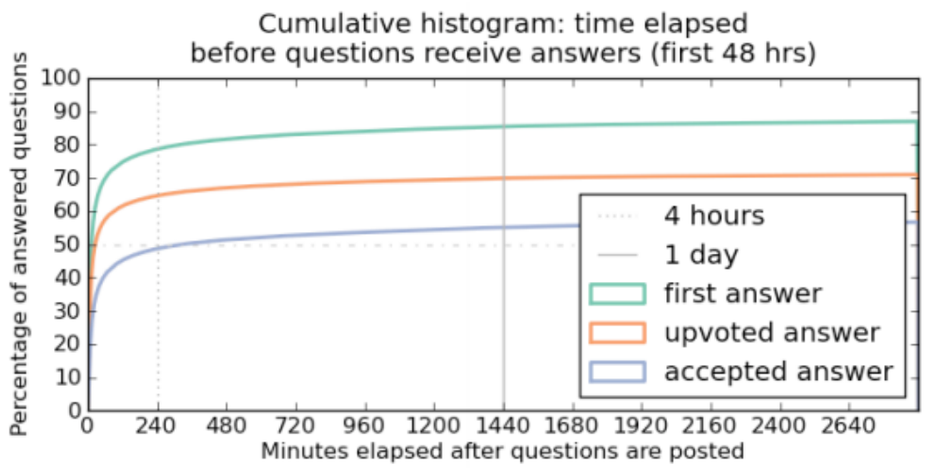


Figure 5: Answers in the first 2 days: little activity after 4 hrs.

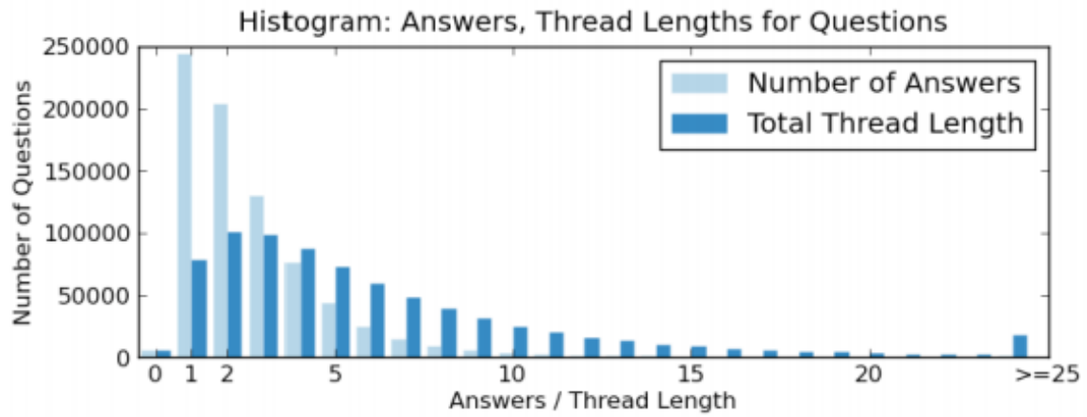


Figure 3: Answers and thread lengths for all questions.

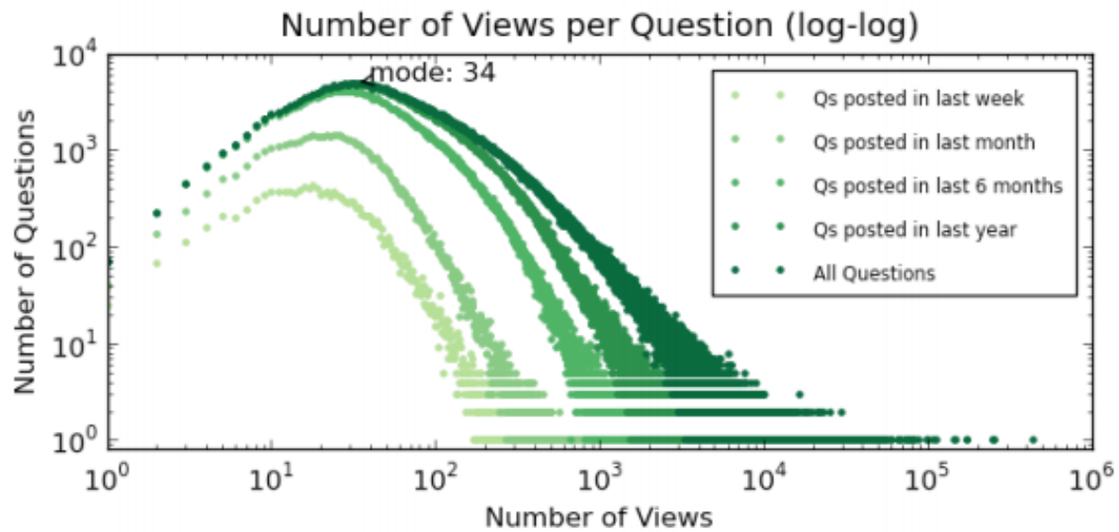


Figure 10: Questions receive dozens of views from visitors.

Lena Mamykina, Bella Manoim, Manas Mittal, George Hripcsak, and Björn Hartmann. 2011. Design lessons from the fastest q&a site in the west. *Conference on Human Factors in Computing Systems,* 2857-2866.

What makes StackOverflow work?

- Make competition productive
  - Tight focus on technical answers & voting system offered strong alternative to conversational forums
  - Game mechanisms through a reputation system led to intense participation
  - Credibility in the community
    - Founders were thought leads that enabled them to gain a critical mass
- Evolutionary approach to design
  - Continuous feedback loop with users, helped prioritize feedback

Lena Mamykina, Bella Manoim, Manas Mittal, George Hripcsak, and Björn Hartmann. 2011. Design lessons from the fastest q&a site in the west. *Conference on Human Factors in Computing Systems,* 2857-2866.

Coverage of APIs

- Googled for all methods in jQuery API, examined top 10 links for type of result

Search Result Type	Coverage	Mean Rank
code snippet site	8.7%	9
q&a	9.8%	9
forum	20.2%	8
official bug tracker	21.4%	3
mailing list entry	25.4%	7
official documentation	30.1%	3
official forum	37.0%	3
unofficial documentation	63.6%	6
stackoverflow	84.4%	6
blog post	**87.9%**	**5**
official API	99.4%	1

Chris Parnin and Christoph Treude. 2011. Measuring API documentation on the web. *International Workshop on Web 2.0 for Software Engineering*, 25-30.

Sharing expertise

- Many developers do similar tasks everyday

- Write similar code —> code reuse
- Fix similar defects —> ???
- Do similar performance optimizations —> ???
- ....

- Can solve with StackOverflow, but requires developers to share knowledge & developer to find it
- Goals:
  - Increase breadth & quantity of expertise shared
  - Decrease time to access expertise
  - Increase probability that relevant expertise is found

On Demand Expert Assistance

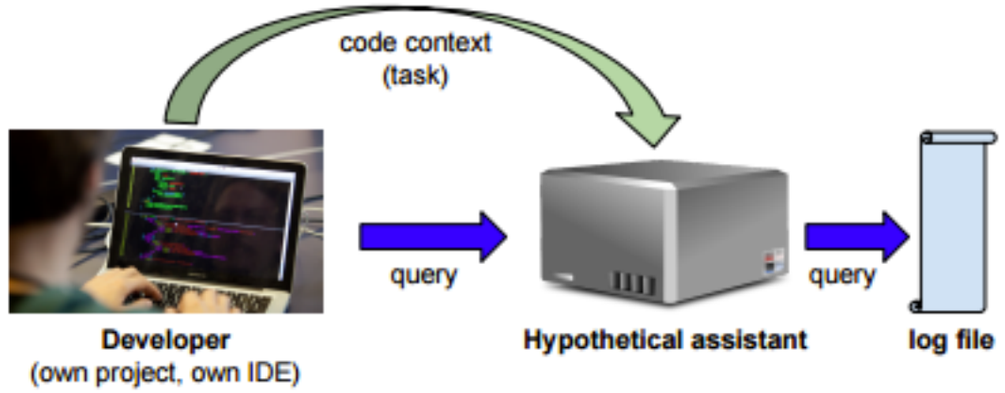


Figure 1. Setup for our hypothetical assistant study. Developer participants were asked to bring their own task to complete, and ask questions from our hypothetical assistant as if it could answer any support question needed. Participants were still allowed to use traditional online resources and augment them with the assistant as they saw fit. Context related to their programming task was collected at the beginning of each study, and audio of their questions was recorded during the session.

Description	# Sessions (out of 5)	# / Session
Memory Aids: Participants sought a specific function name	2	0.6
Explanatory Requests: Participants sought examples or explanations of their code	4	1.8
High-Level Strategic Guidance: Participants sought best ways to approach problems	5	4.6
Code Requests: Participants sought specific pieces of code	2	0.6
Bug Fixing: Participants sought specific solutions to program errors	2	1.0
Code Refactoring: Participants asked for code improvements	2	0.8
Effort-Saving Requests: Participants handed off tasks to save time and effort	4	4.0

Table 1. Common query types observed during our hypothetical assistant study, with corresponding frequencies. Each of these query types suggests a support role that remote software development assistants can play in future systems. "Number of Sessions" indicates the number of different sessions that each type of question occurred in, while "Number of Queries per Session" indicates the number of queries that referred to solving one of these query types, on average.

Y. Chen, S. Oney, **W.S. Lasecki**. (2016) Towards Providing On-Demand Expert Support for Software Developers. In *Proceedings of the ACM Conference on Human Factors in Computing Systems.*

On Demand Expert Assistance

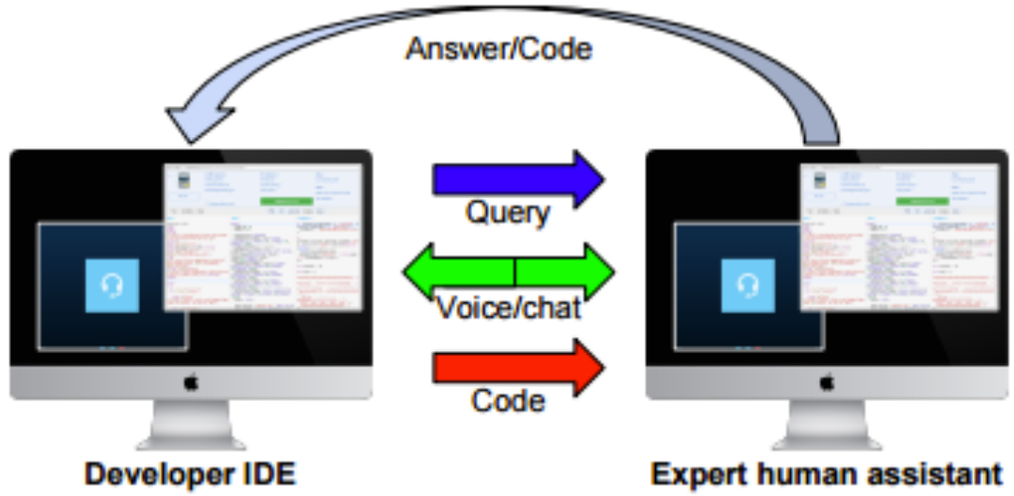


Figure 2. Setup for our human expert assistant study. In each trial, one "requester" participant (developer) was paired with one "helper" (expert, based on a pre-study skill assessment). Participants could chat via either text or voice, and requesters were able to share their screen with helpers as needed using Skype. The helper was tasked with assisting the requester *reactively*, meaning that they only responded to queries, and did not proactively propose solutions or approaches. This simulates a "best case" (repeated, non-multiplexed helper) on-demand model where human experts are not expected to be continuously available between end-user queries.

Patterns	Description	# Sessions (out of 12)	# / Session	# Interviews (out of 12)	# / Interview
Background	Helpers wanted to know the requester's experience level and background	11	1.5	7	0.7
Context	Helpers wanted to know what the high-level goals and context were	11	1.5	9	0.8
Sharing	Participants wanted a shared editor that lets helpers type code directly	11	2.1	11	1.1
Real-Time Response	Participants needed immediate responses (some preferred voice, others text)	12	N/A	10	0.9
Integrated System	Participants did not like switching windows, and would prefer a single system	9	N/A	7	0.8
Personalized Help	Requesters wanted help suited to their intent, e.g., specifying "teach me" when more explanation was desired	9	1.0	10	1.2

Y. Chen, S. Oney, **W.S. Lasecki**. (2016) Towards Providing On-Demand Expert Support for Software Developers. In *Proceedings of the ACM Conference on Human Factors in Computing Systems.*

HelpMeOut

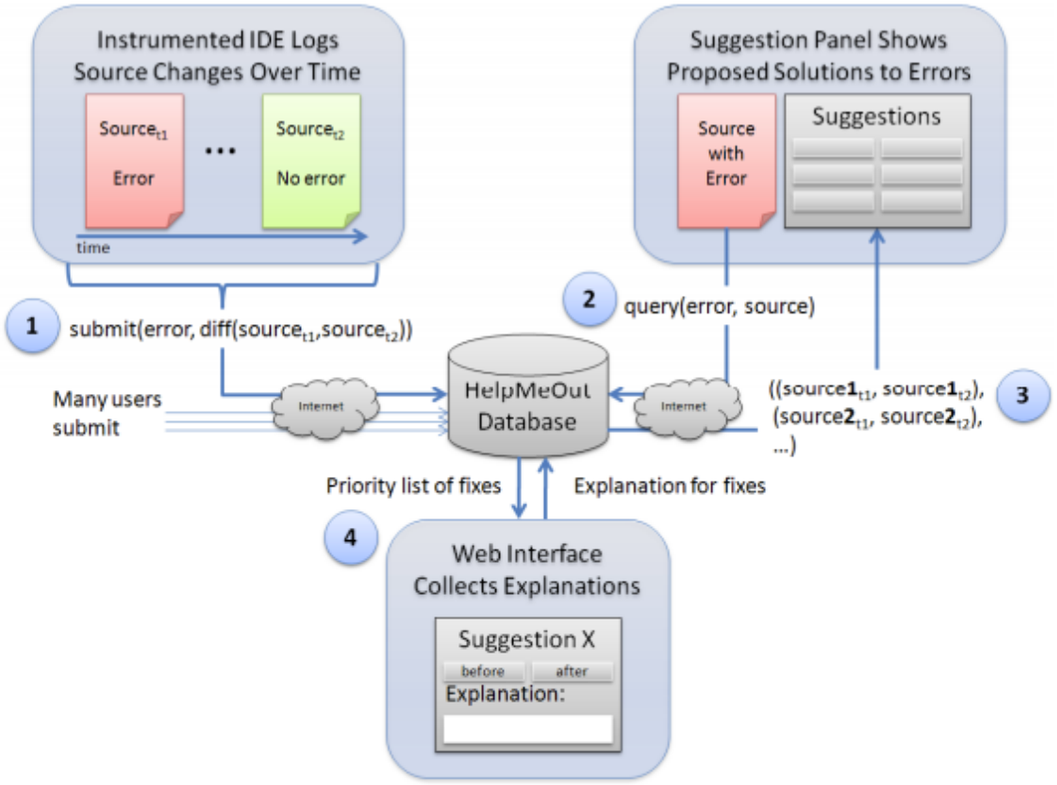
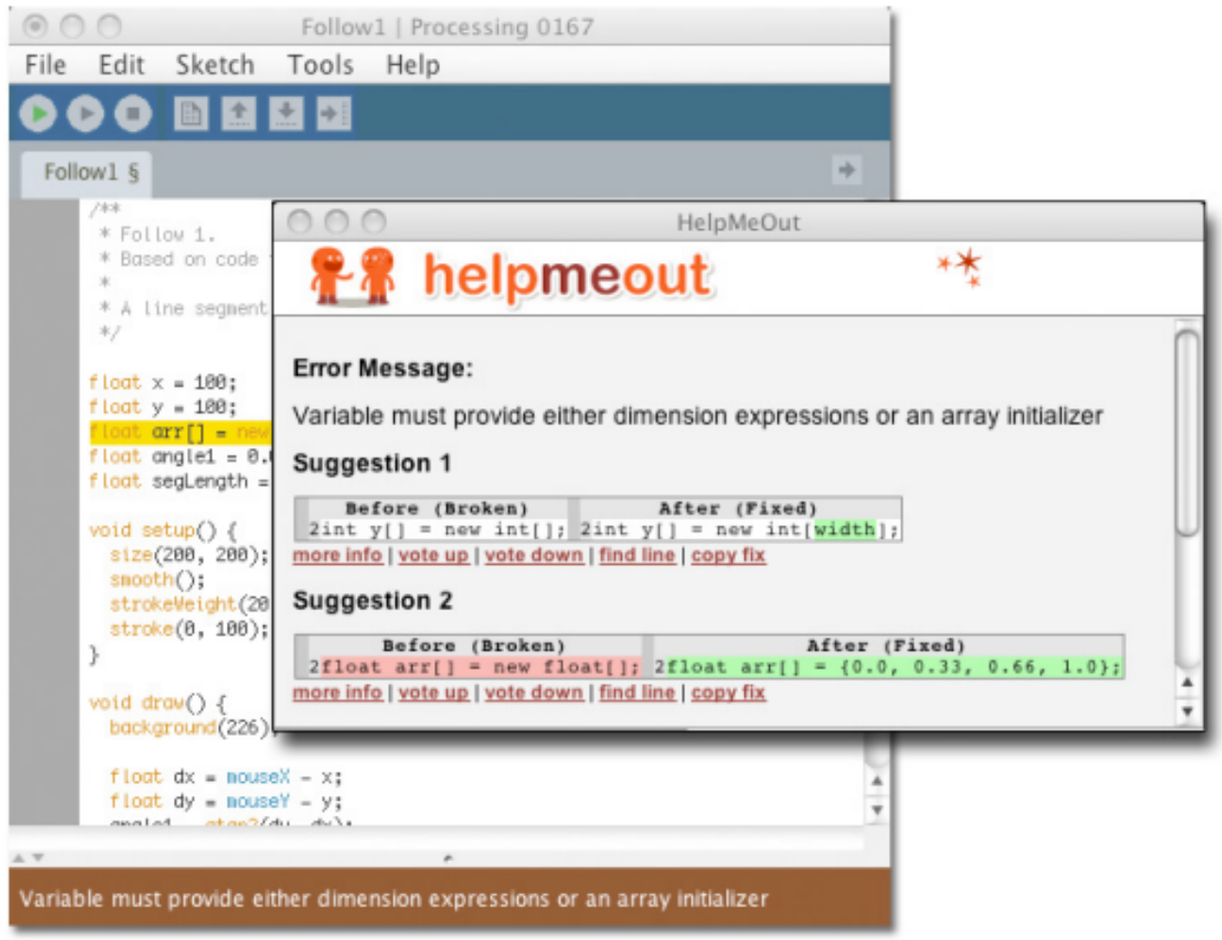
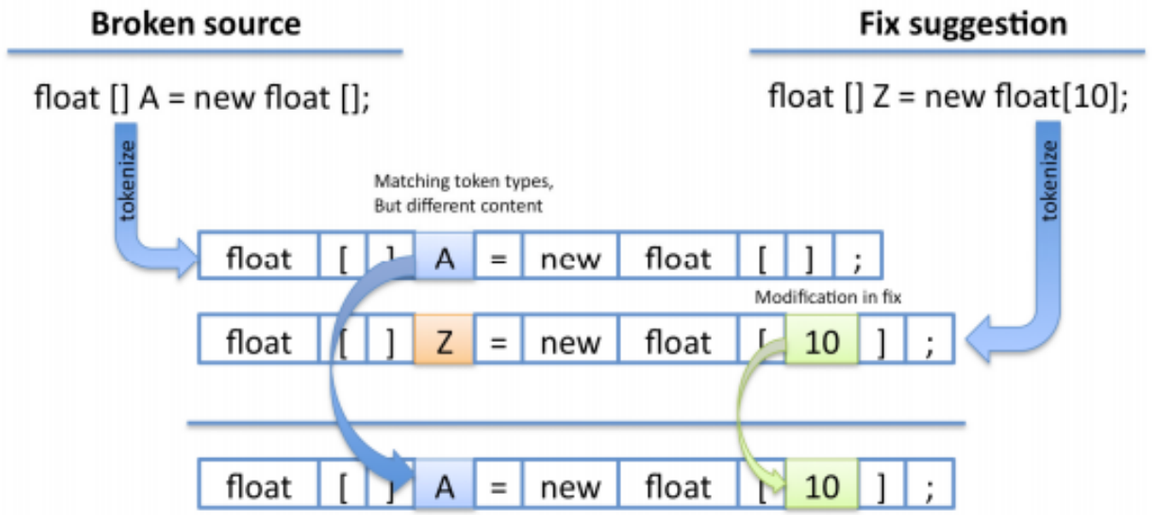


