

# Rethinking Programming “Environment”

Technical and Social Environment Design toward Convivial Computing

Convivial Computing Salon, May 5, 4-5pm London (Zoom)

**Jun Kato<sup>\*1</sup>, Keisuke Shimakage<sup>\*2</sup>**

<sup>\*1</sup> National Institute of Advanced Industrial Science and Technology (AIST)

<sup>\*2</sup> OTON GLASS, Inc.



# Rethinking programming “environment”

## BACKGROUND

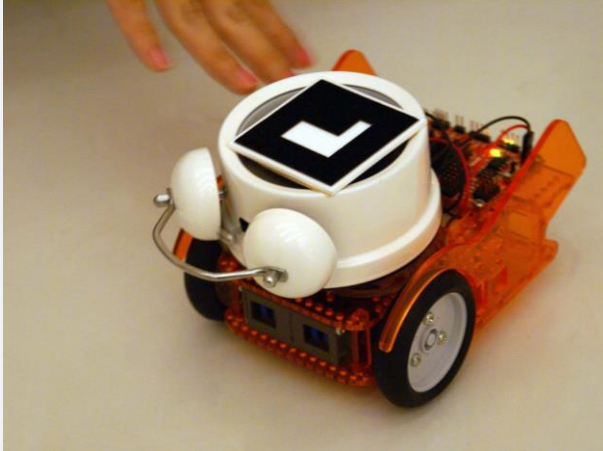
Programming environment design for development of programs that run in the real world

## TOWARD CONVIVIAL COMPUTING

- Technical environment design for collaborations
- Social environment design for more inclusion

## CONCLUSION

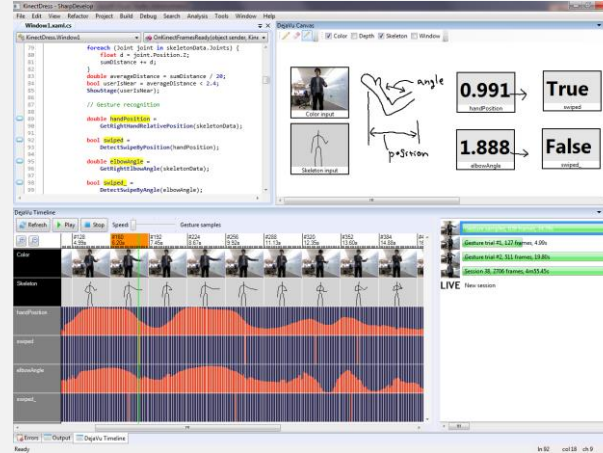
# A bit of self introduction...



## Phybots

A toolkit (API and runtime debugger)  
for making “robotic things”

