Crowdsourcing
Software Engineering

SWE 795, Fall 2019
Software Engineering Environments
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 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.
```javascript
function myAbs(value) {
    if (value < 0) {
        return -value;
    }
    return value;
}
```
Games with a Purpose (Pipejam)
PipeJam

Figure 2: Detail of two boards from the Pipe Jam game shown in Figure 1. The left board demonstrates wide and narrow pipes, pinch points, a merge (near the bottom center), and a subnetwork. The right board demonstrates two collisions, each of which prevents the game from being solved. At the top, a large ball collides with a pinch point. At the bottom, a wide ball gets stuck trying to merge into a narrow pipe. The collisions are highlighted by red circles. A pipe segment is outlined in green if it contains no collision, and in red if it contains a collision. The gray pipe on the right board cannot be adjusted in width.
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

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

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

Whole project (OSS)

Derivative projects (OSS)

Work items (Competitions)

Design decisions (Q&A sites)
Task length & Task context

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

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Open source</th>
<th>TopCoder</th>
<th>UserTesting.com</th>
</tr>
</thead>
<tbody>
<tr>
<td>crowd size</td>
<td>small – medium</td>
<td>small</td>
<td>medium</td>
</tr>
<tr>
<td>task length</td>
<td>hours – days</td>
<td>days - week</td>
<td>minutes</td>
</tr>
<tr>
<td>expertise demands</td>
<td>moderate</td>
<td>extensive</td>
<td>minimal</td>
</tr>
<tr>
<td>locus of control</td>
<td>workers</td>
<td>client</td>
<td>client</td>
</tr>
<tr>
<td>incentives</td>
<td>intrinsic</td>
<td>extrinsic</td>
<td>extrinsic</td>
</tr>
<tr>
<td>task interdependence</td>
<td>moderate</td>
<td>low</td>
<td>low</td>
</tr>
<tr>
<td>task context</td>
<td>extensive</td>
<td>minimal</td>
<td>none</td>
</tr>
<tr>
<td>replication</td>
<td>none</td>
<td>several</td>
<td>many</td>
</tr>
</tbody>
</table>
Programming is becoming more social and fluid

- **Stack Overflow**
  - Share hard-earned expertise
  - >7M monthly visits

- **Codecademy**
  - Learn programming through structured examples
  - >24M users

- **GitHub**
  - Reputation made visible
  - >8M users

- **Code Hunt**
  - Competitive programming duels
  - >100K users
Crowdsourcing SE opportunities

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
- Crowdsourcing systems for 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

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

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

Coverage of APIs

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

<table>
<thead>
<tr>
<th>SEARCH RESULT TYPE</th>
<th>COVERAGE</th>
<th>MEAN RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>code snippet site</td>
<td>8.7%</td>
<td>9</td>
</tr>
<tr>
<td>q&amp;a</td>
<td>9.8%</td>
<td>9</td>
</tr>
<tr>
<td>forum</td>
<td>20.2%</td>
<td>8</td>
</tr>
<tr>
<td>official bug tracker</td>
<td>21.4%</td>
<td>3</td>
</tr>
<tr>
<td>mailing list entry</td>
<td>25.4%</td>
<td>7</td>
</tr>
<tr>
<td>official documentation</td>
<td>30.1%</td>
<td>3</td>
</tr>
<tr>
<td>official forum</td>
<td>37.0%</td>
<td>3</td>
</tr>
<tr>
<td>unofficial documentation</td>
<td>63.6%</td>
<td>6</td>
</tr>
<tr>
<td>stackoverflow</td>
<td>84.4%</td>
<td>6</td>
</tr>
<tr>
<td>blog post</td>
<td><strong>87.9%</strong></td>
<td><strong>5</strong></td>
</tr>
<tr>
<td>official API</td>
<td>99.4%</td>
<td>1</td>
</tr>
</tbody>
</table>

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

![Diagram showing setup for hypothetical assistant study]

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.

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

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.

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.