Figure 1. HelpMeOut offers asynchronous collaboration to suggest corrections to programming errors. **1**: IDE instrumentation collects bug fixes and sends them to a remote database. **2**: Other programmers query the database when they encounter errors. **3**: Suggested fixes are shown inside their IDE. **4**: Explanations for fixes are collected in a web interface.





Björn Hartmann, Daniel MacDougall, Joel Brandt, and Scott R. Klemmer. 2010. What would other programmers do: suggesting solutions to error messages. *Conference on Human Factors in Computing Systems*, 1019-1028.

Tutorons

```
def get_video_data(video_page_url):
    video_data = {}
    response = requests.get(root_url + video_page_url)
    soup = bs4.BeautifulSoup(response.text)
    video_data['title'] = soup.select('div#videobox h3')[0].get_text()
    video_data['speakers'] = [a.get_text() for a in soup.select('div#sidebar a[href^=/speaker]')]
    video_data['youtube_url'] = soup.s
```

A few things to note about this functio

- The URLs returned from the scraping o
- The session title is obtained from the <
  because the `select()` call returns a li
- The speaker names and YouTube links

Now all that remains is to scrape the v
very simple to write as a continuation
we can also scrape the likes and dislik

```
def get_video_data(video_page_url):
```

You found a CSS selector.

The selector 'div#sidebar a[href^=/speaker]' chooses links with URLs starting with '/speaker' from a container with the ID 'sidebar'.

If you haven't seen them before, selectors pick sections of HTML pages by their names or properties. Once you've 'grabbed' elements with a selector, you can manipulate them, like changing their appearance or text.

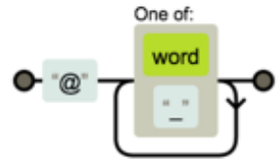
Here's an example of what this selector will finc

```
<div id="sidebar">
    <a href="/speaker">
    </a>
</div>
```

```
regex_str = [
    emoticons_str,
    r'<[^>]+>', # HTML tags
    r'(?:@[\w_]+)', # @-mentions
    r"(?:\#+[\w_]+[\w\'_\-]*[\w_]+)", # hash-tags
```

You found a regular expression!

We compare text to regular expressions to see if they match a pattern or not. To read this regular expression diagram, follow it from left to right as you see the letters and symbols it looks for in a line of text.



NODE	EXPLANATION
(?:	group, but do not capture:
@	'@'
[\w_]+	any character of: word characters (a-z, A-Z, 0-9, _), '_' (1 or more times (matching the most amount possible))
)	end of grouping

```
?)', # numbers
)", # words with - and '
rds


|'.join(regex_str)+')', r
+emoticons_str+'$', re.VE


s)

alse):


oticon_re.search(token) e


just an example! :D http:

:', 'just', 'an', 'exampl
```

A. Head, C. Appachu, M. A. Hearst and B. Hartmann, "Tutorons: Generating context-relevant, on-demand explanations and demonstrations of online code," *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Atlanta, GA, 2015, pp. 3-12.

Codex

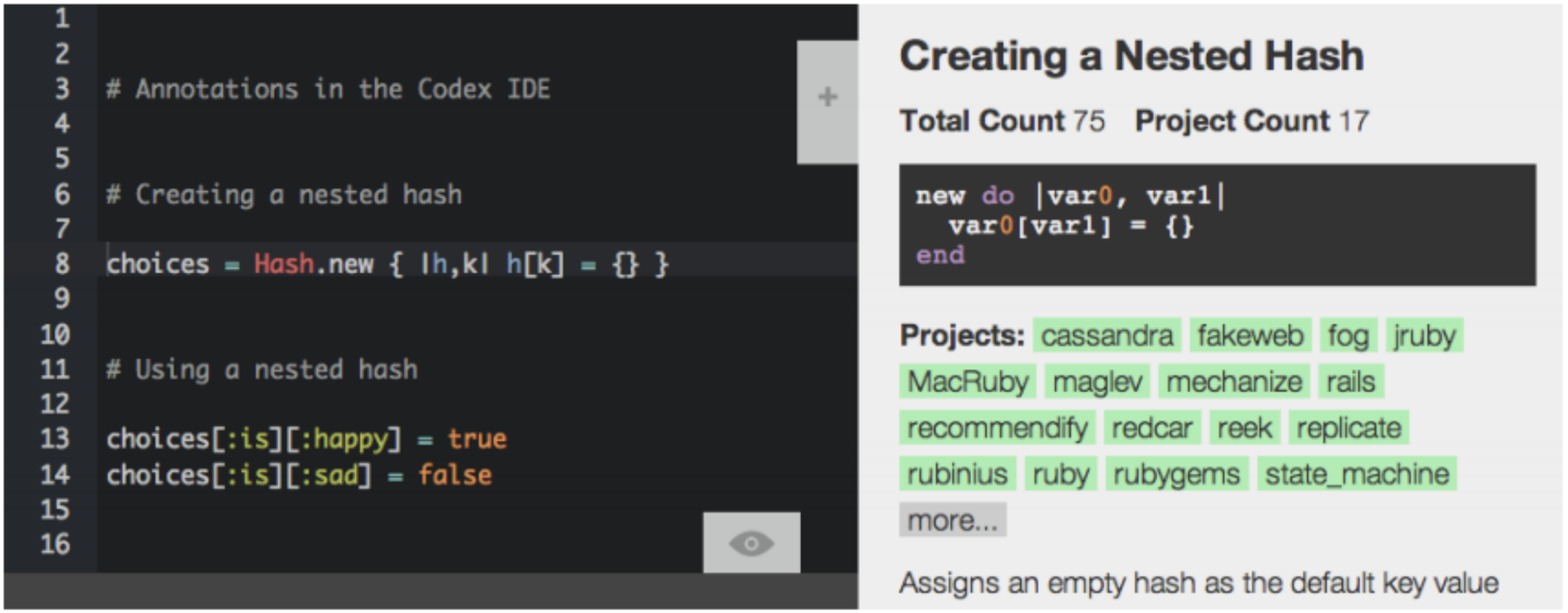


Figure 1. Codex draws on millions of lines of open source code to create software engineering interfaces that integrate emergent programming practice. Here, Codex's pattern annotation calls out popular idioms that appear in the user's code.

https://www.youtube.com/watch?v=jAIWXbygKuc

Ethan Fast, Daniel Steffee, Lucy Wang, Joel R. Brandt, and Michael S. Bernstein. 2014. Emergent, crowd-scale programming practice in the IDE. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2491-2500.

CodeOn

- Requestor speaks in IDE, requests help
  - IDE captures code context
- Responders author descriptions, annotations, snippets, explanations
- Requestor can integrate responses

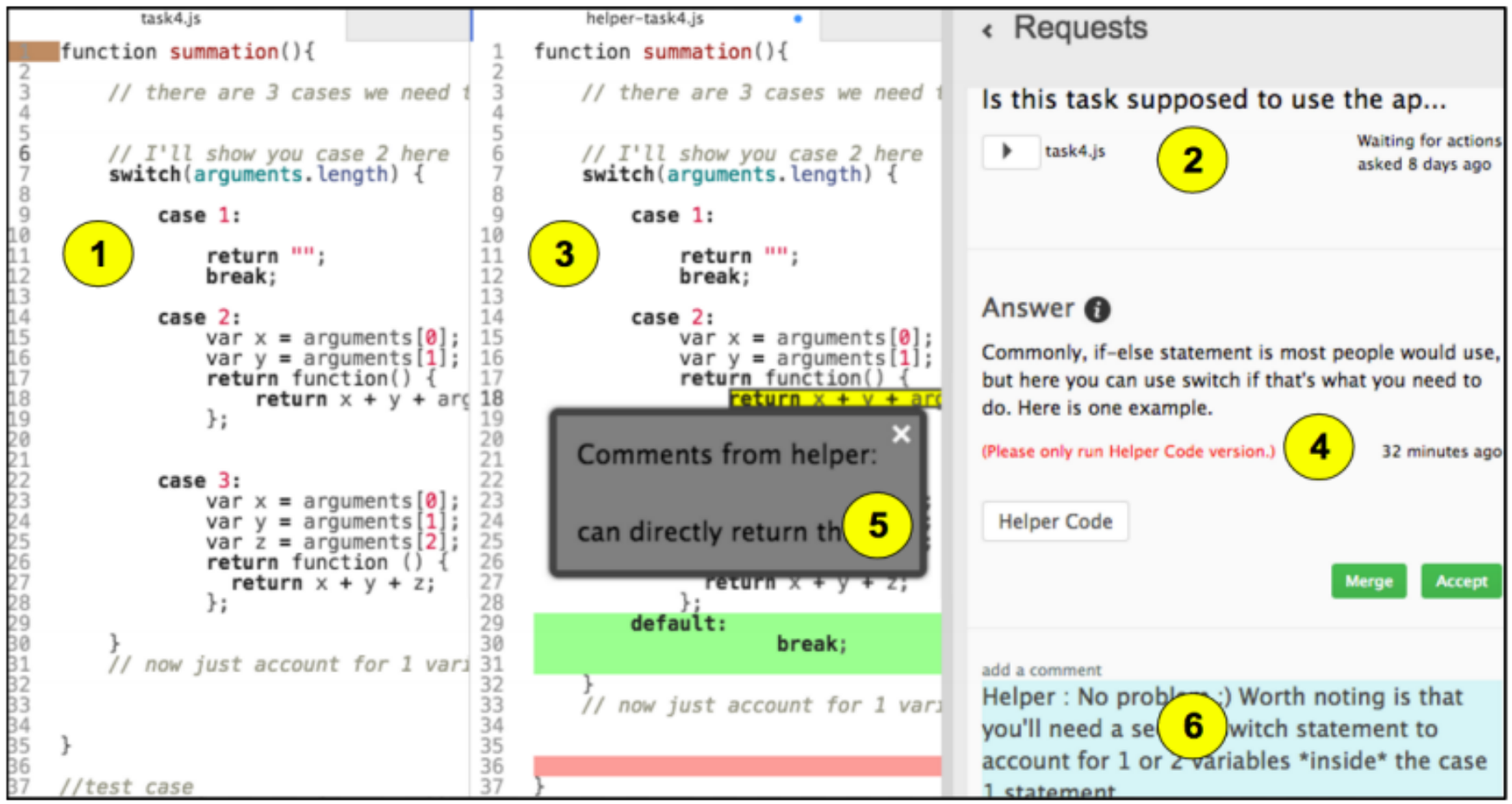


Figure 2: Codeon interface where the requests and responses are on the right panel(2). Developer's code(1) and helper's code(3) are side by side for better comparison. Other responses includes explanation(4), annotation(5), and comments(6)

Y. Chen, S.W. Lee, Y. Xie, Y. Yang, **W.S. Lasecki**, S. Oney. Codeon: On-Demand Software Development Assistance. *Conference on Human Factors in Computing Systems (CHI 2017)*.

Increasing parallelism in programming

Build a large application in a day?

>24 million users x 1 day = ???

Reduce joining barriers

- OSS imposes *joining barriers* that dissuade casual contributions
  - Identifying appropriate contacts & receiving timely feedback
  - Identifying appropriate tasks and corresponding artifacts
  - Understanding project structure, complex code, setting up a workplace
  - Unclear documentation & info overload
  - Learning project practices, domain knowledge, technical expertise

I. Steinmacher, M. A. G. Silva, M. A. Gerosa, and D. F. Redmiles, "A systematic literature review on the barriers faced by newcomers to open source software projects," *Inf. Softw. Technol.*, vol. 59, pp. 67–85, 2015.

Costs & challenges

- Task context & interdependence

- Handoffs — Specification often implicit through interpretation of requirements & domain, rather than explicitly represented in artifacts.

Key questions

To what extent can software development work be decomposed at a granularity smaller than commits?

What task context would such tasks require?

How could such work be coordinated and organized?

Collabode

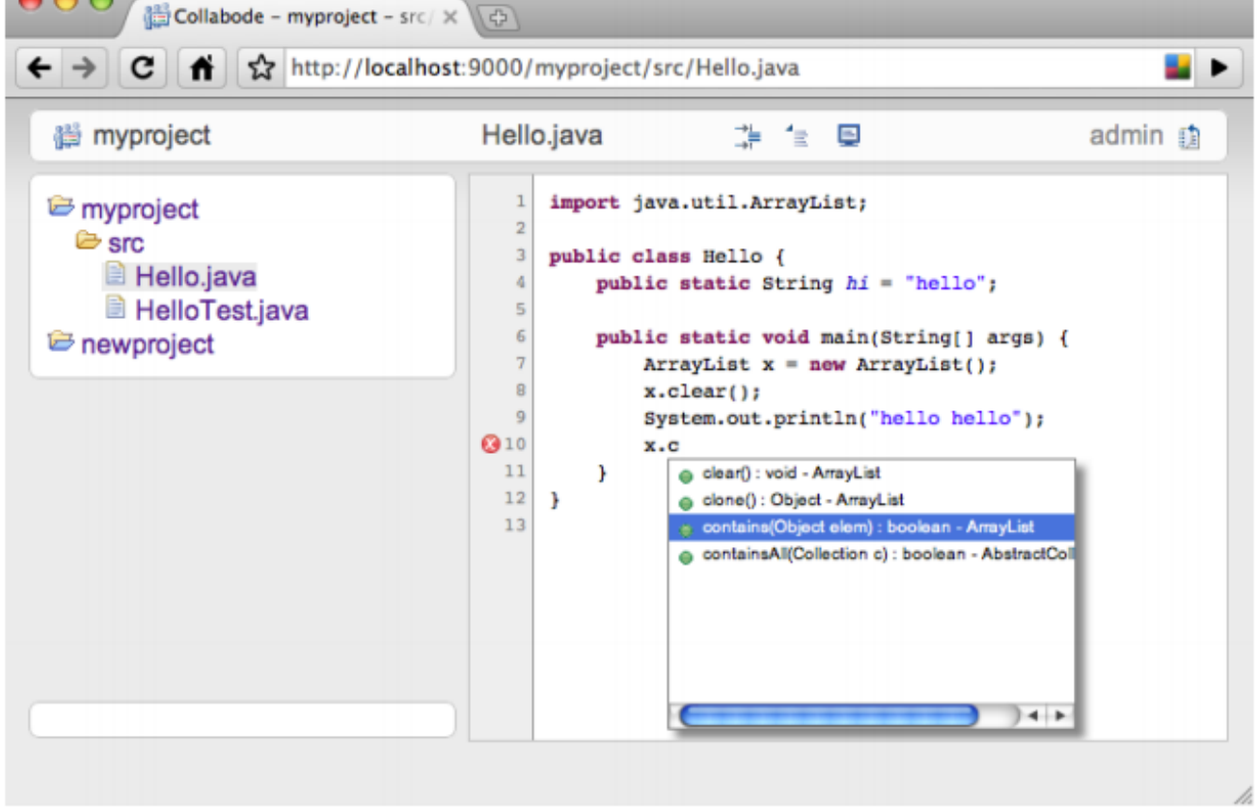


Figure 1: The Collabode web-based IDE allows multiple simultaneous editors to work together.

- Micro-outsourcing
  - Enables “original programmer” to describe custom microtasks in prose
  - Each microtask completed by workers
  - Workers may discuss microtasks with original programmer for clarification
  - Original programmer responsible for feedback

Max Goldman, Greg Little, and Robert C. Miller. Real-Time Collaborative Coding in a Web IDE. UIST '11.
Max Goldman, Greg Little, and Robert C. Miller. Collabode: Collaborative Coding in the Browser. CHASE '11.

Collabode Evaluation

- Was possible to use the workflow for programming
- But…
  - Managing the crowd imposed large overhead on requestor to answer questions & eval contributions
  - Code often had subtle bugs, which were time consuming to find and identify
  - Anonymous workers led to low responsibility for work

Microtask programming model

- Programming tasks
  - are short (< 10 minutes)
  - are completed by pool of transient workers
  - are automatically generated by system
  - iteratively update artifacts

EDIT A FUNCTION 10 pts

Can you write some code in the function below? but don't be a hog, if there's much to do be sure to use pseudocalls or pseudocode to leave some work for others.