ACM DIS 2012



## DeJaVu

IDE extensions for developing  
Interactive camera-based programs

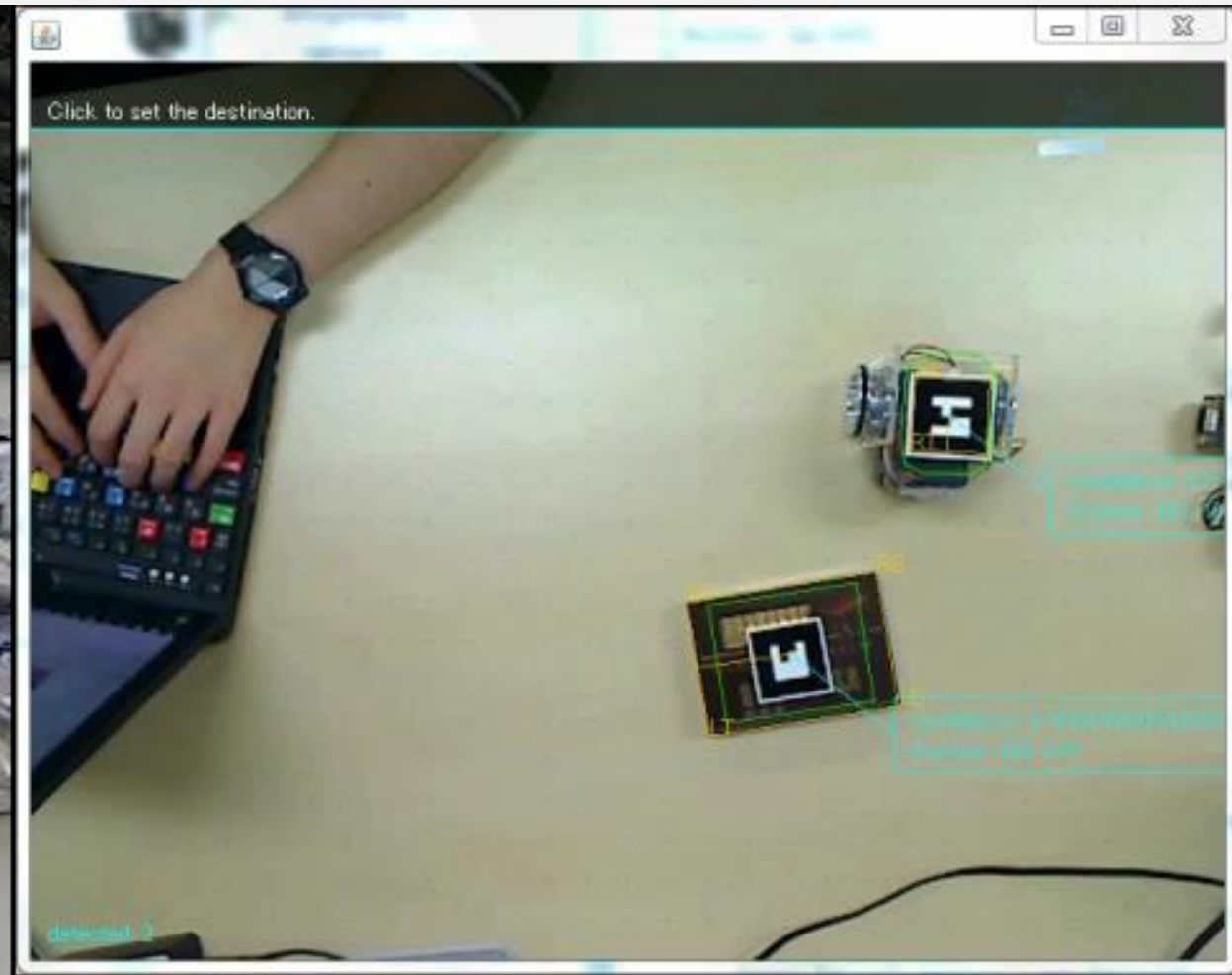
ACM UIST 2012



## Songle Sync

APIs for controlling various devices  
in synchronization with music

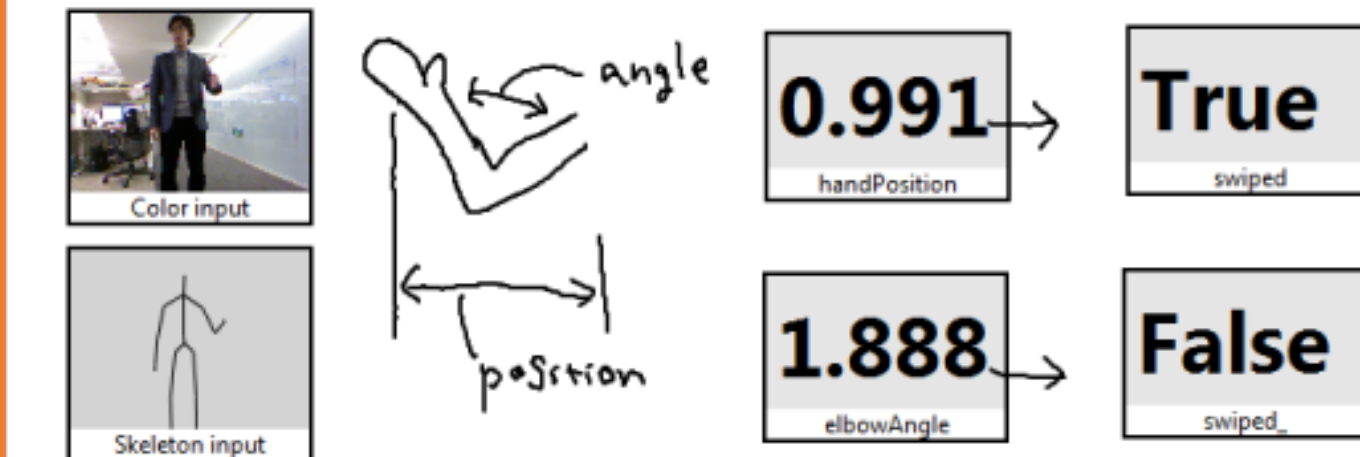
ACM Multimedia 2018



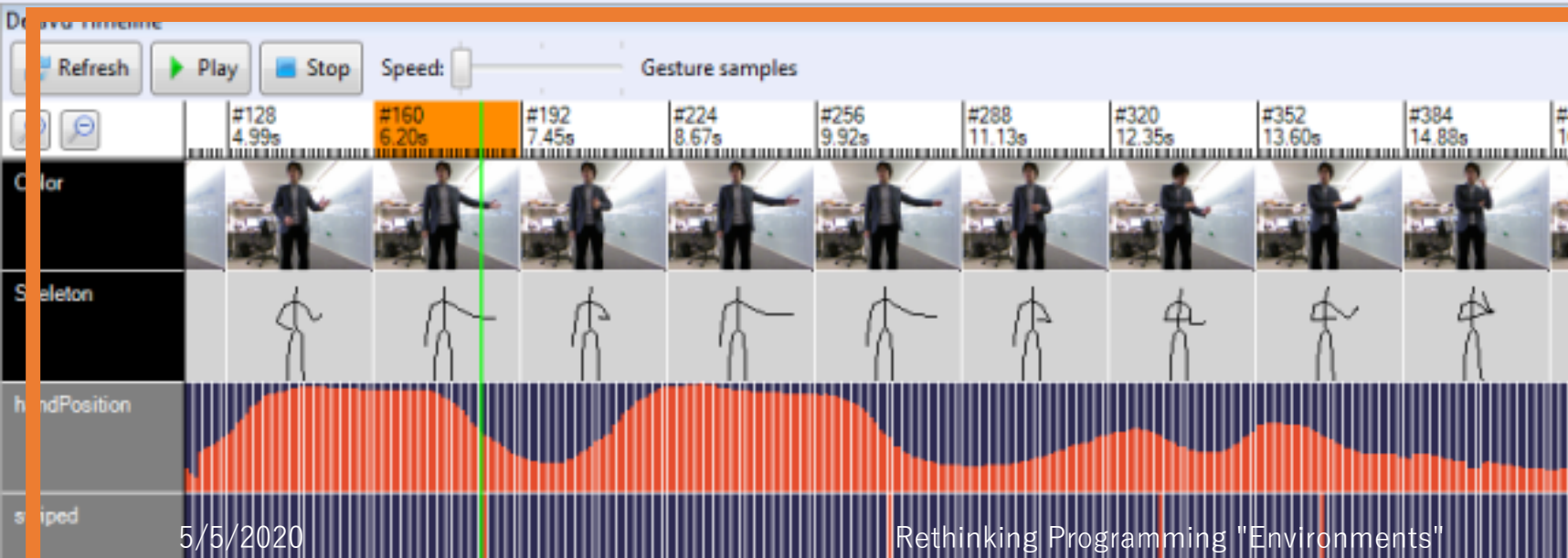
```

80     float d = joint.Position.Z;
81     sumDistance += d;
82 }
83 double averageDistance = sumDistance / 20;
84 bool userIsNear = averageDistance < 2.4;
85 ShowStage(userIsNear);
86
87 // Gesture recognition
88
89 double handPosition =
90     GetRightHandRelativePosition(skeletonData);
91
92 bool swiped =
93     DetectSwipeByPosition(handPosition);
94
95 double elbowAngle =
96     GetRightElbowAngle(skeletonData);
97
98 bool swiped_ =
99     DetectSwipeByAngle(elbowAngle);