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

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.


https://www.youtube.com/watch?v=jAIWXbygKuc
Increasing parallelism in programming

Build a large application in a day?

\[ >24 \text{ million users} \times 1 \text{ day} = ??? \]
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

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

- 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

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:

```javascript
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

**/

function renderElement(element) {
    if (element.type == "Line") {
        // render line
    } else if (element.type == "Rectangle") {
        // render rectangle
    } else if (element.type == "Freehand") {
        // render freehand
    }
    return {};
}
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).
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**
create a FreeHand

**FUNCTION SIGNATURE**

```javascript
/**
 * 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)`
- `y (Number)`
- `action (Action)`
- `prevElement (Element)`

**RETURN VALUE**
- `(Element)`
Defined by the Action and using the

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

position of Mouse - left is 0
position of Mouse - top is 0

tdent action
ervious version of the Element - may be null

x (Number) | 10

y (Number) | 10

action (Action) | 

"type": "Freehand",
"elementID": 0,
"mouseDownX": 10,
"mouseDownY": 10

prevElement (Element) | 
x = null

Unexpected end of input

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

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

- described
- written
- !buggy
Edit a Function

!described

described
!written

testRun

described
written
buggy

described
written
!buggy
Debug a Test Failure

- !described
- described
- written
- !written
- testRun
- described
- written
- buggy
- described
- written
- !buggy
Edit a Function

- !described
- described
- !written
- testRun
- described
- written
- buggy
- described
- written
- !buggy
Coordinating work

- One microtask in progress per artifact
Edit a Function

function f

function add(Number[] numbers)

Number a, Number b

microtask_1

test_1
test_2
test_3

test_4
test_5

microtask_2

microtask_3
function f → function add(Number[] numbers)

microtask1

microtask6

microtask2

test1 test2 test3

microtask4

test4 test5

[55]
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

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)

<table>
<thead>
<tr>
<th>Microtask type</th>
<th>Completed</th>
<th>Skipped</th>
<th>Reissued</th>
<th>Median time (m:s)</th>
<th>Total time (h:m:s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>Session</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write test</td>
<td>158</td>
<td>102</td>
<td>22</td>
<td>7</td>
<td>1:29</td>
</tr>
<tr>
<td>Write function</td>
<td>44</td>
<td>56</td>
<td>25</td>
<td>21</td>
<td>4:57</td>
</tr>
<tr>
<td>Write test cases</td>
<td>40</td>
<td>30</td>
<td>9</td>
<td>4</td>
<td>3:50</td>
</tr>
<tr>
<td>Debug test failure</td>
<td>14</td>
<td>18</td>
<td>5</td>
<td>6</td>
<td>2:32</td>
</tr>
<tr>
<td>Write function</td>
<td>8</td>
<td>16</td>
<td>3</td>
<td>0</td>
<td>3:12</td>
</tr>
<tr>
<td>Write call</td>
<td>7</td>
<td>9</td>
<td>2</td>
<td>2</td>
<td>1:37</td>
</tr>
<tr>
<td>Reuse search</td>
<td>9</td>
<td>10</td>
<td>3</td>
<td>0</td>
<td>0:42</td>
</tr>
<tr>
<td>Total</td>
<td>540</td>
<td>468</td>
<td>91</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>Overall total</td>
<td>1008</td>
<td>153</td>
<td>16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Results - Small contributions
Results - Fixing bugs
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

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?
Figure 1: Mercury allows programmers to continue their work on-the-go. (1) When a user leaves their workstation, Mercury generates microtasks from their code, (2) then serves the tasks to their mobile device. These tasks are brief and require little attention. (3) Finally, Mercury integrates the user’s microtask responses into the their workstation’s source files.

https://www.youtube.com/watch?v=Urh-TQk4ebQ&feature=youtu.be
Software design recombination

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

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.
All designers borrowed

User experience

Architecture & design

source designer (best first)

Control

borrowing designer (best first)

w/ ranking

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

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.

https://www.youtube.com/watch?time_continue=34&v=A_Pngz1mbDs&feature=emb_logo
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,