■ FUNCTION CONVENTIONS

Want to sketch something? Write a line of **pseudocode** with '//#'

Want to call a function? Write a **pseudocall** beginning it with '//! brief description of what the function should do'

Example:

```
function foo() {
    var values = [ 128, 309 ];
    //# calc the least common multiple of values
    var avg;
    //!avg = Compute the average of (values)
    return { average: avg, lcm: lcm };
}
```

Note that all function calls are pass by value (i.e., if you pass an object to a function, and the function changes the object you will not see the change).

■ AVAILABLE DATA TYPES

- String
- Number
- Boolean
- Element
- Position
- LineSegment
- RenderInstructions
- Action

■ JAVASCRIPT TUTORIAL

```
/**
  Renders the specified array of Elements, creating and returning a
  RenderInstructions describing how they should be rendered to the
  screen. Lines and freehand shapes should be rendered in order of each
  of their lines. Rectangles should be rendered with a series of
  segments beginning in the top left and proceeding  clockwise.

  @param Element element , the element to be rendered

  @return RenderInstructions
**/
function renderElement(element){
    //#Mark this function as implemented by removing this line.
    return {};
}
```

Skip Submit

```
ple of values

f (values)
n };
```

(i.e., if you pass an object to a function, and the function changes the object you will not see the change).

```
/**
  Renders the specified array of Elements, creating and returning a
  RenderInstructions describing how they should be rendered to the
  screen. Lines and freehand shapes should be rendered in order of each
  of their lines. Rectangles should be rendered with a series of
  segments beginning in the top left and proceeding  clockwise.

  @param Element element , the element to be rendered

  @return RenderInstructions
**/
function renderElement(element){
    if (element.type == "Line")
    {
        //# render line
    }
    else if (element.type == "Rectangle")
    {
        //# render rectangle
    }
    else if (element.type == "Freehand")
    {
        //# render freehand
    }

    return {};
}
```

```
  var avg = calcAverage(values);
  return avg;
}
// return the average of the values
function calcAverage(values){}
```

Note: all function calls are pass by value (i.e., if you pass an object to a function, and the function changes the object, you will not see the changes).

- JAVASCRIPT TUTORIAL
- AVAILABLE DATA TYPES

> String
> Number
> Boolean
> Element

Elements are shapes in a drawing and may be of type "Rectangle", "Freehand", or "Line".
For a Rectangle Element, the vertices describes the four corners and are listed in clockwise order (the first vertices can be any of the four).
For a Line or Freehand Element, the vertices describes the path of the Element.
All Elements are identified by a unique id, beginning at 0.

DATA STRUCTURE

```
{
  id: Number
  type: String
  vertices: Position[]
}
```

EXAMPLES: Rectangle

```
{
  "id": 1,
```

```
12 function moveElement(mouseDownPos, mouseCurrPos, origElem){
13     if (!validPosition(mouseDownPos) || !validPosition(mouseCurrPos) || !va
14         return null;
15     }
16     if (!validElementType(origElem.type)) {
17         return null;
18     }
19     var element = {
20         "id": origElem.id,
21         "type": origElem.type,
22         "vertices": Array.prototype.slice(origElemVerticies)
23     };
24     var xOffset = mouseCurrPos.x - mouseDownPos.x;
25     var yOffset = mouseCurrPos.y - mouseDownPos.y;
26     for (var vertex in origElem.vertices) {
27         var newPos;
28         //# adjust newPos based on xOffset and yOffset
29         element.vertices.push(newPos);
30     }
31     return element;
32 }
33 // validElementType -- Returns true if the given string matches one of "Rec
34 // "Freehand", or "Line".
35 function validElementType(str) {}
```

1 PROBLEM(S) FOUND:

1. Line 22: 'origElemVerticies' is not defined.

Skip Submit

WRITE A TEST 10 pts

For the following test case, can you implement a test, providing a JSON object literal for each input parameter and for the expected return value?
Tip: Descriptions of the data types are on the left with examples you can copy and paste.

■ TEST CASE DESCRIPTION

Report an issue in the test case

create a FreeHand

■ FUNCTION SIGNATURE

```
/**
  Creates a new Element, as specified by the Action and using the
  provided current mouse coordinates.

  For Freehand Elements, which contain many positions, copies the
  existing list of positions from prevElement, if it exists.

  @param Number x , the current x position of Mouse - left is 0
  @param Number y , the current y position of Mouse - top is 0
  @param Action action , the current action
  @param Element prevElement , previous version of the Element - may be null

  @return Element
**/
function createElement(x, y, action, prevElement)
```

■ AVAILABLE DATA TYPES

- String
- Number
- Boolean
- Element
- Position
- LineSegment
- RenderInstructions
- Action

INPUT PARAMETERS

x (Number)

Paste Example

y (Number)

Paste Example

action (Action)

Paste Example

prevElement (Element)

Paste Example

RETURN VALUE

(Element)

Skip Submit

```
fied by the Action and using the
ces.

ntain many positions, copies the
 prevElement, if it exists.

 position of Mouse - left is 0
 position of Mouse - top is 0
ent action
evious version of the Element - may be null


on, prevElement)
```

INPUT PARAMETERS

x (Number)

```
10
```

Paste Example

y (Number)

```
10
```

Paste Example

action (Action)

```
{
  "type": "Freehand",
  "elementID": 0,
  "mouseDownX": 10,
  "mouseDownY": 10
}
```

Paste Example

prevElement (Element)

```
nu
```

Unexpected end of input

Paste Example

RETURN VALUE

Enabling fine-grained contributions

Local changes to a single function or test

Type system, request functionality through pseudocalls

Preconfigured environment

Tutorial system introducing microtasks

Encourage mass **parallelism**

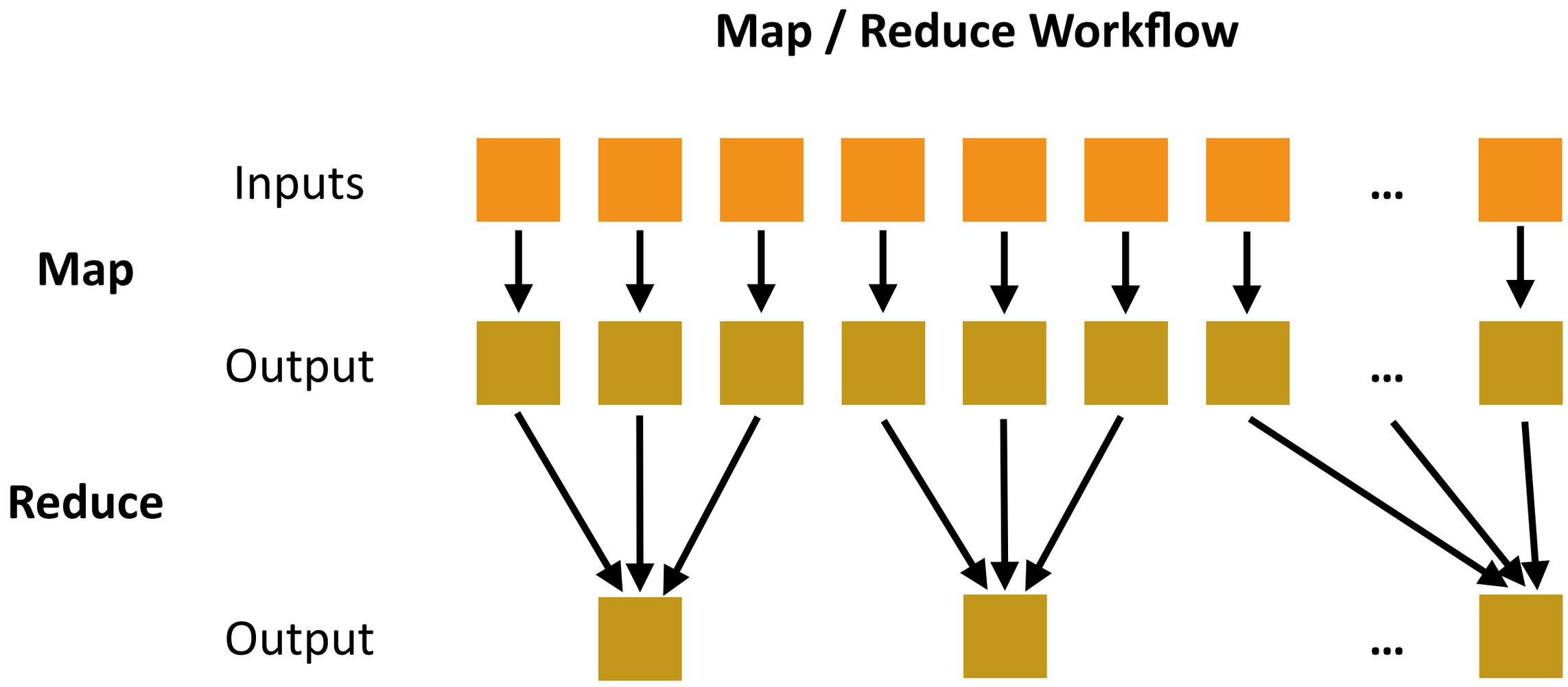
Collective ownership of code

Time & space box contributions

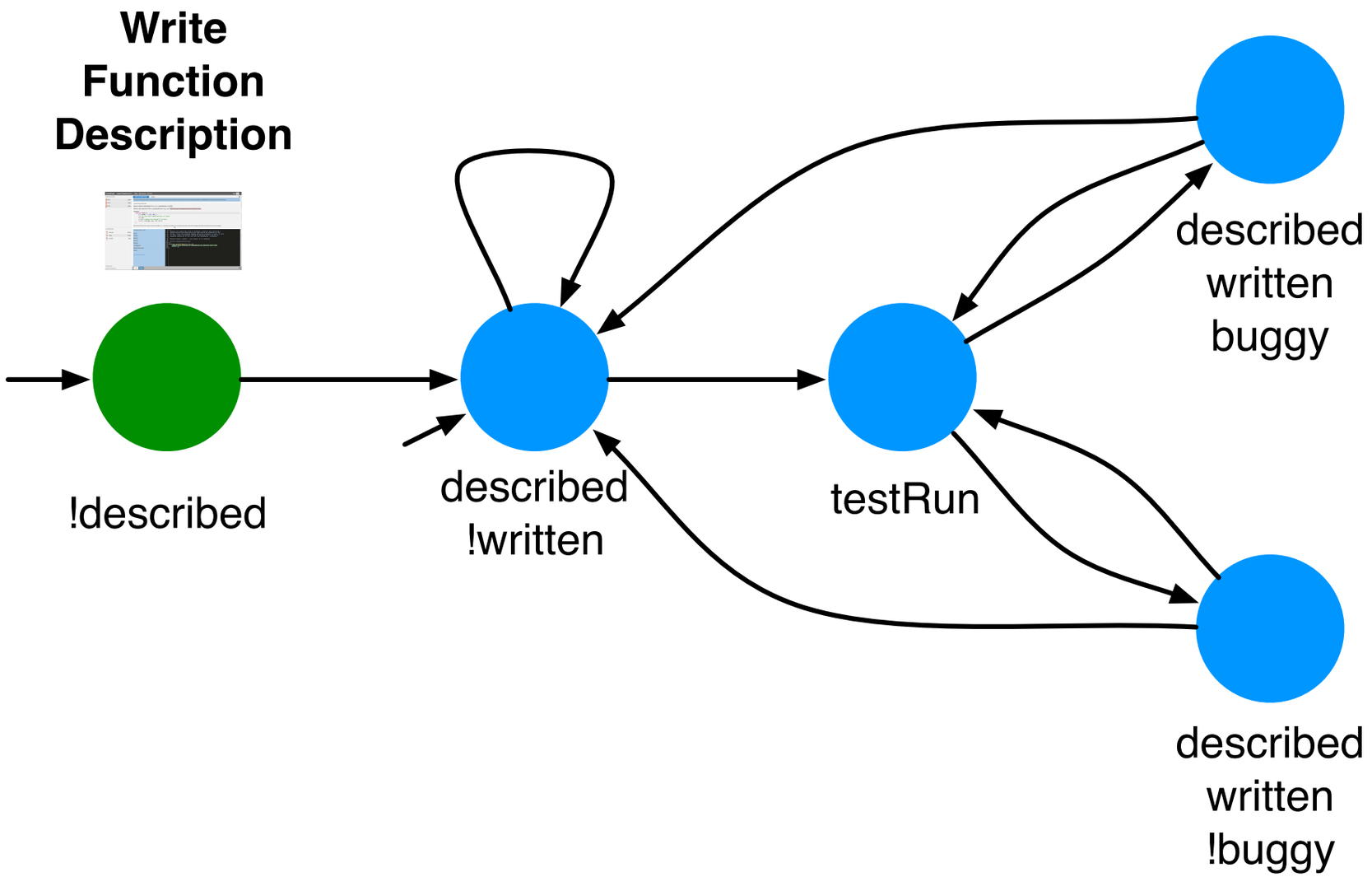
Automatic microtask generation

Existing microtask workflows enumerate microtasks statically (e.g., map-reduce)

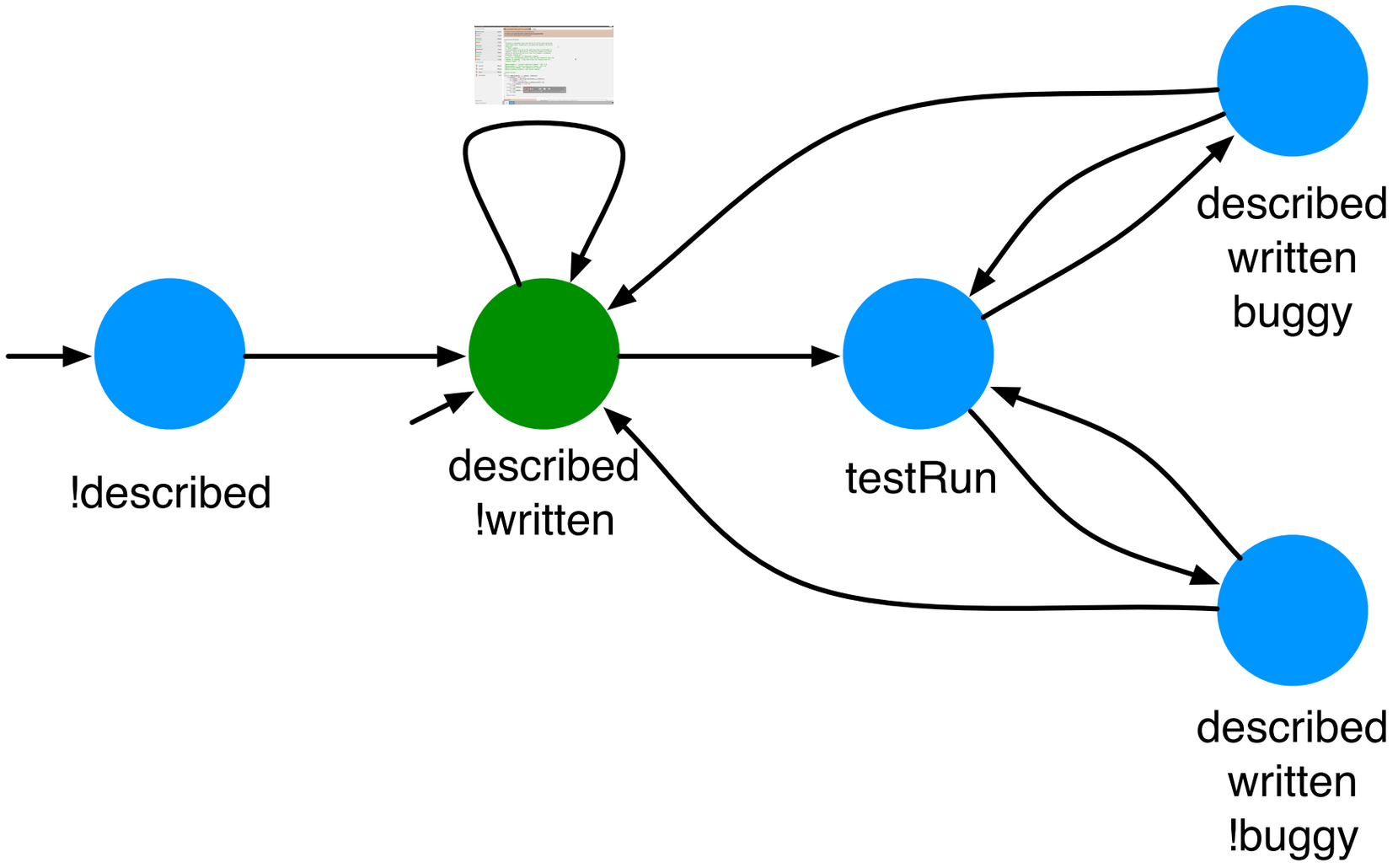
Programming work cannot be statically enumerated upfront.
— Work created in response to work completed.