```



Canvas



Timeline

Gesture samples, 639 frames, 24.79s

Gesture trial #1, 127 frames, 4.99s


Gesture trial #2, 511 frames, 19.80s

Session 38, 2706 frames, 4m55.45s

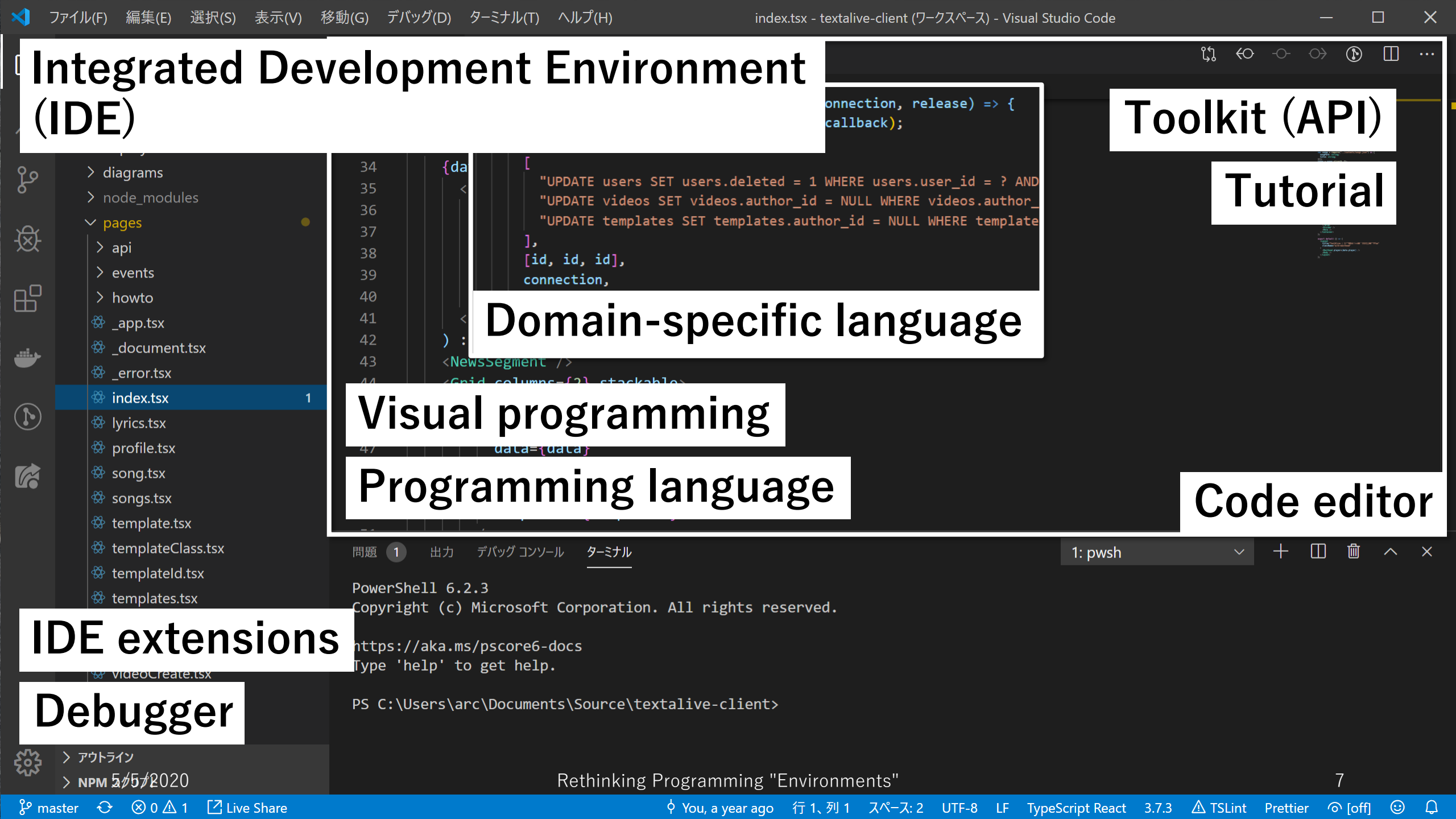
LIVE New session



<https://api.songle.jp>



In our demonstration experiment, over 110 devices were connected to the Songle Sync platform.



# Integrated Development Environment (IDE)

Toolkit (API)

Tutorial

Domain-specific language

Visual programming

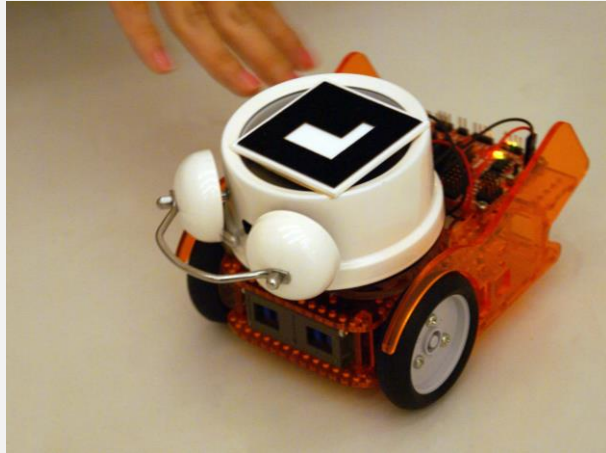
Programming language

Code editor

IDE extensions

Debugger

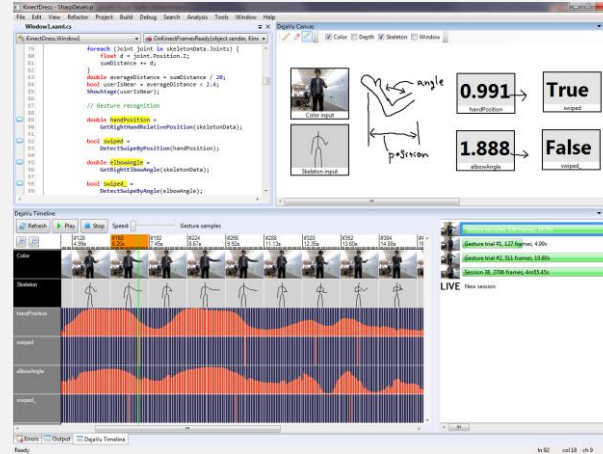
# A bit of self introduction...



## Phybots

A toolkit (API and runtime debugger)  
for making “robotic things”

ACM DIS 2012



## DeJaVu

IDE extensions for developing  
interactive camera-based programs

ACM UIST 2012



## Songle Sync

APIs for controlling various devices  
in synchronization with music

ACM Multimedia 2018

# PX for programs running in the real-world





# f3.js: A Parametric Design Tool for Physical Computing Devices for Both Interaction Designers and End-users

**f3.js** Edit the project Basic Information Source Code Design Alternatives

**</> Source Code**

Provide the source code of a microcontroller or tiny computer in JavaScript. Node.js-based computers are supported. Require f3.js package and use its API to design the device enclosure.

```

1  /** LED blinking app for Intel Edison */
2
3  var n = 2 // number of LEDs [1,4]
4    , width = 130
5    , height = 105
6    , thickness = 45;
7
8  // for building the enclosure layout
9  var f3js = require('f3js')
10   , c = f3js.createContainer()
11   , rect = c.drawJointRectangle(0, 0, width, height)
12   , line = c.drawLine(50, height/2 - 5, width - 50, height/2 - 5);
13
14  // for blinking LEDs
15  var groveDriver = require('jsupm_grove')
16    , leds = [];
17
18  // use this line as the guide path
19  line.layout = { name: 'distribute', rotate: false };
20
21  for (var i = 0; i < n; i++) {
22    var led = new groveDriver.GroveLed(i+2);
23
24    // put an LED module
25    var ledc = c.add(led, line);
26
27    // open a hole for the wire
28    ledc.drawRectangle(- 10, 15, 20, 10);
29
30    // start blinking the LED (details omitted)
31    leds.push(led);
32    setTimeout(function (l) { return function () {
33      l.handler = setInterval(blink(l), 1000);
34    } }(led), i * 100);
  
```

**Layout**

Selected: Rectangle  
Hovered: -

**Customization**

number of LEDs (2)

**Layout view options**

Update Delete

© f3.js Project 2016, 2017 - 日本語

**f3.js** Get Started IoT Projects Modules

**Preview** Page: 1

**Customize**

number of LEDs (2)

**Layout view options**

- Show module names
- Show labels near adjacent edges
- Show warnings on module interferences

Disabled items are those proposed by somebody with the "🔒" button and not yet implemented. Got interested?

**Building Instructions**

- Purchase**  
Purchase hardware components
- Print & Assemble**  
Print out enclosure components
- Install**  
Install and run the program

**Modules required for assembly**

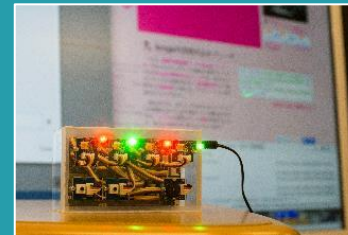
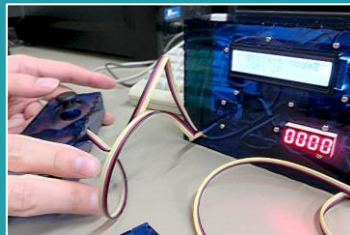
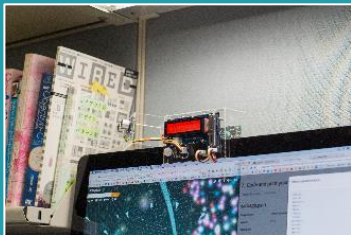
2 Grove LED

**Layout**

Printer type

**Program**

Archive type



<https://f3js.org>

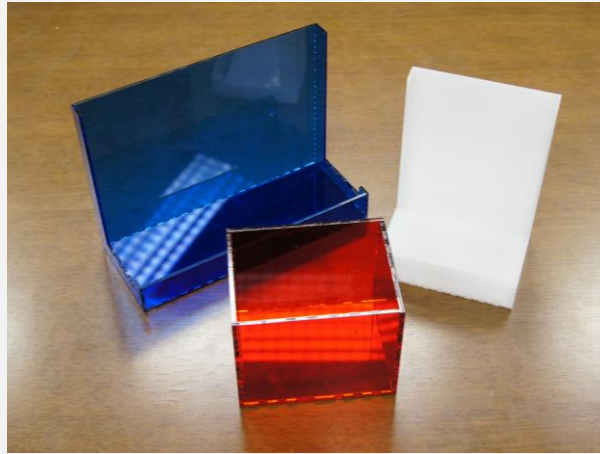
Jun Kato, Masataka Goto

# Personal fabrication made easy

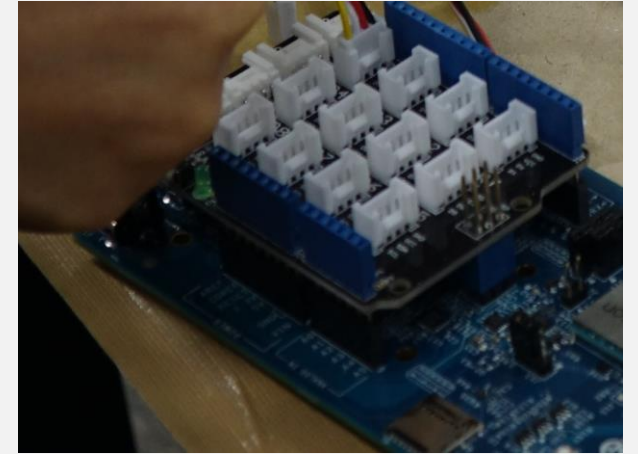


3D printers

Photo taken by Atsushi Tadokoro (CC BY 2.0)  
<https://www.flickr.com/photos/tadokoro/5138646645>



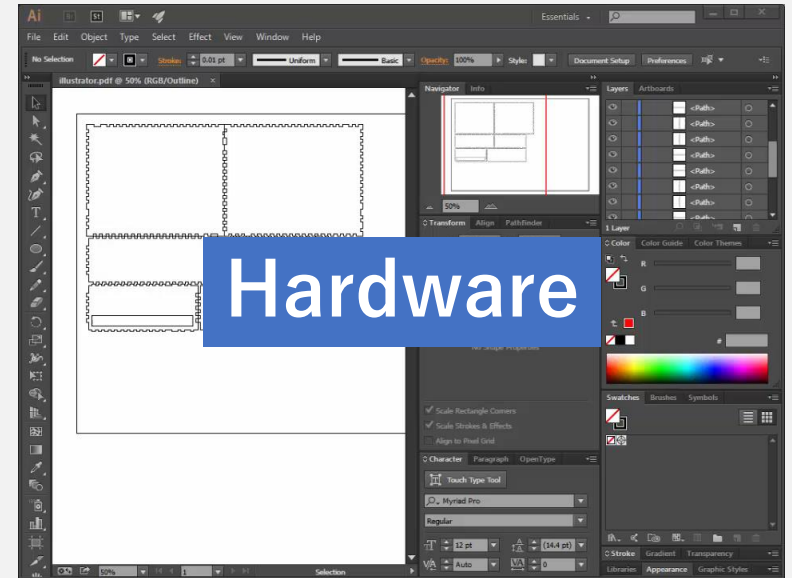
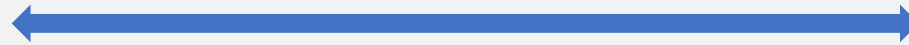
Laser cutters



Sensor and  
actuator modules

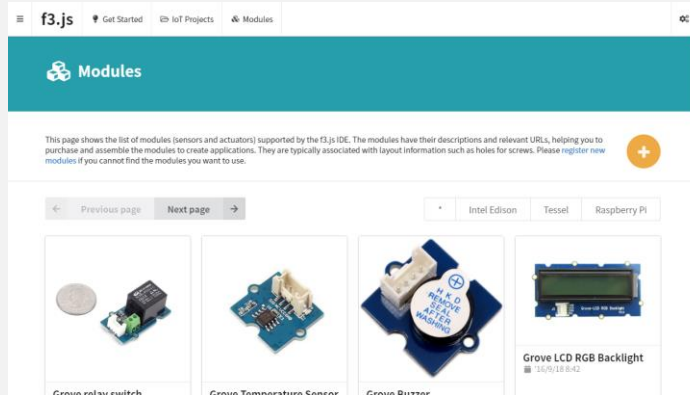
## How about programming and device assembling?

# Different tools and expertise needed for software and hardware design

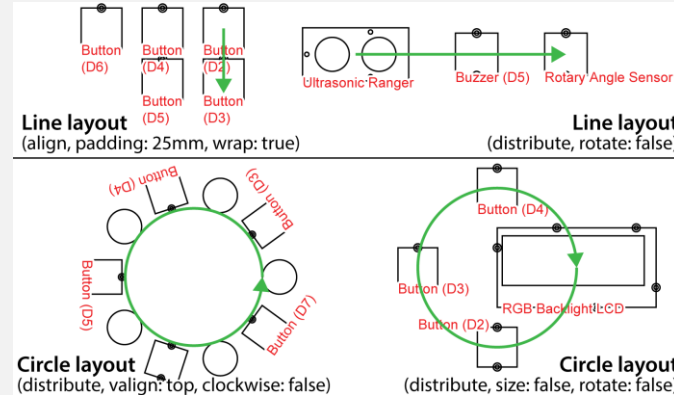


- Programmers need to imagine hardware while writing code
- “new Button()” does not infer any hardware layout