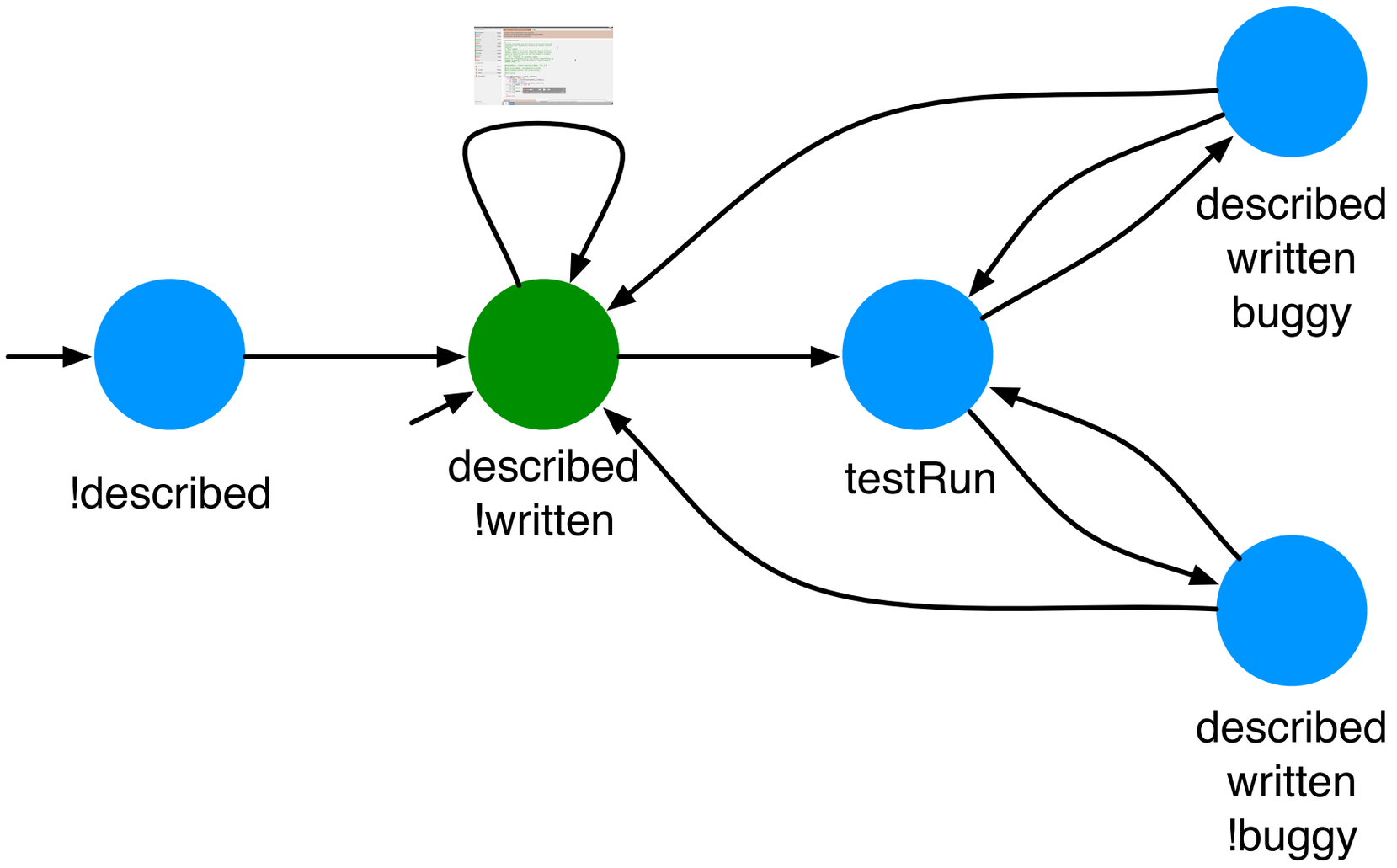
Function state machine - example

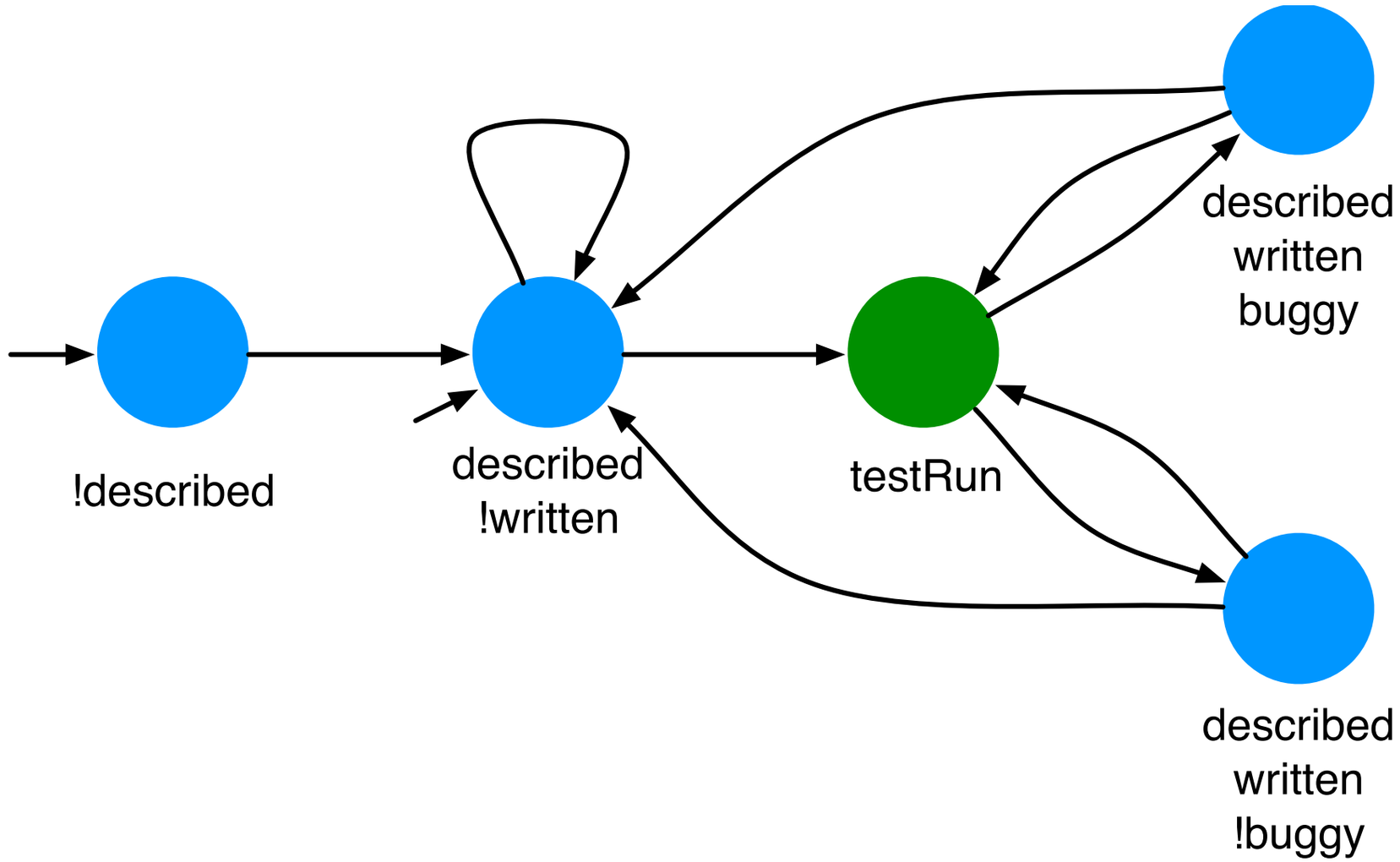


Edit a Function

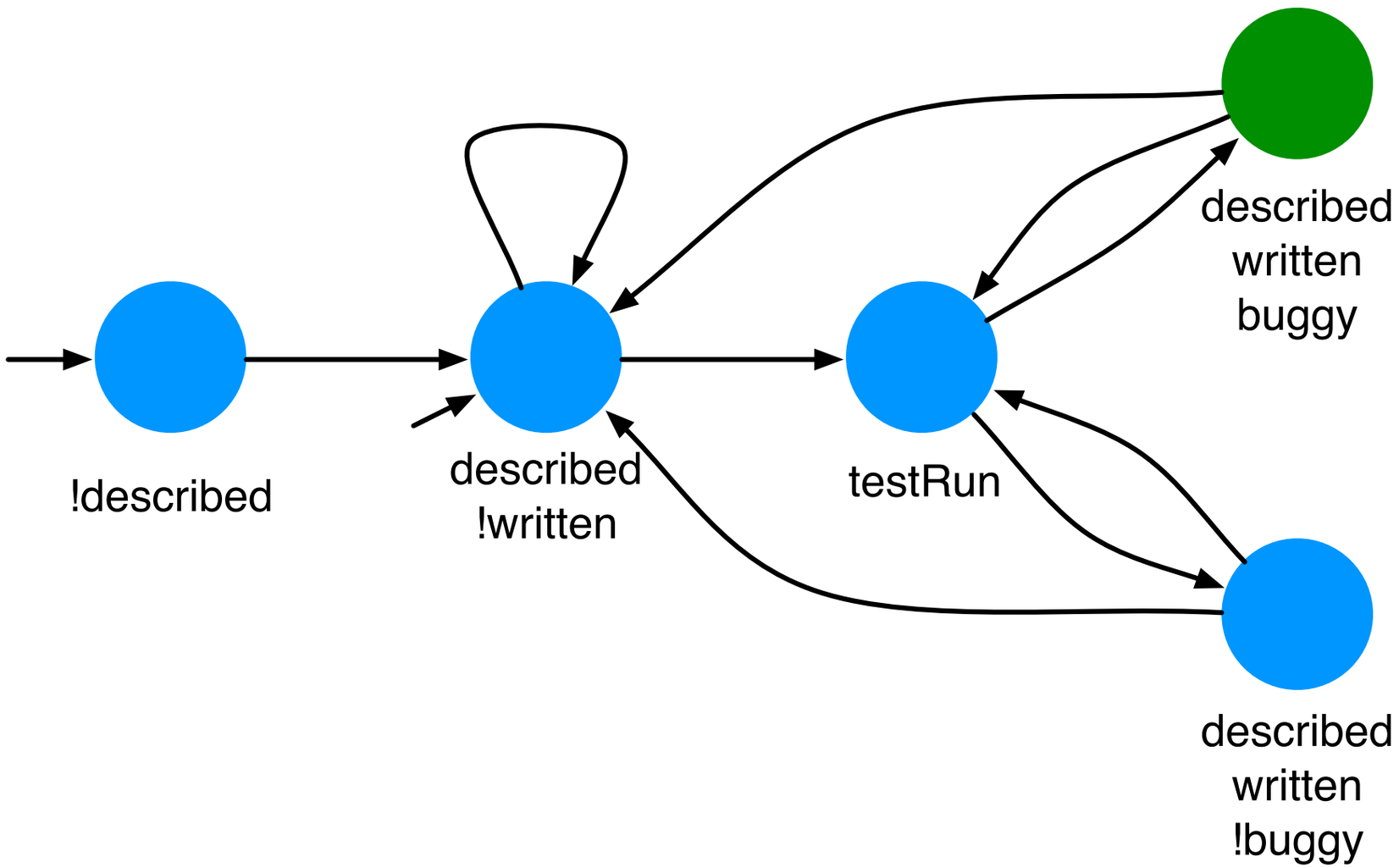


Edit a Function

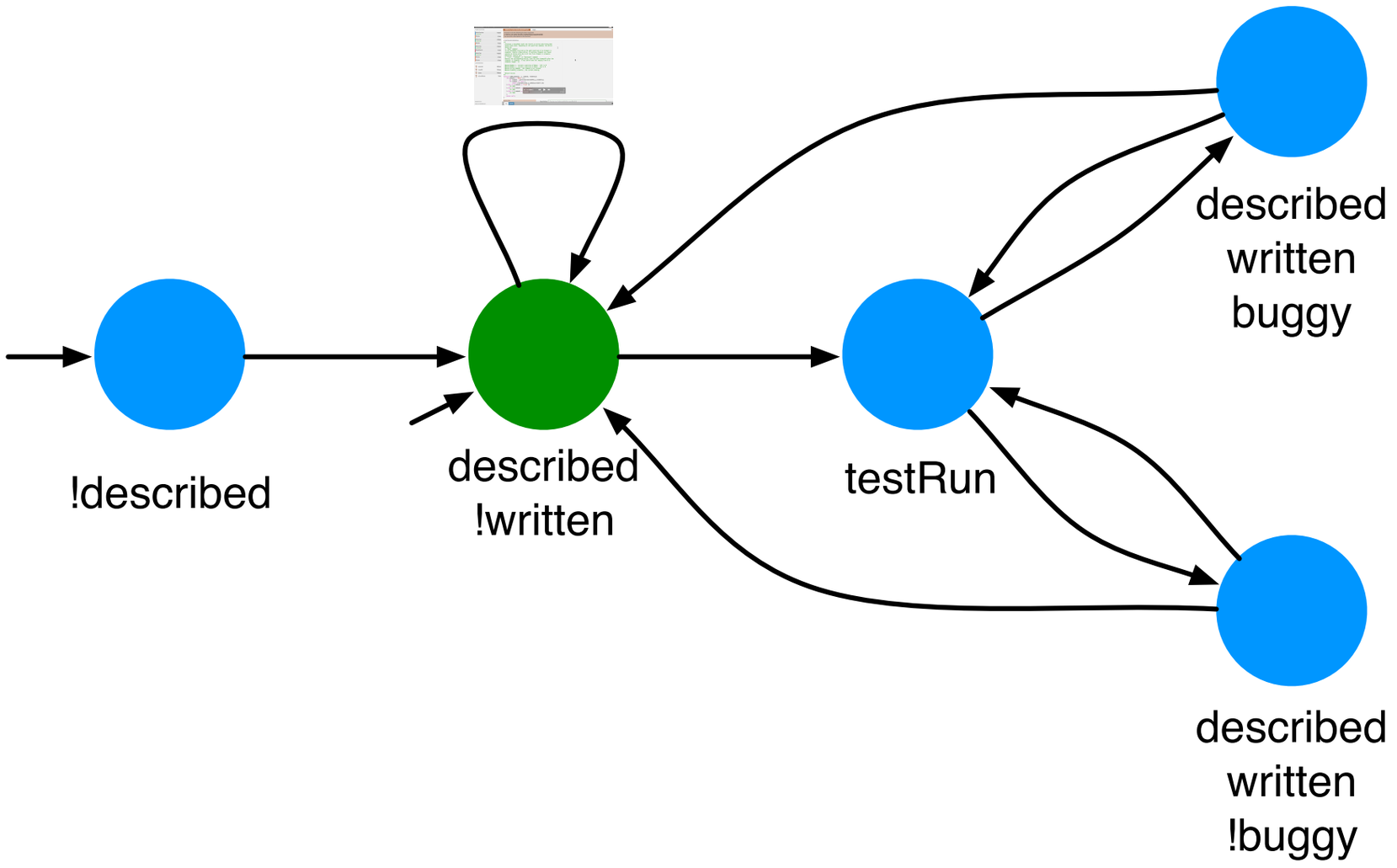


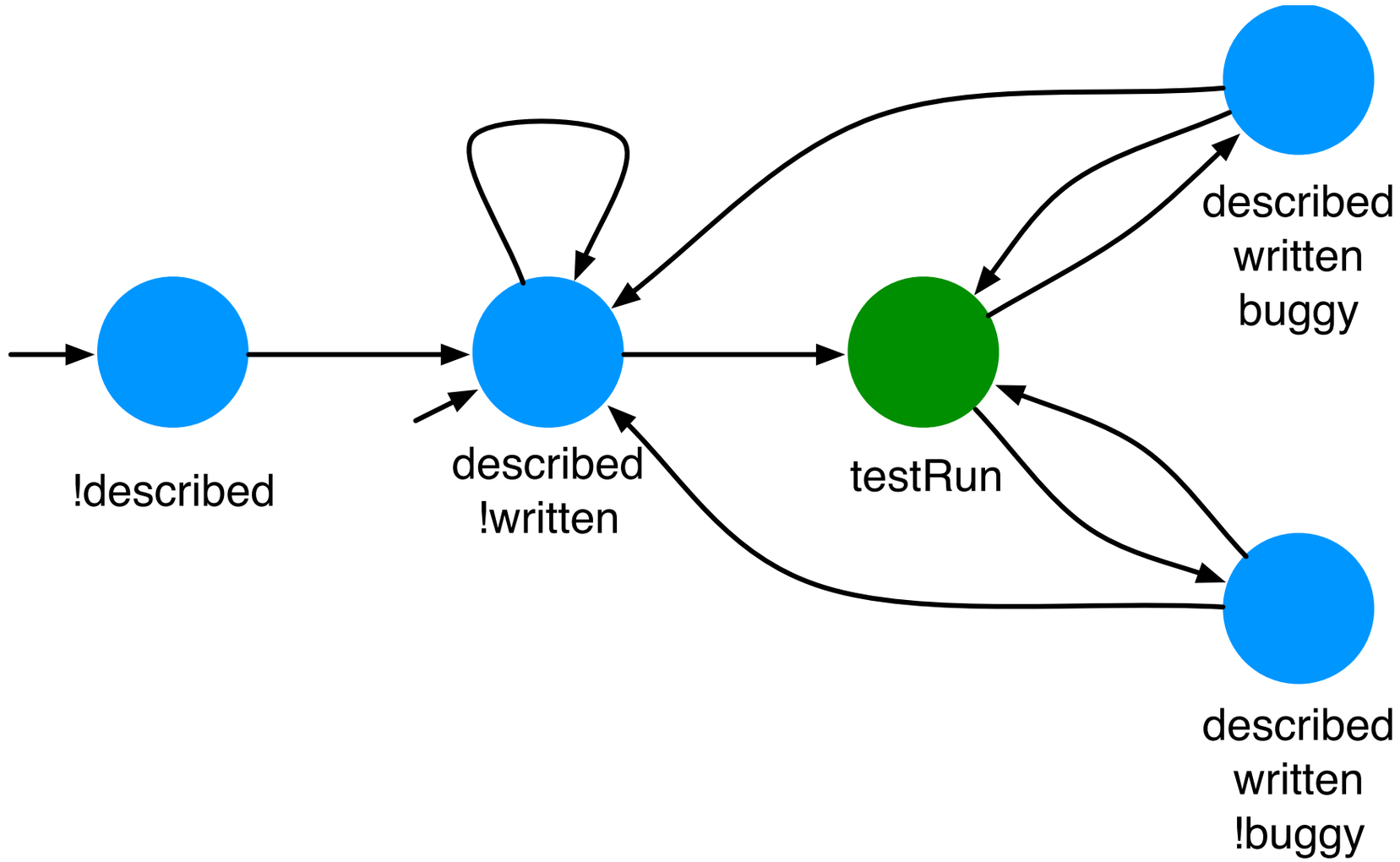

described
written
buggy
!described
described
!written
testRun
described
written
!buggy