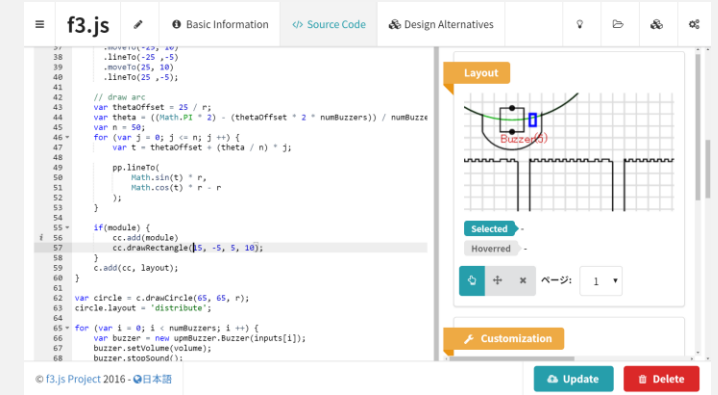
# f3.js



Module repository for hardware metrics



APIs for enclosure layout



Live programming editor for microcontroller firmware and enclosure layout

# IDE for creating laser-cut interfaces and microcontroller programs from single code base



## Source Code

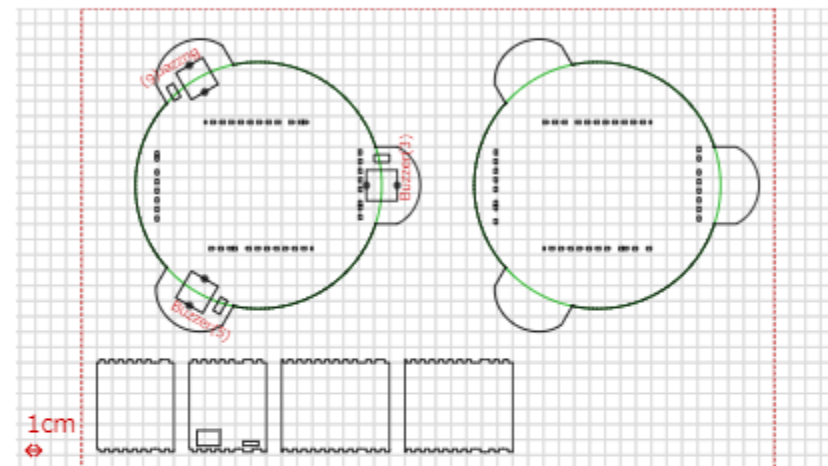
Provide the source code of a microcontroller or tiny computer in JavaScript. Node.js-based computers are supported. Require f3.js package and use its API to design the device enclosure.

```

1 var WebSocket = require('ws');
2 var serverAddr = 'ws://192.168.10.100:8080/entry'; // Server URL
3 var ws = new WebSocket(serverAddr);
4 var upmBuzzer = require('jsupm_buzzer');
5
6 var numBuzzers = 3; // Number of buzzers [1,5]
7 var volume = 57; // Volume [0,100]
8
9 var inputs = [3, 5, 6, 9];
10 var buzzers = [];
11 var r = 80;
12
13 var f3js = require('f3js');
14 var c = f3js.createContainer();
15 c.x = 50;
16 c.y = 50;
17
18 var c1 = 43
19   , c2 = 36
20   , x = 70
21   , y = 140
22   , width = 70;
23 var a = c.createPath();
24
25 var ps = a.extrude(60);
26 ps[0].y = 50;
27 ps[0].x = 270;
28 f3js.add(ps[0]);
29
30 var dx = [ 65, 145, 65, -15]
31   , dy = [-15, 65, 145, 65];
32
33 function addPetal(layout, module) {
34   var cc = f3js.createContainer();

```

## Layout



Selected

Hoverred



## Customization

Server URL

ws://192.168.10.100:8080/entry

Number of buzzers (3)

Update

Delete



Integrated Development Environment (IDE)

Toolkit (API)

Tutorial

Programming environments  
are designed for programmers

IDE extensions

Debugger

# Rethinking programming “environment”

## BACKGROUND

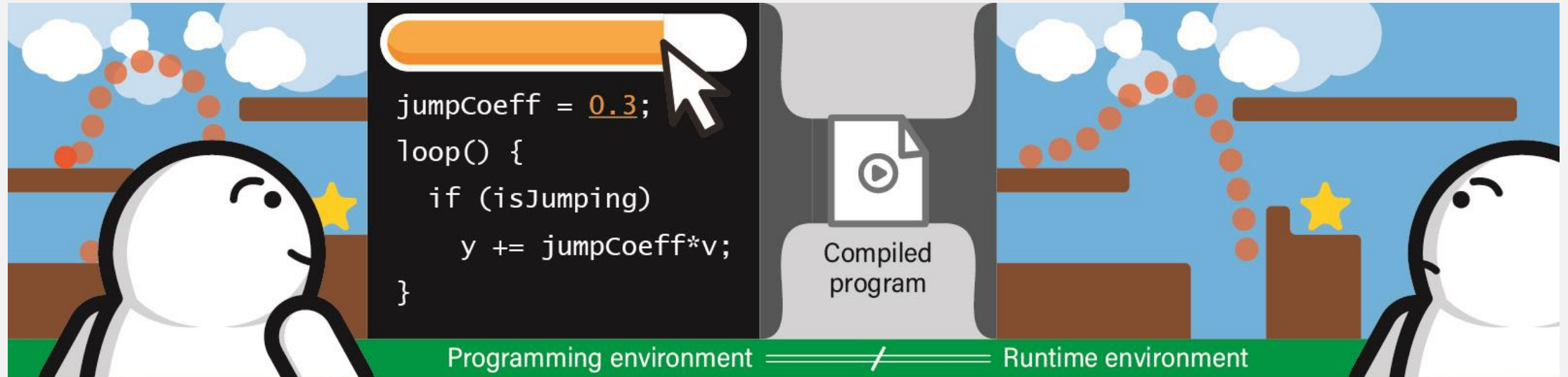
Programming experience design for development of programs that run in the real world

## TOWARD CONVIVIAL COMPUTING

- Technical environment design for collaborations
- Social environment design for more inclusion

## CONCLUSION

# Programming environments are (usually) different from runtime environments



- **Programmers** develop programs and publish them
- **Users** install “apps” and use them
- Once published, **the programs cannot be edited**



# Some environments allow “remix”

Scratch – Imagine, Program, Share. <https://scratch.mit.edu/>

The screenshot shows the Scratch project page for "Scratch Month!" by ScratchCat. The project features a large blue banner with the Scratch cat and the text "SCRATCH MONTH". To the right of the banner are sections for "Instructions" and "Notes and Credits". Below the banner are statistics: 2793 likes, 2254 stars, 3309 comments, and 37757 views. At the bottom, there is a "Comments" section and a "Remixes" section. Two callout boxes are present: one pointing to a "See inside" button in the top right corner, and another pointing to a "Remixes" button in the bottom right corner. The "See inside" button is labeled "“See inside” to read and edit source code". The "Remixes" button is labeled "“Remixes” to play with edited programs".

Scratch Month!  
by ScratchCat

See inside

Instructions

This year, to celebrate Scratch's birthday, we are going beyond Scratch Day and will be celebrating throughout the entire month of May! We will be sharing new activities you can participate in each week.

The first activities start next Monday, May 4th, 2020 and will be featured on the home page in the "studios" row

Notes and Credits

Press space or tap the screen, if on mobile, to move through the project.

Remix this project to add yourself dancing with me in anticipation of Scratch Month! The first activities start next Monday, May 4th, 2020! We hope you'll join us in celebrating :D

2793 2254 3309 37757

Apr 28, 2020 Copy Link

Remixes View all

SCRATCH MONTH

Comments

nista11

“See inside”  
to read and edit source code

“Remixes”  
to play with edited programs

Some environments allow “remix”

Still, **programming environments**  
are designed for **programmers**

2793 2254 3309 37757

Apr 28, 2020

Copy Link

Comments

nista11

Remixes

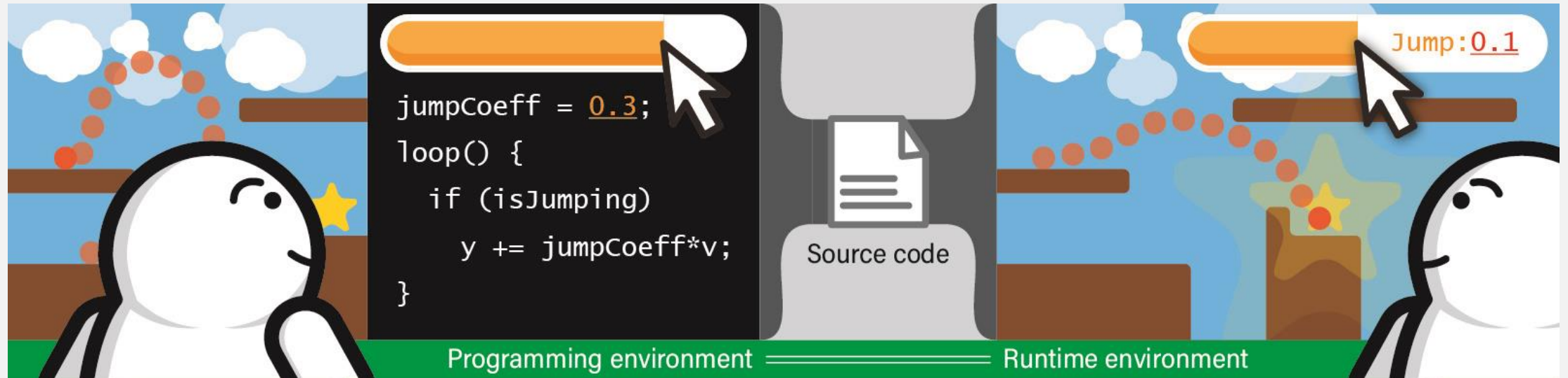
View all

SCRATCH MONTH

“Remixes”

to play with edited programs

# What if we design it for “both”

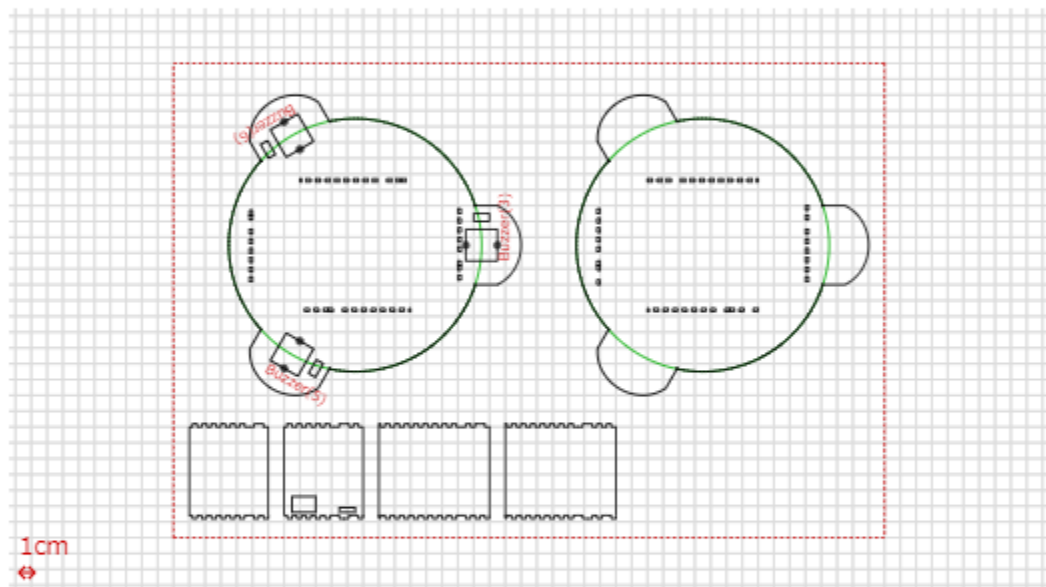


- Instead of compiled programs, source code is shared
- Furthermore, users benefit from a respective UI design
- Users can **safely edit the programs without breaking them**

## Enclosure layout

### Preview

Page: 1 ▼



### Customize

#### Server URL

#### Number of buzzers (3)



#### Volume (57)



### Layout view options

- ☒ Show module names
- ☐ Show labels near adjacent edges
- ☐ Show warnings on module interferences

## Live Tuning:

Users can get a customized variation of the device





This page shows

**Basic Information**

Yet another m

**Authors**

**Updated at**



### Propose More Customization

Customization type \*

Number



Customization label \*

e.g. Object height [mm]

Minimum value \*

30

Maximum value \*

120

## User-Generated Variables:

Users can propose parameters for the device spec

# A multi-layered UI design approach

- Base layer for programmers
- Another layer for users

Shneiderman. Promoting universal usability with multi-layer interface design. 2002.

# Meta-design framework

- Programmers as meta-designers
- Users as designers
- Programming environments become socio-technical systems

Fischer et al. Revisiting and broadening the meta-design framework for end-user development. 2017.

# Rethinking programming “environment”

## BACKGROUND

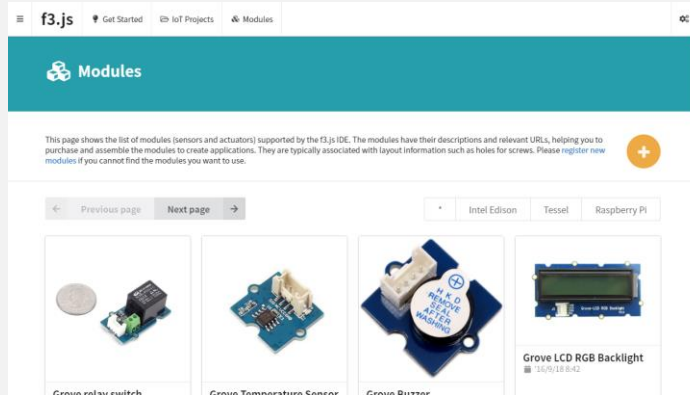
Programming experience design for development of programs that run in the real world

## TOWARD CONVIVIAL COMPUTING

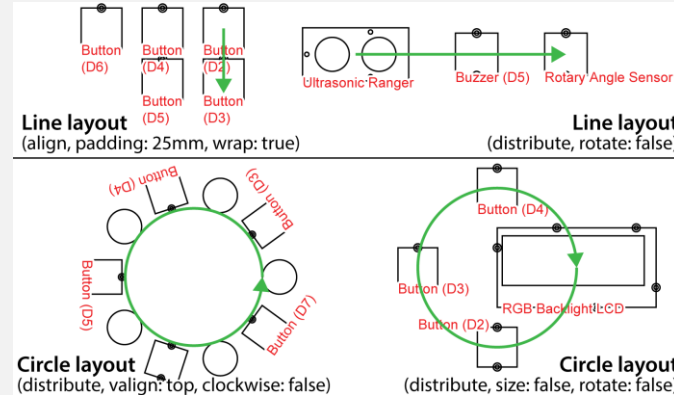
- Technical environment design for collaborations
- Social environment design for more inclusion