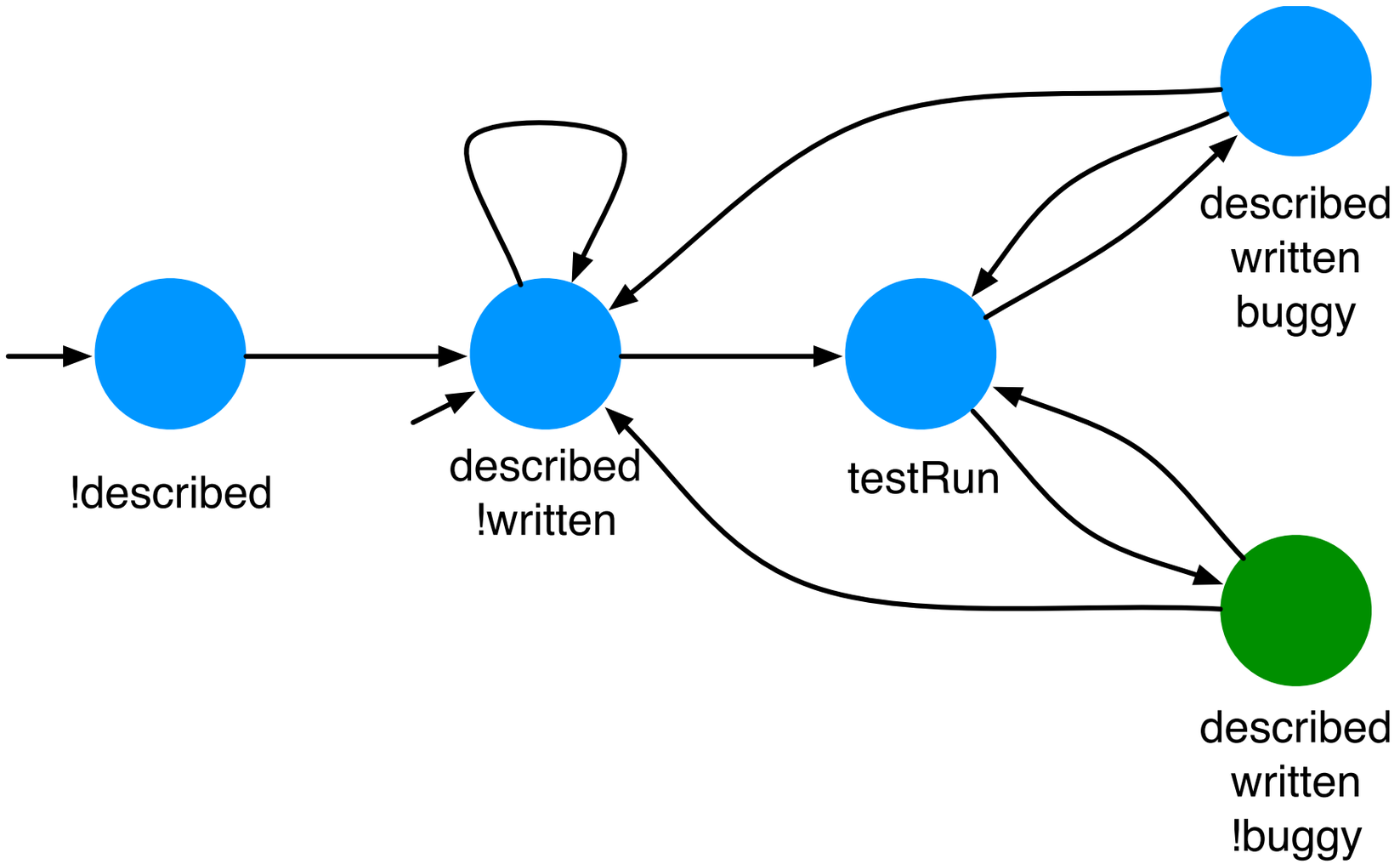
Debug a Test Failure



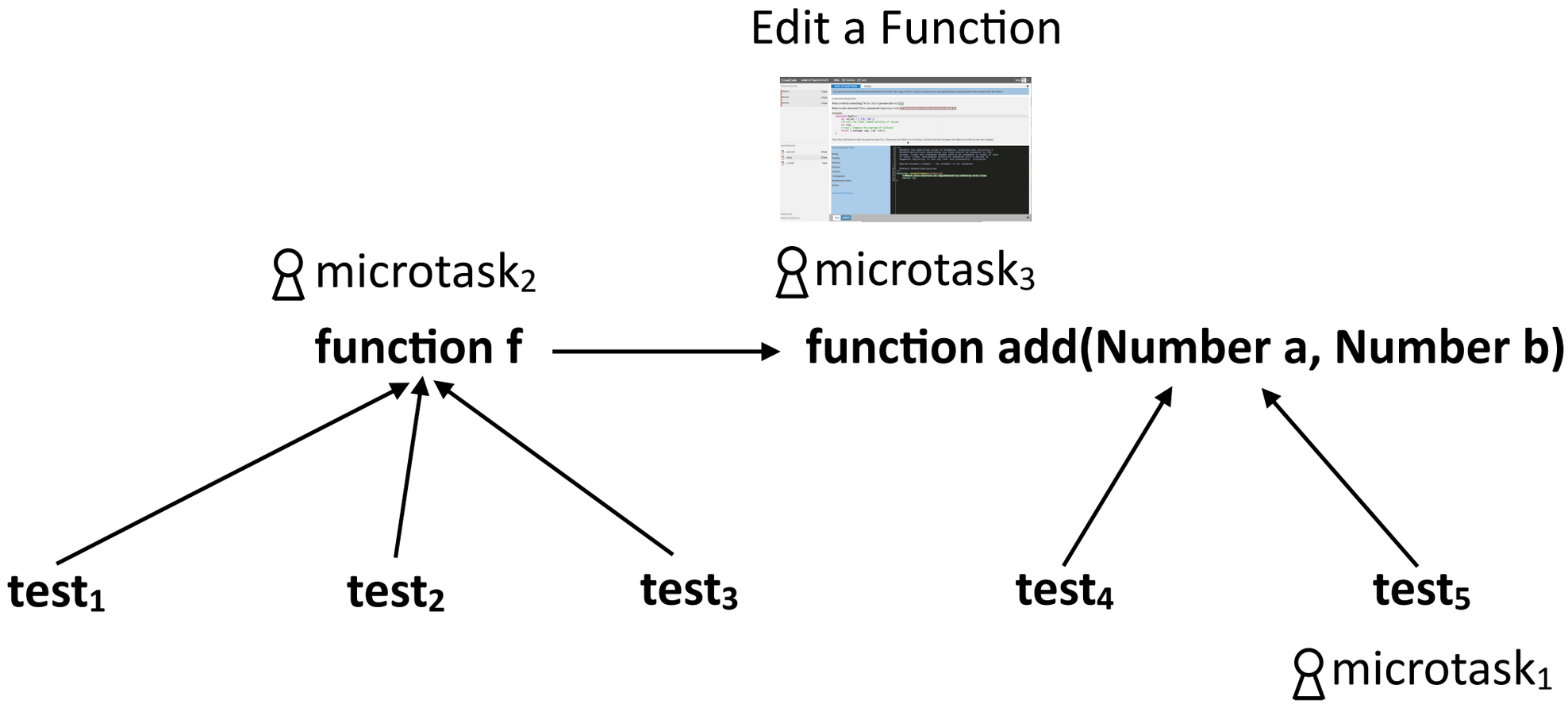
Edit a Function




described
written
buggy
!described
described
!written
testRun
described
written
!buggy

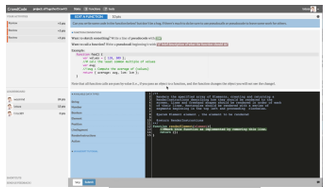

described
written
buggy
!described
described
!written
testRun
described
written
!buggy

Coordinating work



- One microtask in progress per artifact

Edit a Function



microtask$_2$

function f

microtask$_3$

~~**Number a, Number b**~~

function add(Number[] numbers)

test$_1$

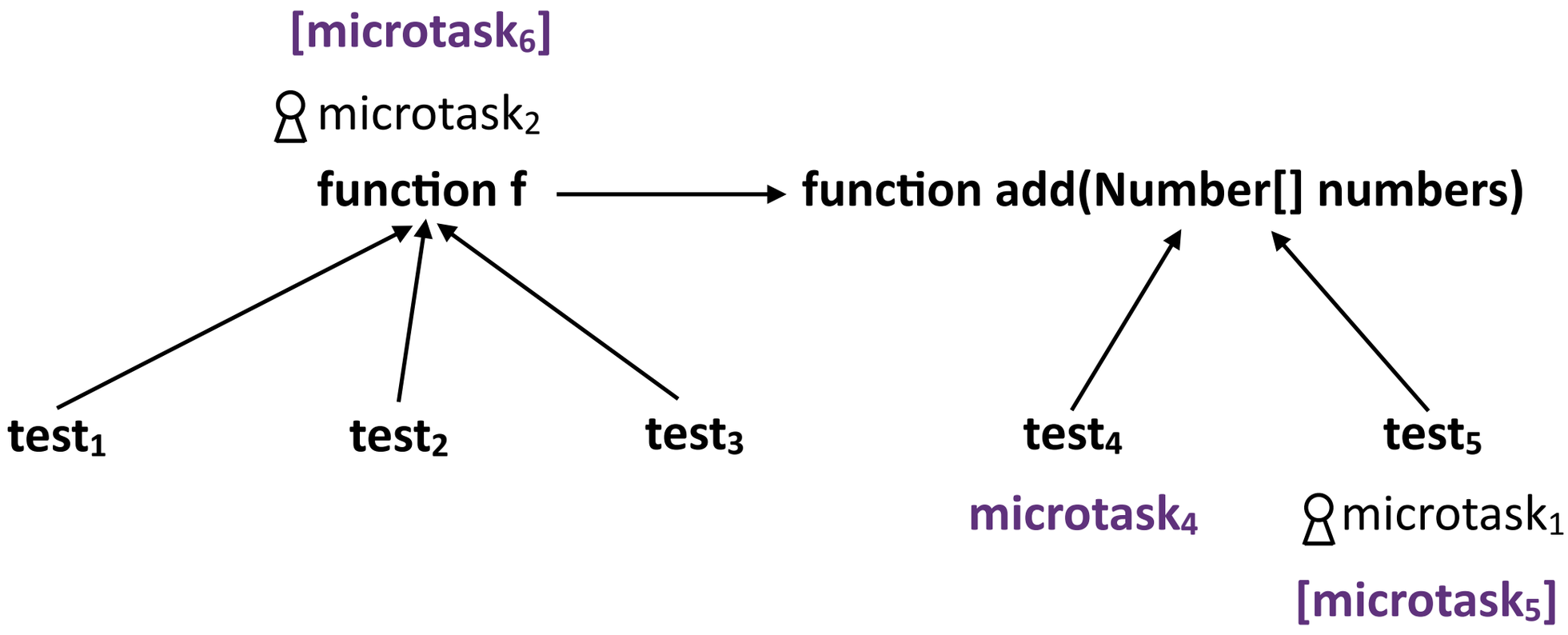
test$_2$

test$_3$

test$_4$

test$_5$

microtask$_1$

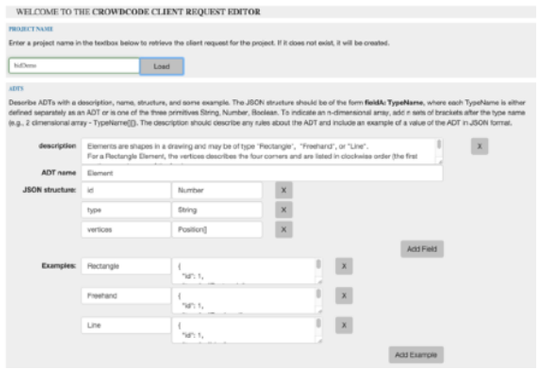

[microtask6]
microtask2
function f
function add(Number[] numbers)
test1
test2
test3
test4
test5
microtask4
microtask1
[microtask5]

Ensure quality through iterative work

- Key principle: any crowd authored content can be revised by crowd
- Sketch & revision w/ small contributions
- Report issues w/ dependent artifacts that can’t be directly edited
- Review & test

Workflow

client request



API

API function

API function

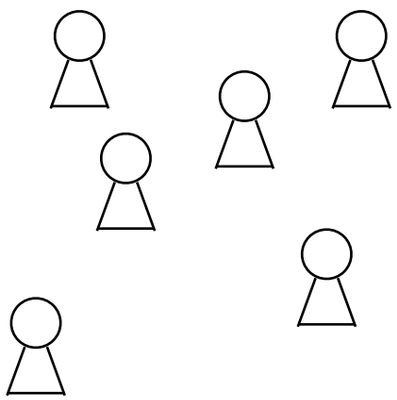
API function

function

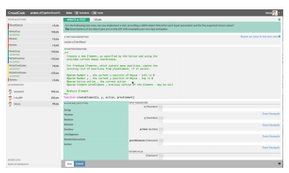
function

function

implemented library



crowd



microtask

microtask

microtask

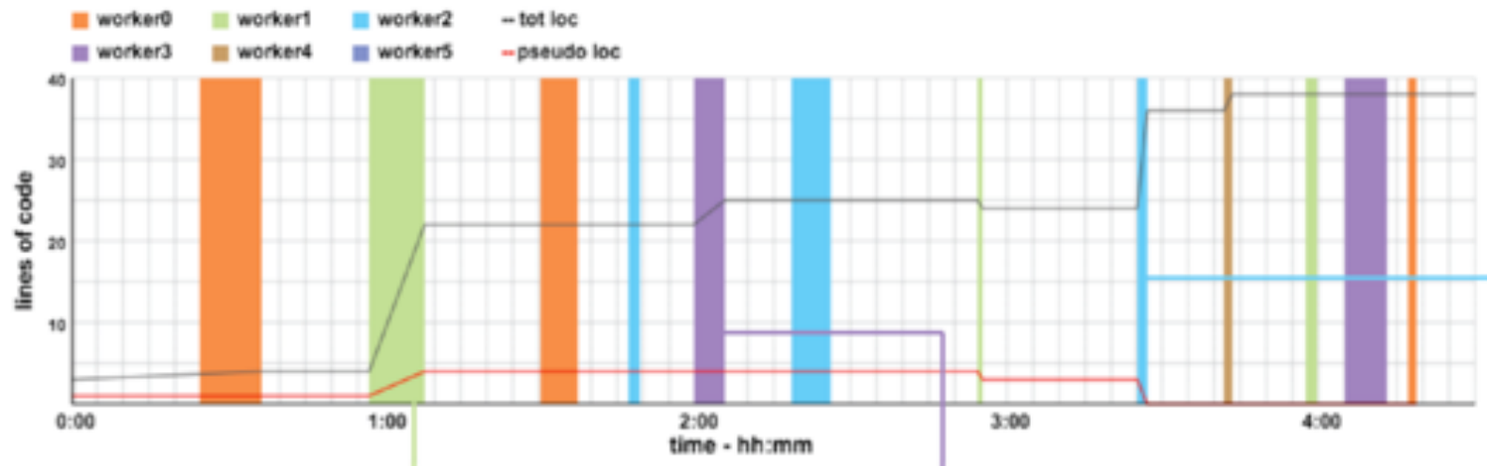
microtask queue

Results - Automatically generating microtasks

- Implemented 22 functions (490 lines of code), including 14 new functions
- Created 149 unit tests (2920 lines of code)

Microtask type	**Completed**		**Skipped**		**Reissued**		**Median time** (m:s)		**Total time** (h:m:s)	
Session	A	B	A	B	A	B	A	B	A	B
Review	260	227	22	22	-	-	1:27	1:14	9:29:32	6:43:43
Write test	158	102	22	7	40	41	1:29	1:21	6:35:41	3:15:12
Write function	44	56	25	21	16	22	4:57	2:31	3:59:28	3:40:03
Write test cases	40	30	9	4	11	13	3:50	2:28	2:57:02	2:05:53
Debug test failure	14	18	5	6	1	4	2:32	4:00	0:57:22	1:21:21
Write function	8	16	3	0	0	10	3:12	2:44	0:30:21	1:03:58
Write call	7	9	2	2	0	3	1:37	2:28	0:15:02	0:36:49
Reuse search	9	10	3	0	-	3	0:42	1:35	0:06:37	0:22:33
Total	540	468	91	62	68	96			24.51.0	19:09:32
Overall total	**1008**		**153**		**16**				**44:00:37**	

Results - Small contributions



A

```
function createAction(mouseDownPos, command, elements) {
 //# mark this function as complete removing this line
 //# mark this function as complete by removing this line
 if(command  == "Move"){
  var el = null;
  for (el in elements) {
   if (onOutline(mouseDownPos, el)) {
    return {
     "type":"Move",
     "elementId":el.id,
     "mouseDownPos":mouseDownPos
    };
   }
  }
 }else if(command == "Line" || command == "Freehand"
                || command == "Rectangle"){
  //# handle Line
  //# handle Freehand
  //# handle Rectangle
 }else{
  return null;
 }
 return null;
}
```

B

```
function createAction(mouseDownPos, command, elements) {
 //# mark this function as complete by removing this line
 if(command  == "Move"){
  var el = null;
  for (el in elements) {
   if (isOnOutline(mouseDownPos, el)) {
    return {
     "type":"Move",
     "elementId":el.id,
     "mouseDownPos":mouseDownPos
    };
   }
  }
 } else if(command == "Line" || command == "Freehand"
                || command == "Rectangle"){
  //# handle Line
  //# handle Freehand
  //# handle Rectangle
 }else{
  return null;
 } else {
  var nextId = getNextId(elements);
  if (command == "Line") {
  //# handle Line and return it
  } else if (command == "Freehand") {
  //# handle Freehand and return it
  } else if (command == "Rectangle") {
  //# handle Rectangle and return it
  }
 }
 return null;
}
```