## CONCLUSION

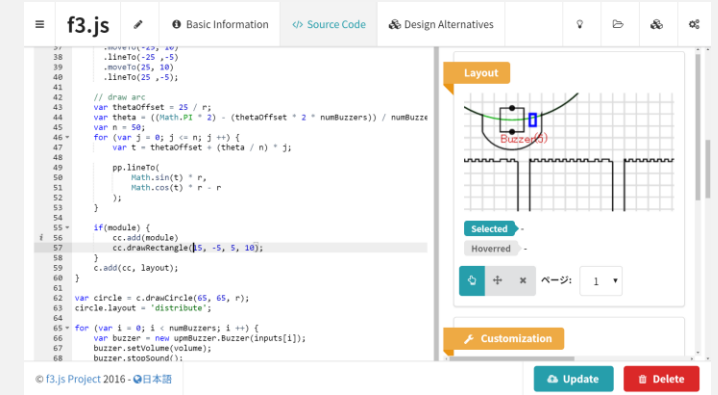
# f3.js



Module repository for hardware metrics



APIs for enclosure layout

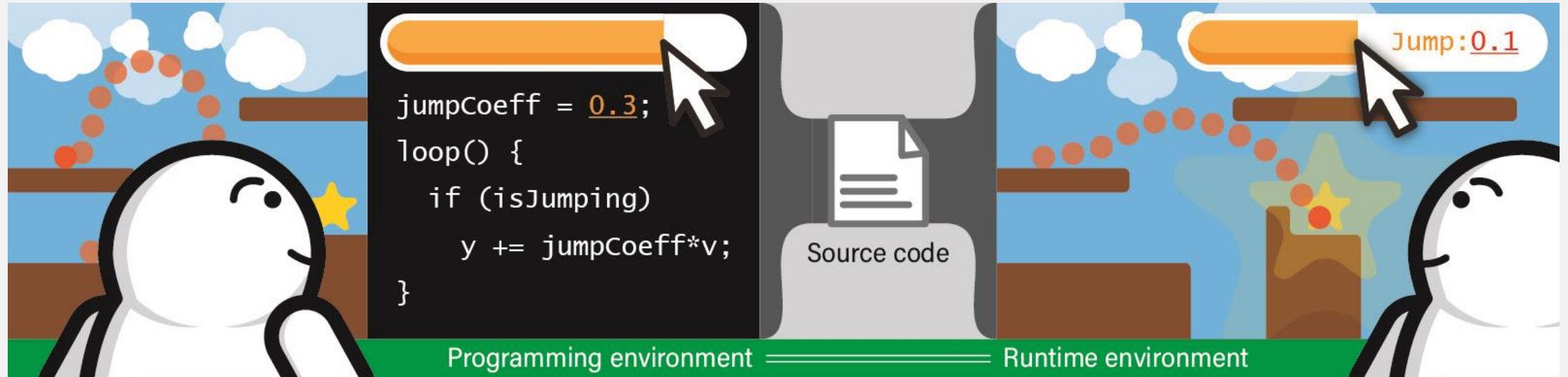


Multi-layered user interfaces for programmers and users

A programming environment is usually designed to consist of (merely) computational artifact



A merely technical approach is “scalable” but sometimes in short of “social inclusion”



- Computational support **X** is available for many people but for some people with characteristics **Y**
- For instance, **f3.js** can be used by a variety of people but is not designed for **people with visual impairment**

The above studies emphasise how rapid prototyping tools hold great potential for producing individualised, and affordable, AT. However, despite use of the DIY acronym, people with disabilities have been framed as primarily users or consumers, rather than producers, of DIY-ATs. For example, Hook et al. defined DIY-AT generally as ‘the development of AT by non-professionals’ [29:598]—referring to parents, friends and care-givers. Buehler et al. [14] noted that it is rarely the people with disabilities themselves who create and share DIY-AT designs online. While *DIY* is not an unfamiliar concept in AT, it seems to

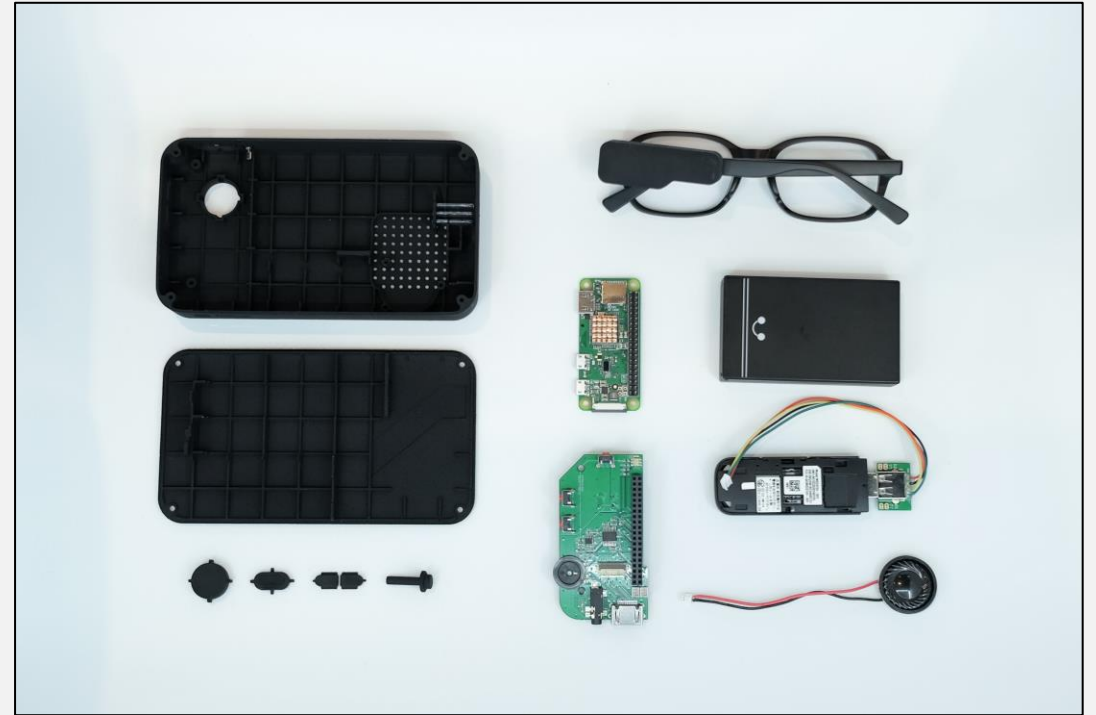
Do-It-Yourself Empowerment as Experienced by Novice Makers with Disabilities