C

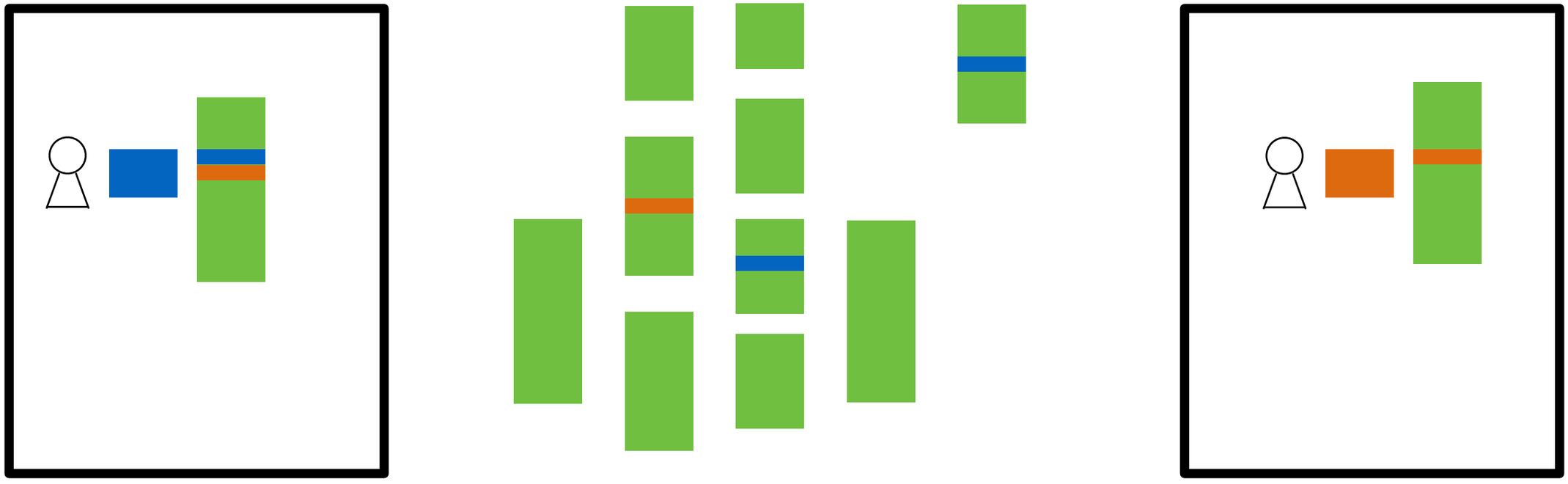
```
function createAction(mouseDownPos, command,
                elements) {
 if(command  == "Move"){
  var el = null;
  for (el in elements) {
   if (isOnOutline(mouseDownPos, el)) {
    return {
     "type":"Move",
     "elementId":el.id,
     "mouseDownPos":mouseDownPos
    };
   }
  }
 } else {
  var nextId = getNextValidId(elements);
  if (command == "Line") {
  //# handle Line and return it
   return {
    "type": "Line",
    "elementId": nextId,
    "mouseDownPos": mouseDownPos
   };
  } else if (command == "Freehand") {
  //# handle Freehand and return it
   return {
    "type": "Freehand",
    "elementId": nextId,
    "mouseDownPos": mouseDownPos
   };
  } else if (command == "Rectangle") {
  //# handle Rectangle and return it
   return {
    "type": "Rectangle",
    "elementId": nextId,
    "mouseDownPos": mouseDownPos
   };
  }
 }
 return null;
}
```

Results - Fixing bugs

```
function createElement(id, type, mouseDownPos, mouseCurrPos, prevElement)
{
        if (typeof id != "number" || typeof type != "string"
            || typeof mouseDownPos != typeof {} || typeof mouseCurrPos != typeof {}
            || typeof prevElement != typeof {}) {
            return null;
        }
        // check if mouse coordinates are within bounds
        if (!validPosition(mouseDownPos) || !validPosition(mouseCurrPos))
            return null;

        var element = {
            "id": id,
            "type": type
        };
        if (type === "Rectangle") {
            // Create in clockwise order
            element.verticies = [
                mouseDownPos,
                {x: mouseCurrPos.x, y: mouseDownPos.y},
                mouseCurrPos,
                {x: mouseDownPos.x, y: mouseCurrPos.y}
            ];
        } else if (type === "Line") {
            element.vertices = [mouseDownPos, mouseCurrPos];
        } else { //type === "Freehand"
            if (prevElement !== null && mouseCurrPos === prevElement.vertices[0]) {//we have come "full circle"
                element.vertices = prevElement.vertices;  //just copy those
            } else if (prevElement === null) {
                element.vertices = [mouseDownPos, mouseCurrPos];
            } else {
                // Build off of prevElement
                element.vertices = prevElement.vertices; //if prevprevElement is not null, copy its vertices to new element


            }
        }
    return element;
}
```

```
function createElement(id, type, mouseDownPos, mouseCurrPos, prevElement)
{
        if (typeof id != "number" || typeof type != "string"
            || typeof mouseDownPos != typeof {} || typeof mouseCurrPos != typeof {}
            || typeof prevElement != typeof {}) {
            return null;
        }
        // check if mouse coordinates are within bounds
        if (!validPosition(mouseDownPos) || !validPosition(mouseCurrPos))
            return null;

        var element = {
            "id": id,
            "type": type
        };
        if (type === "Rectangle") {
            // Create in clockwise order
            element.vertices = [
                mouseDownPos,
                {x: mouseCurrPos.x, y: mouseDownPos.y},
                mouseCurrPos,
                {x: mouseDownPos.x, y: mouseCurrPos.y}
            ];
        } else if (type === "Line") {
            element.vertices = [mouseDownPos, mouseCurrPos];
        } else { //type === "Freehand"
            if (prevElement === null) {


                element.vertices = [mouseDownPos, mouseCurrPos];
            } else if (prevElement.type === type ) {
                // Build off of prevElement
                element.vertices = prevElement.vertices; //if prevprevElement is not null, copy its vertices to new element
                element.vertices.push(mouseCurrPos);
            } else {
                return null;
            }
        }
    return element;
}
```

Still a need for coordinating crosscutting information



- Much of the crosscutting information related to **decisions**
  - Interpretation of requirements & specifications
  - Representation of state in data structures
- Building good interface often as much work (or more) than doing implementation
  - Need for global view for the crowd to be more involved in crafting API

Scaffolding coordination and knowledge sharing

Idea

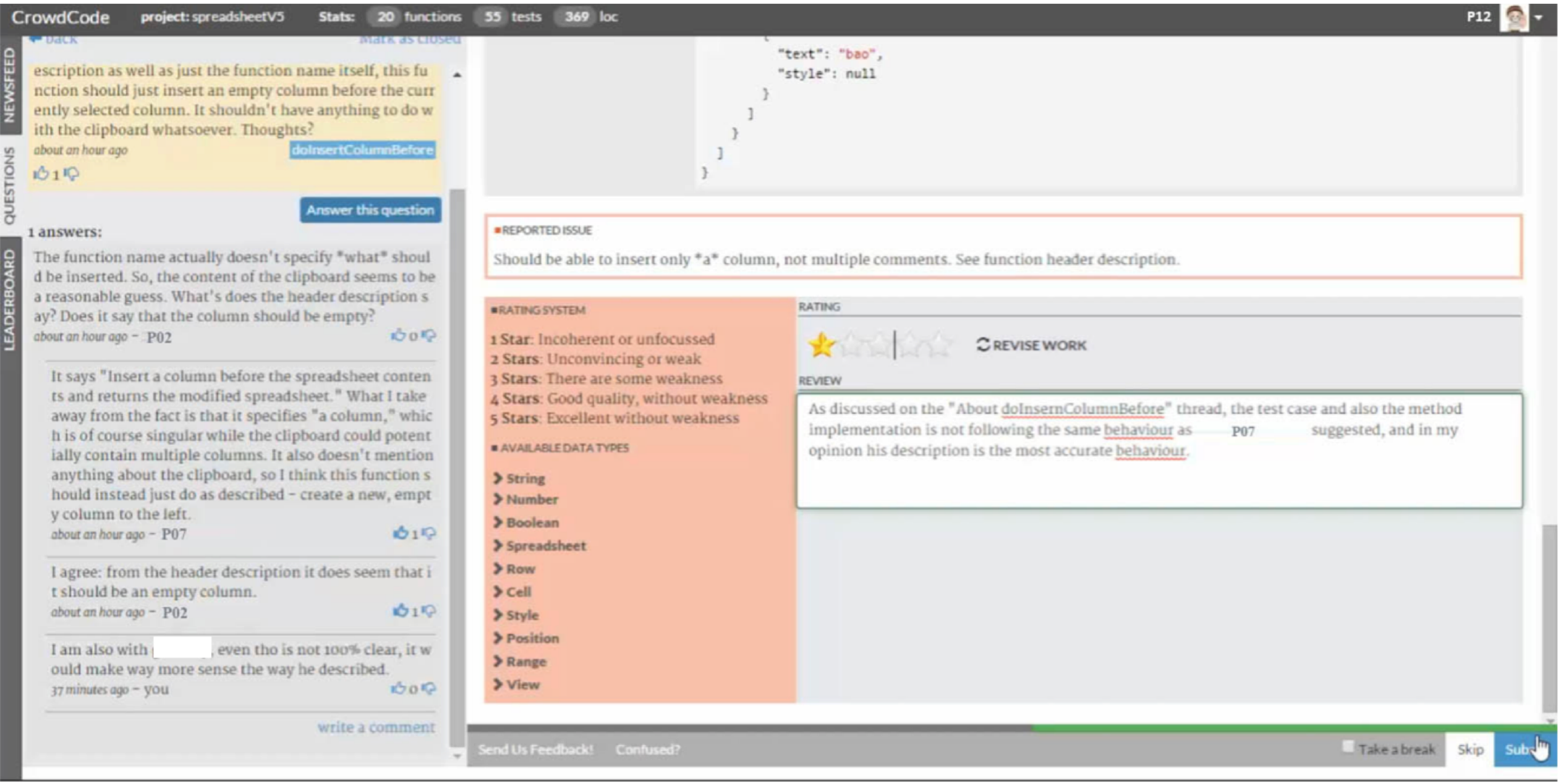
make design decisions **explicit**

link artifacts to design decisions

enable developers to coordinate by **discussing** design decisions

LaToza, Di Lecce, Ricci, Towne, van der Hoek. Ask the crowd: Scaffolding coordination and knowledge sharing in microtask programming. VL/HCC 2015.

Extended environment w/ Q&A system directly connected to code



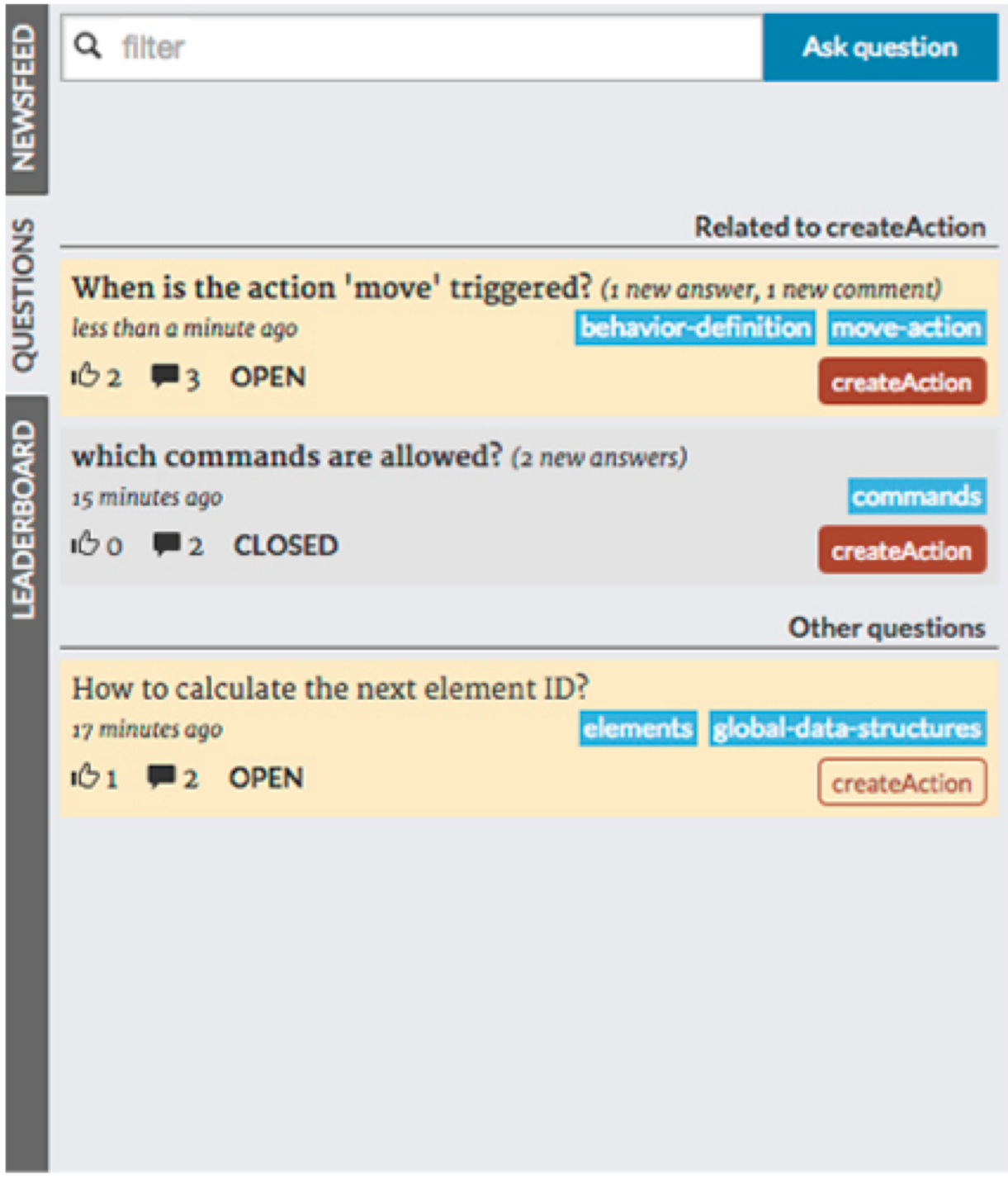
Developers can search for existing answers

Developers can search questions with full text **filter**

Questions marked as **closed** (yellow) or **open** (gray)

Questions marked as **relevant** to current artifact (top) and all other questions

If no answer, can ask new question.



Developers can discuss questions and answers

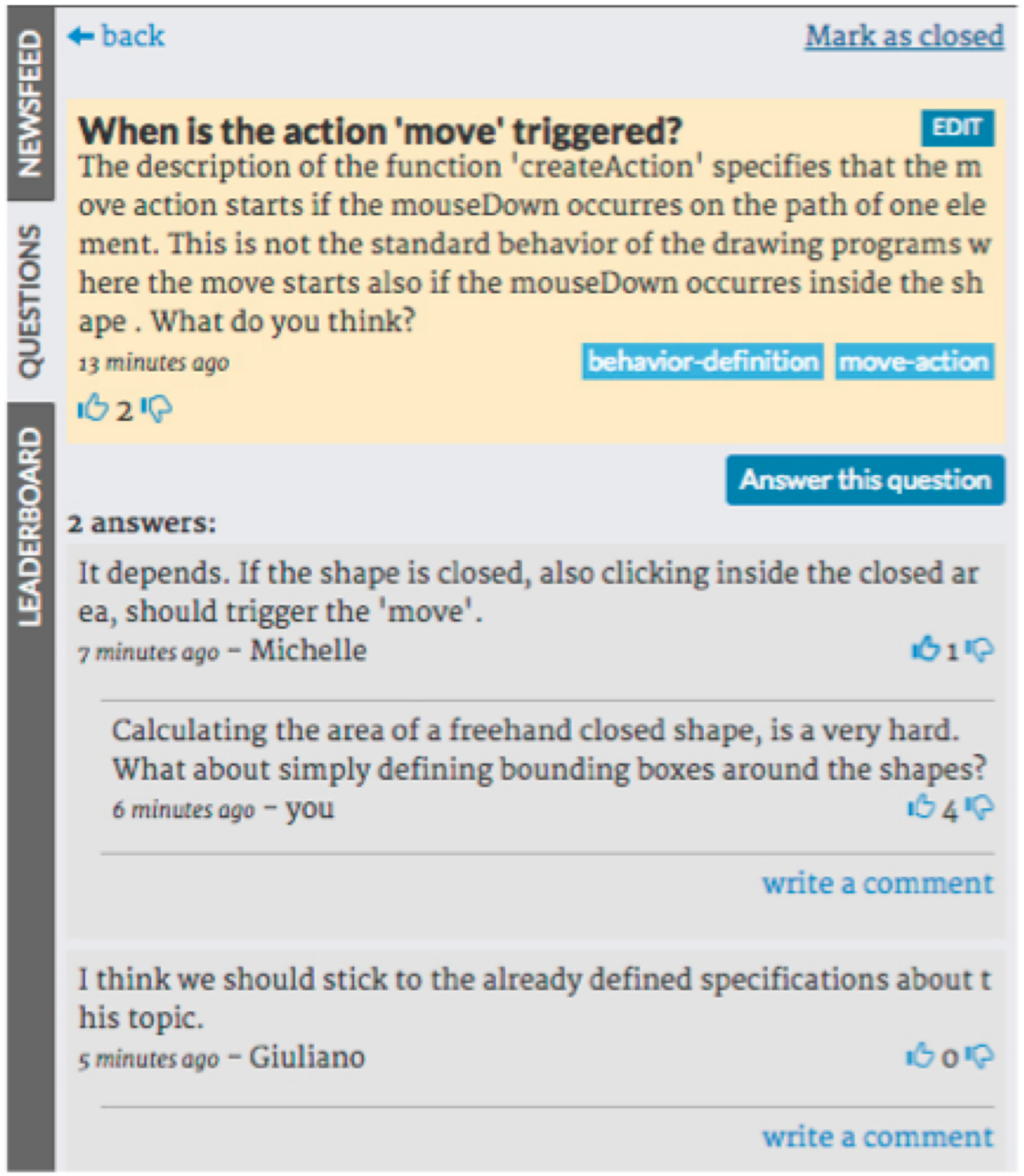
Any worker may at **any time** discuss question.

Threaded into answers & comments.

Up vote & down-vote.

Discussion **synchronous** (real time updates) & **asynchronous** (notifications on issues developers follows)

Question title, text, tags, open/closed **collaboratively** editable.



Organizing discussion around decisions helps structure coordination

Workers used Q&A to ask questions on variety of topics
— crosscutting decisions
— clarify function descriptions
— state decisions already made, indicating convention
— ask questions about using CrowdCode environment
— explicit coordination

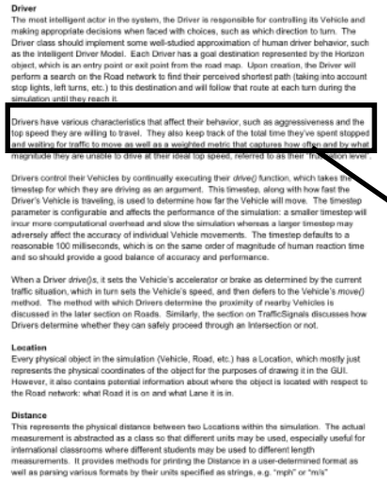
Compared to unstructured global chat, several preferred Q&A
— “easier to transmit and spread the information” (P2)
— easier to “find relevant issues very easily” (P5)
— better than “tutorials and documentation” (P18)
— less distracting, easier to reach agreement, easier to get up to speed

Some limitations

- Lots of work for client to craft the right API
- Only for code, not UI
- No independent consideration of alternatives
- Independence important for generating diverse ideas
- Valuable for important design decisions?

Software design recombination

Consider alternatives separately created by **many**, iteratively selecting and **recombining** the best ideas



Driver
The most intelligent actor in the system, the Driver is responsible for controlling its Vehicle and making appropriate decisions when faced with choices, such as which direction to turn. The Driver class should implement some well-studied approximation of human driver behavior, such as the Intelligent Driver Model. Each Driver has a goal destination represented by the Horizon object, which is an entry point or exit point from the road map. Upon creation, the Driver will perform a search on the Road network to find their perceived shortest path (taking into account stop lights, left turns, etc.) to this destination and will follow that route at each turn during the simulation until they reach it.

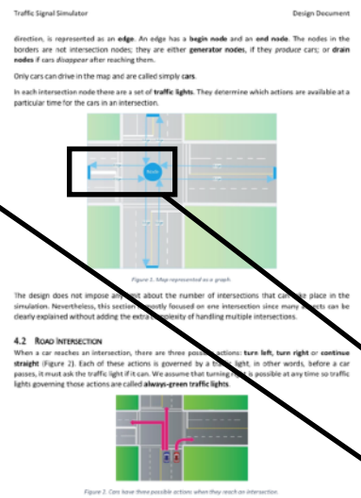
Drivers have various characteristics that affect their behavior, such as aggressiveness and the top speed they are willing to travel. They also keep track of the total time they've spent stopped and waiting for traffic to move, as well as a weighted metric that captures how often and by what magnitude they are unable to drive at their ideal top speed, referred to as their "frustration level."

Drivers control their Vehicles by continually executing their *drive()* function, which takes the timestep for which they are driving as an argument. This timestep, along with how fast the Driver's Vehicle is traveling, is used to determine how far the Vehicle will move. The timestep parameter is configurable and affects the performance of the simulation: a smaller timestep will incur more computational overhead and slow the simulation whereas a larger timestep may adversely affect the accuracy of individual Vehicle movements. The timestep defaults to a reasonable 100 milliseconds, which is on the same order of magnitude of human reaction time and so should provide a good balance of accuracy and performance.

When a Driver *drive()*s, it sets the Vehicle's accelerator or brake as determined by the current traffic situation, which in turn sets the Vehicle's speed, and then defers to the Vehicle's *move()* method. The method with which Drivers determine the proximity of nearby Vehicles is discussed in the later section on Roads. Similarly, the section on TrafficSignals discusses how Drivers determine whether they can safely proceed through an Intersection or not.

Location
Every physical object in the simulation (Vehicle, Road, etc.) has a Location, which mostly just represents the physical coordinates of the object for the purposes of drawing it in the GUI. However, it also contains potential information about where the object is located with respect to the Road network: what Road it is on and what Lane it is in.

Distance
This represents the physical distance between two Locations within the simulation. The actual measurement is abstracted as a class so that different units may be used, especially useful for international classrooms where different students may be used to different length measurements. It provides methods for printing the Distance in a user-determined format as well as parsing various formats by their units specified as strings, e.g. "mph" or "m/s"



Traffic Signal Simulator — Design Document

direction, is represented as an **edge**. An edge has a **begin node** and an **end node**. The nodes in the borders are not intersection nodes; they are either **generator nodes**, if they produce cars; or **drain nodes** if cars disappear after reaching them.

Only cars can drive in the map and are called simply **cars**.

In each intersection node there are a set of **traffic lights**. They determine which actions are available at a particular time for the cars in an intersection.

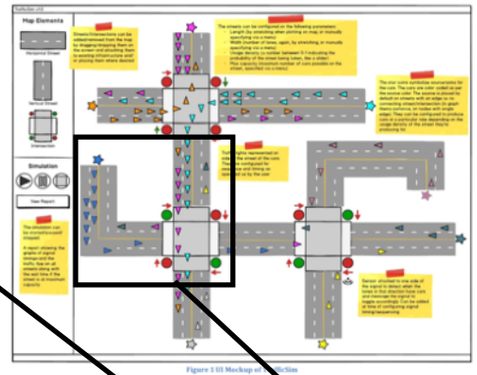
Figure 1. Map represented as a graph.

The design does not impose any limit about the number of intersections that can take place in the simulation. Nevertheless, this section is mostly focused on one intersection since many aspects can be clearly explained without adding the extra complexity of handling multiple intersections.

4.2 Road Intersection

When a car reaches an intersection, there are three possible actions: **turn left, turn right** or **continue straight** (Figure 2). Each of these actions is governed by a traffic light, in other words, before a car passes, it must ask the traffic light if it can. We assume that turning right is possible at any time so traffic lights governing those actions are called **always-green traffic lights**.

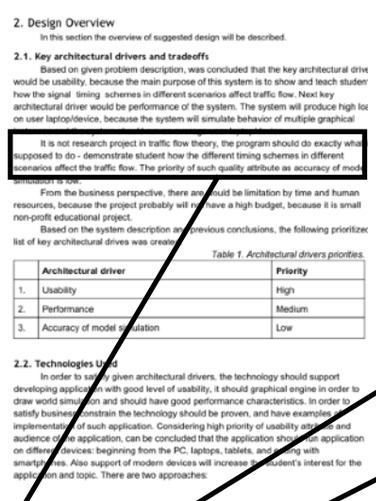
Figure 2. Cars have three possible actions when they reach an intersection.



Prototype
The following image shows a high level mock up of how TrafficSim looks like. This is a first cut design trying to capture and show the various elements of the use-case in a holistic view. This will be referred to in the detailed design later.

Figure 1 UI Mockup of TrafficSim

The post-it notes above explain provide more detail about what the mockup is showing. You may have to zoom in further to read it.



2. Design Overview

In this section the overview of suggested design will be described.

2.1. Key architectural drivers and tradeoffs

Based on given problem description, was concluded that the key architectural drive would be usability, because the main purpose of this system is to show and teach studen how the signal timing schemes in different scenarios affect traffic flow. Next key architectural driver would be performance of the system. The system will produce high lo on user laptop/device, because the system will simulate behavior of multiple graphical

It is not research project in traffic flow theory, the program should do exactly wha supposed to do - demonstrate student how the different timing schemes in different scenarios affect the traffic flow. The priority of such quality attribute as accuracy of mod simulation is low.

From the business perspective, there are should be limitation by time and human resources, because the project probably will have a high budget, because it is small non-profit educational project.

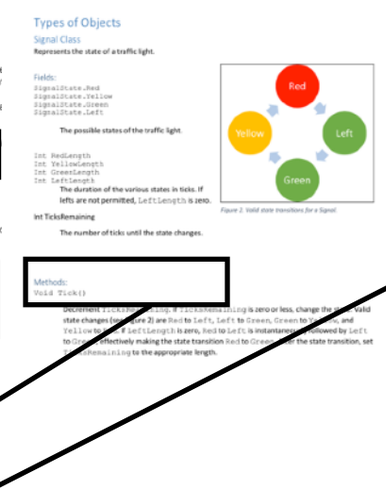
Based on the system description and previous conclusions, the following prioritized list of key architectural drives was created:

Table 1. Architectural drivers priorities.

	Architectural driver	Priority
1.	Usability	High
2.	Performance	Medium
3.	Accuracy of model simulation	Low

2.2. Technologies Used

In order to satisfy given architectural drivers, the technology should support developing application with good level of usability, it should graphical engine in order to draw world simulation and should have good performance characteristics. In order to satisfy business constrain the technology should be proven, and have examples of implementation of such application. Considering high priority of usability attribute and audience of the application, can be concluded that the application should be application on different devices: beginning from the PC, laptops, tablets, and ending with smartphones. Also support of modern devices will increase the student's interest for the application and topic. There are two approaches:



Types of Objects

Signal Class
Represents the state of a traffic light.

Fields:
`SignalState.Red`
`SignalState.Yellow`
`SignalState.Green`
`SignalState.Left`

The possible states of the traffic light.

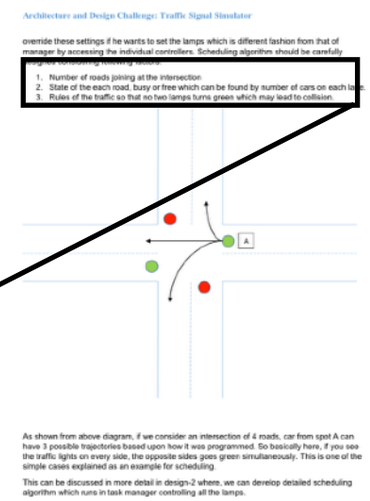
Figure 2. Valid state transitions for a Signal.

`int RedLength`
`int YellowLength`
`int GreenLength`
`int LeftLength`
The duration of the various states in ticks. If lefts are not permitted, `LeftLength` is zero.

`int TicksRemaining`
The number of ticks until the state changes.

Methods:
`Void Tick()`

Decrement `TicksRemaining`. If `TicksRemaining` is zero or less, change the state. Valid state changes (see Figure 2) are `Red` to `Left`, `Left` to `Green`, `Green` to `Yellow`, and `Yellow` to `Red`. If `LeftLength` is zero, `Red` to `Left` is instantaneous and followed by `Left` to `Green`, effectively making the state transition `Red` to `Green`. After the state transition, set `TicksRemaining` to the appropriate length.



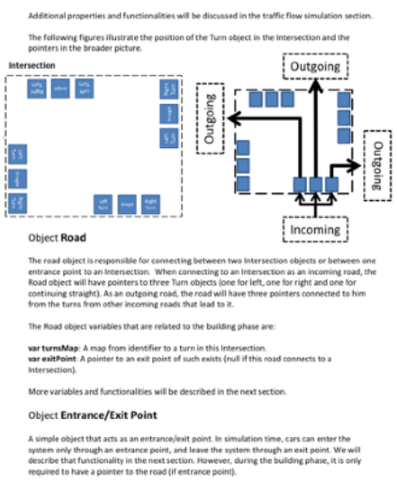
Architecture and Design Challenge: Traffic Signal Simulator

override these settings if he wants to set the lamps which is different fashion from that of manager by accessing the individual controllers. Scheduling algorithm should be carefully

1. Number of roads joining at the intersection
2. State of the each road, busy or free which can be found by number of cars on each lane
3. Rules of the traffic so that no two lamps turns green which may lead to collision.

As shown from above diagram, if we consider an intersection of 4 roads, car from spot A can have 3 possible trajectories based upon how it was programmed. So basically here, if you see the traffic lights on every side, the opposite sides goes green simultaneously. This is one of the simple cases explained as an example for scheduling.

This can be discussed in more detail in design-2 where, we can develop detailed scheduling algorithm which runs in task manager controlling all the lamps.



Additional properties and functionalities will be discussed in the traffic flow simulation section.

The following figures illustrate the position of the Turn object in the Intersection and the pointers in the broader picture.

Intersection

Object Road

The road object is responsible for connecting between two Intersection objects or between one entrance point to an Intersection. When connecting to an Intersection as an incoming road, the Road object will have pointers to three Turn objects (one for left, one for right and one for continuing straight). As an outgoing road, the road will have three pointers connected to him from the turns from other incoming roads that lead to it.

The Road object variables that are related to the building phase are:

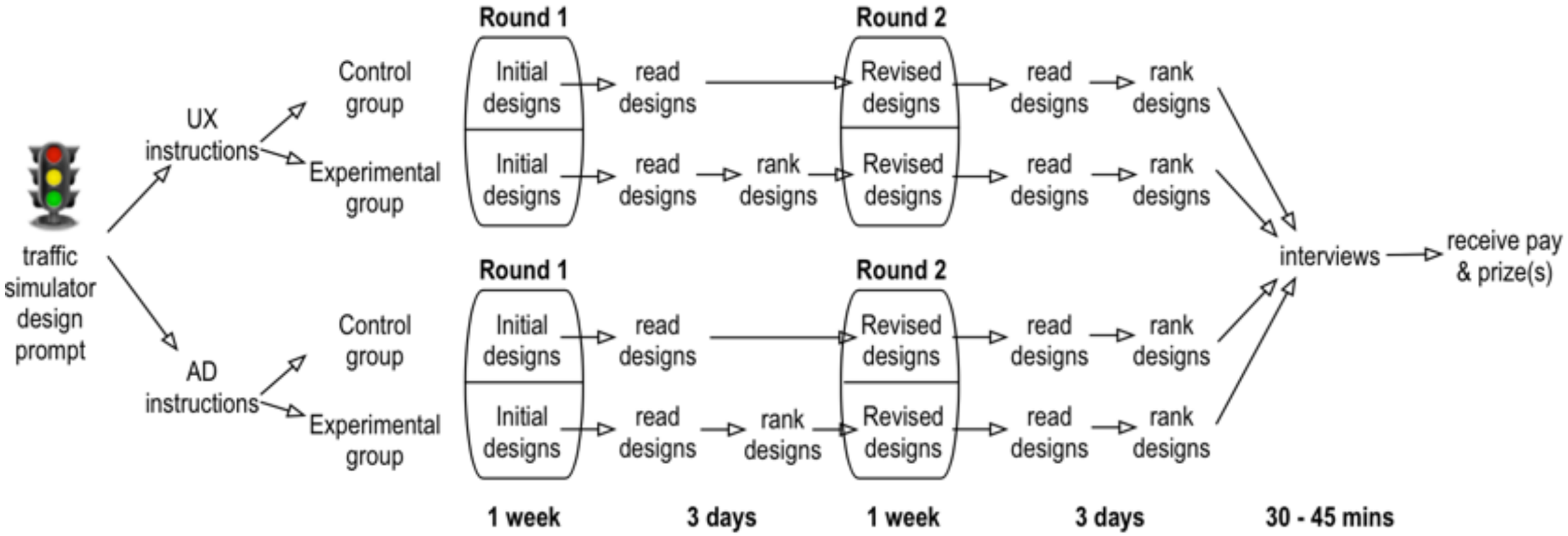
var turnsMap: A map from identifier to a turn in this Intersection.
var exitPoint: A pointer to an exit point of such exists (null if this road connects to a Intersection).

More variables and functionalities will be described in the next section.

Object Entrance/Exit Point

A simple object that acts as an entrance/exit point. In simulation time, cars can enter the system only through an entrance point, and leave the system through an exit point. We will describe that functionality in the next section. However, during the building phase, it is only required to have a pointer to the road (if entrance point).

UX and architectural design competitions



12 UX participants 10 AD participants

Designs independently **scored** on 1-7 scale by 4 expert panelists
Evaluated on elegance, clarity, and completeness

LaToza, Chen, Jiang, Zhao, van der Hoek. Borrowing from the crowd: A study of recombination in software design competitions. ICSE 2015.

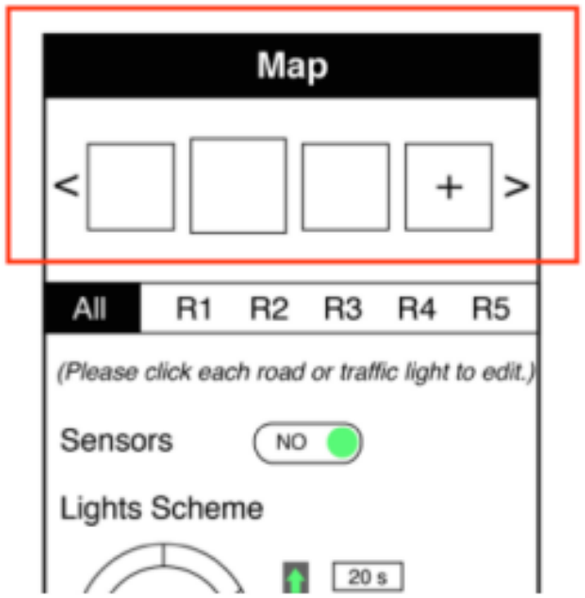
Designs increased in quality

User experience designs: **+1.8 points** out of 21 (p = .03)
75% of designs improved

Architectural designs: **+1.6 points** out of 21 (p = .009)
80% of designs improved

LaToza, Chen, Jiang, Zhao, van der Hoek. Borrowing from the crowd: A study of recombination in software design competitions. ICSE 2015.

Designers borrowed features and presentation elements



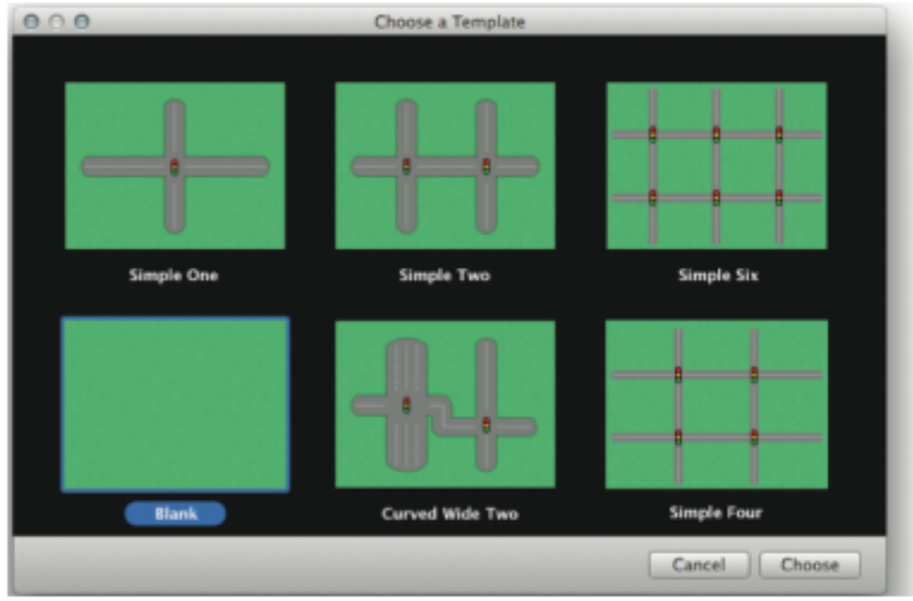
When user opens this simulator, there e is a default simulation provided.