Meissner et al., ACM DIS 2017



## **OTON GLASS**

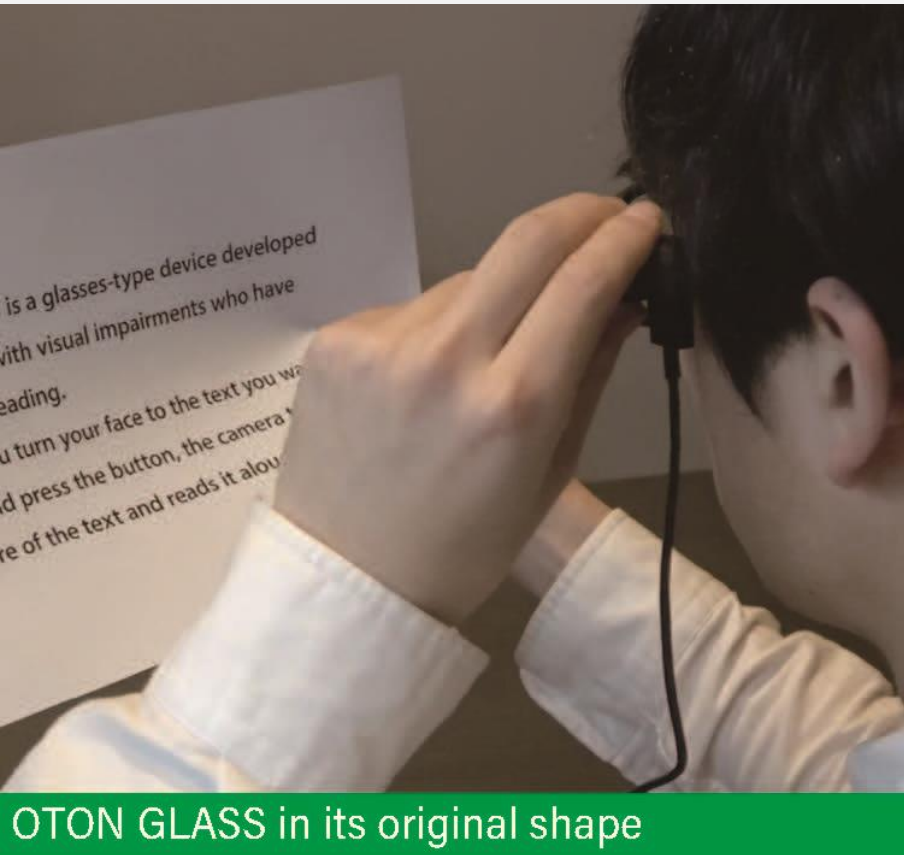
A smart glasses built with a 3D printer and Raspberry Pi to help the father of one of the authors (Keisuke) who acquired dyslexia



# OTON GLASS as a toolkit



# People with visual impairments teamed up with evangelists to develop smart glasses

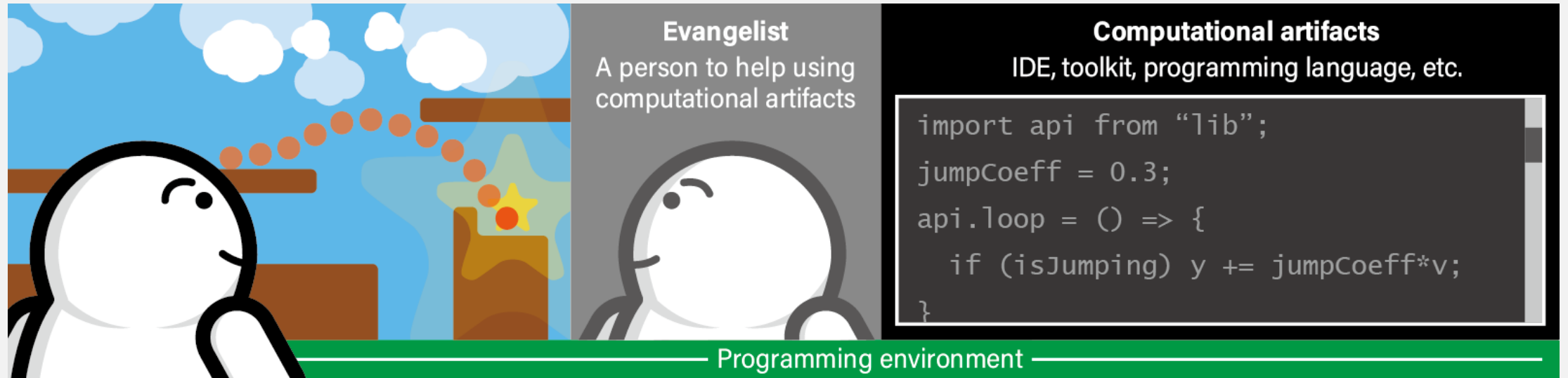






A Paralympian with visual impairment and his professional supporter proposed to add the voice chat feature and implemented it with help of the evangelists.

# An evangelist as part of a programming environment



- Social inclusion is a hybrid of technical and social implementations
- Programming environment design can be community design
- Programming as communication between diverse kinds of people

# Programming by a community of people

- A programming environment consists of not only computational artifacts but also a community of people to collaborate
- “People are message-passing objects” [Salon 2020 Day 1]

# Programming as communication

- Unlike tailor-made model, “the programmer” with visual impairment produces ideas and decides what to build
- The programming activity inherently involves communication and enables empowerment

# Rethinking programming “environment”

## BACKGROUND

Programming experience design for development of programs that run in the real world

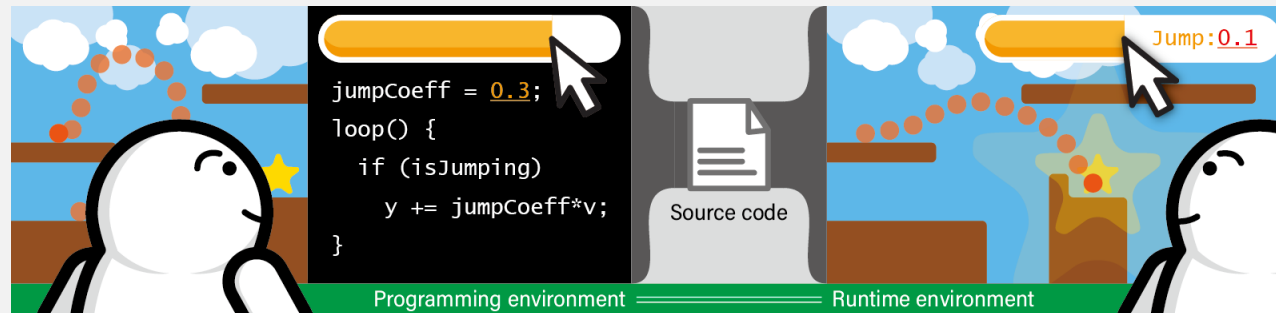
## TOWARD CONVIVIAL COMPUTING

- Technical environment design for collaborations
- Social environment design for more inclusion

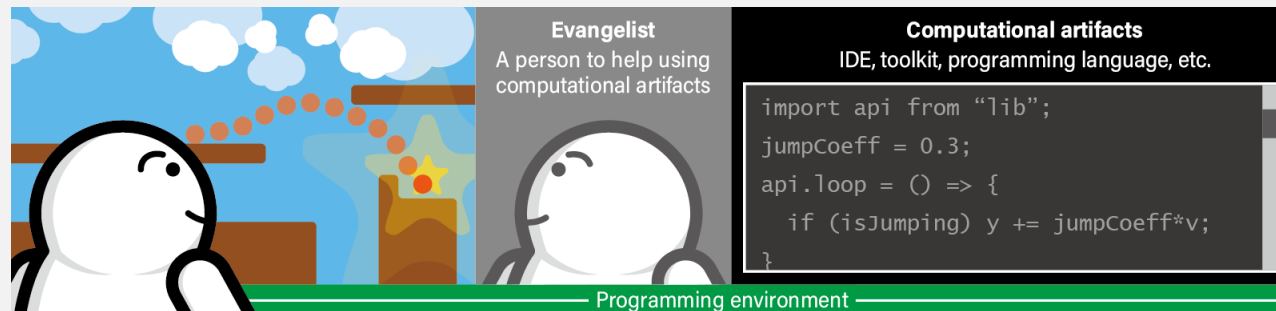
## CONCLUSION

# Programming Experience (PX) research for convivial computing

Programming environments are usually designed exclusively for programmers but **should be more inclusive!**



They can be shared among programmers and users



They can be hybrid of people and computational artifacts