User can just use the default map,

or choose other maps (different layout, intersections),

or create ("+") his own map (create a map based on his location or entered location).

(a) Pre-designed templates - UE4, Round 1



Available templates for simulation appear as thumbnail views of their maps.

User selects template.

(b) Pre-designed templates - UE2, Round 2

LoopSensor

A LoopSensor detects Vehicles as they drove over top of it. They are primarily used for intelligent traffic flow control in "actuated" Intersections, though they can also be used to capture Statistics about Vehicle flow through particular stretches of Road. LoopSensors detect Vehicles by receiving updates about nearby Vehicle Locations from the Roads these two objects are attached to. As previously mentioned, the MultiIndex structure used to store Vehicles and LoopSensors in Roads allows querying for nearby neighbors. Every time a Vehicle updates its Location within a Road, the Road will query for the nearest LoopSensor in that Vehicle's Lane and alert the LoopSensor to the Vehicles presence. If the Vehicle is close enough to be detected, the LoopSensor will notify its associated TrafficController (if any) and store this information in the Statistics store. As the Vehicle moves away, the Road will again notify the LoopSensor, which will then determine that the Vehicle is gone and as such will notify the TrafficController.

Source design

4.3 SENSORS

Sensors can be attached to edges to detect cars presence. When they detect that a car is passing, they notify to a specific traffic light. The stimulus will be considered by the scheme to determine which traffic light combination will be activated next.

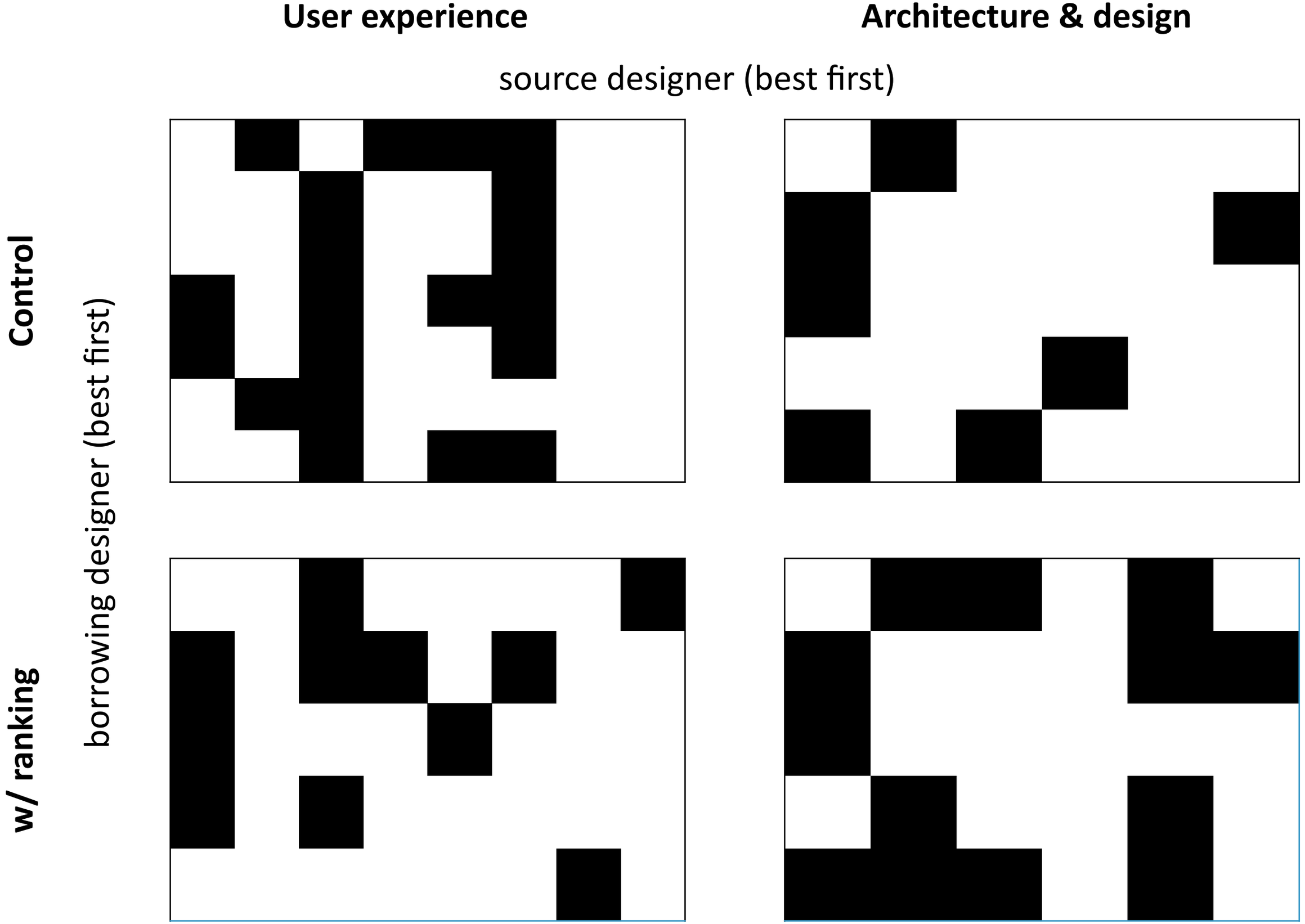
Sensors will be notified when the simulation starts by calling its *start()* method and they will schedule themselves in the VirtualClock to check for cars. For example, if a sensor must check every 50ms whether there are cars or not, they can schedule a Handler to be executed after 50ms and that handler will be perform the checking and will also schedule the next check.

Since sensors have a position in the edge and cars can also be asked for their positions, the sensors can determine whether a car is close enough to be sensed. An efficient indexing must be implemented to relate cars and edges to avoid checking for all the cars in the map.

Revised design

LaToza, Chen, Jiang, Zhao, van der Hoek. Borrowing from the crowd: A study of recombination in software design competitions. ICSE 2015.

All designers borrowed



LaToza, Chen, Jiang, Zhao, van der Hoek. Borrowing from the crowd: A study of recombination in software design competitions. ICSE 2015.

Microtasking software design work

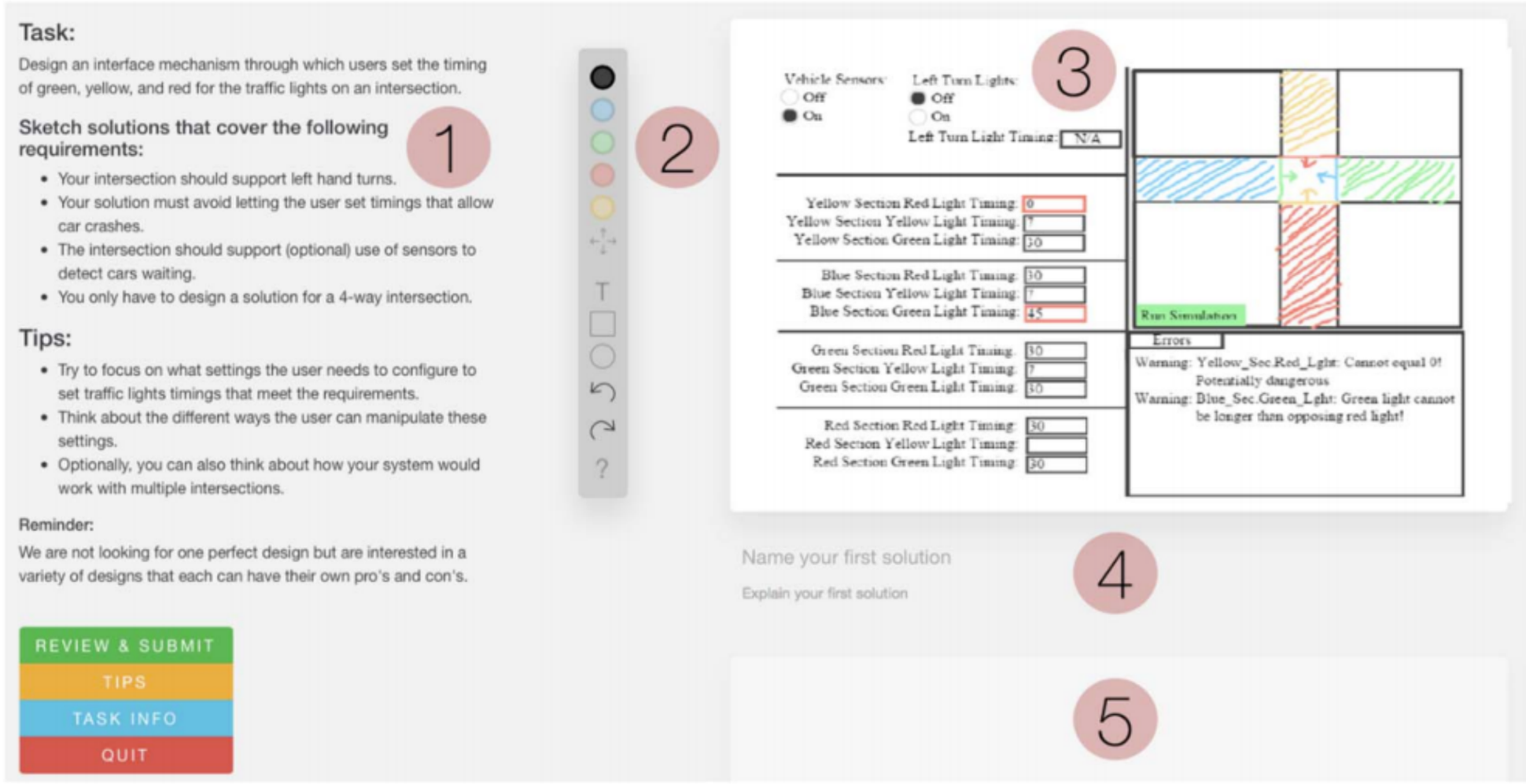


Figure 1. CrowdDesign Platform: (1) sample task description, (2) tool bar for sketching, (3) sketch area, (4) input area for name of the solution alternative and its explanation, and (5) additional canvases, which can be reached by scrolling down.

Edgar R. Q. Weidema, Consuelo López, Sahand Nayebaziz, Fernando Spanghero, and André van der Hoek. 2016. Toward microtask crowdsourcing software design work. *International Workshop on CrowdSourcing in Software Engineering* (CSI-SE '16), 41-44.

Apparition

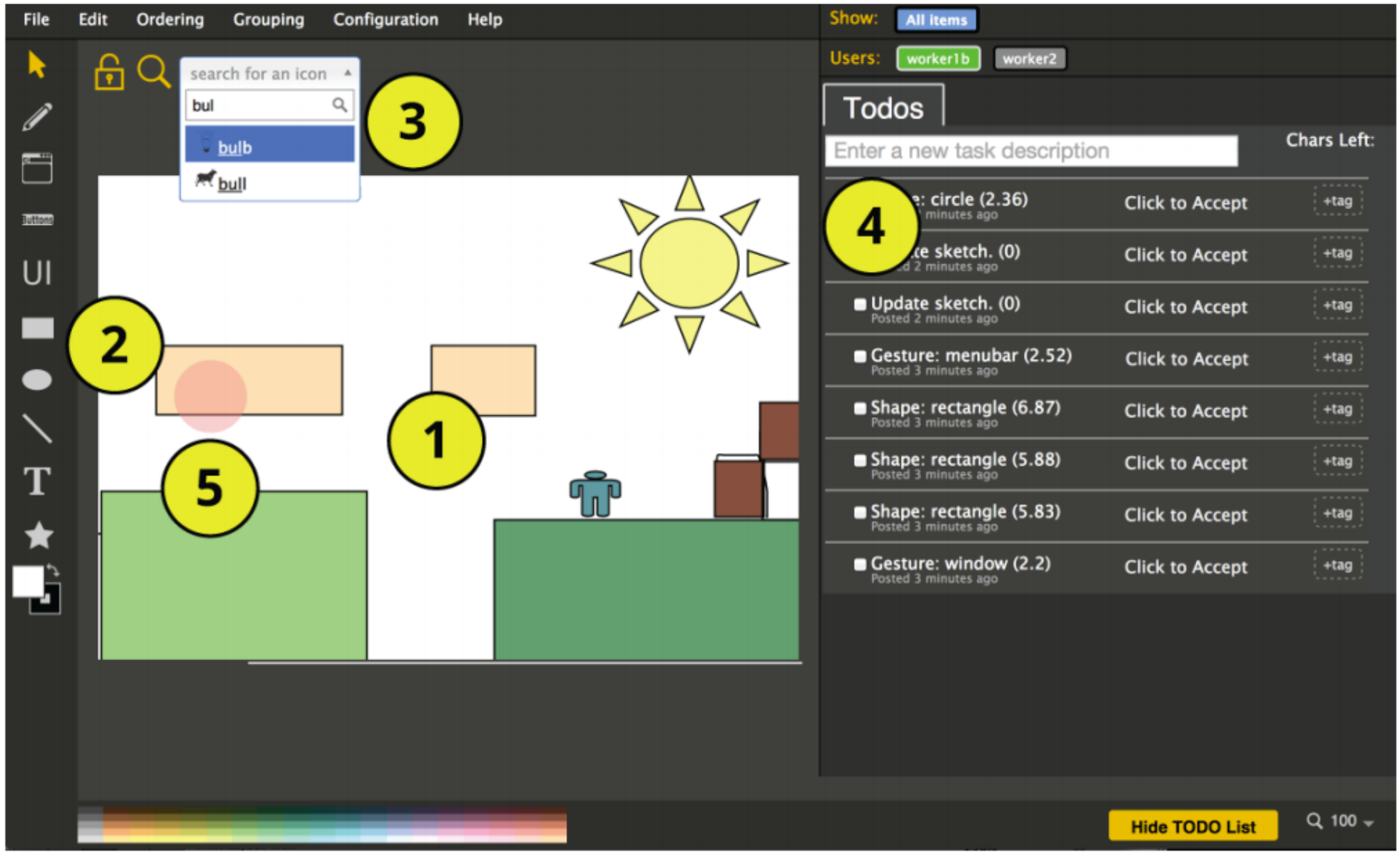


Figure 1. Apparition allows designers to quickly create functional interface prototypes using sketches and verbal descriptions. (1) A collaborative canvas where the user(s) and workers can draw. (2) Drawing tools and icons to quicken workers' creation of UI elements. (3) A search function to help workers find relevant icons. (4) A to-do list that shows what has been drawn by the user but not yet converted to UI elements. Workers can "accept" tasks to signal what they are currently working on. (5) "In-progress" markers for workers to show where they are currently working to avoid conflicts.

https://www.youtube.com/watch?v=tBCB6P7FwWY

Walter S. Lasecki, Juho Kim, Nick Rafter, Onkur Sen, Jeffrey P. Bigham, and Michael S. Bernstein. 2015. Apparition: Crowdsourced User Interfaces that Come to Life as You Sketch Them. *CHI*, 1925-1934.

Flash teams

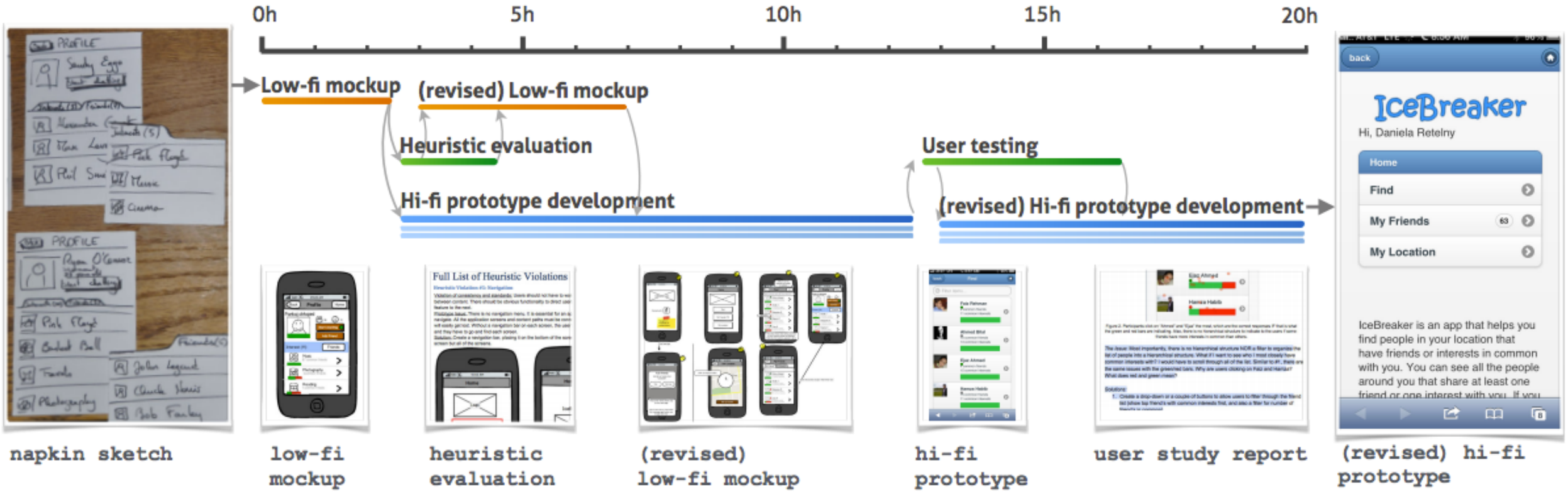


Figure 1: A flash team is a linked set of modular tasks that draw upon paid experts from the crowd, often three to six at a time, on demand. The napkin sketch design team follows the user-centered design process to create a series of prototypes and iterate based on feedback to produce a user-tested software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

https://www.youtube.com/watch?v=IVgTZEpHOzc

Daniela Retelny, Sebastien Robaszkiewicz, Alexandra To, Walter Lasecki, Jay Patel, Negar Rahmati, Tulsee Doshi, Melissa Valentine, Michael Bernstein. Expert Crowdsourcing with Flash Teams. UIST 2014: ACM Symposium on User Interface Software and Technology.