PROGRAMMING LANGUAGES ARE USER INTERFACES

Andrew J. Ko, Ph.D.

Associate Professor
Adjunct Associate Professor
Chief Scientist

The Information School
Computer Science and Engineering
AnswerDash, Inc.
CODE IS CHANGING THE WORLD
010100100100101001010010
100101001010010101001010
101010010101010010101010
100100010010111001010101
001010010101010010010100
101010100010101010010101
001010101001010101100100
BUT THE WORLD ISN’T CHANGING CODE

it’s still difficult to learn, write, test, debug, design, deploy, fix etc.
headlines from the last month

Computer Error Costs Indiana Millions In Education Grants

United Continental CEO: Still fixing bugs in new computer system

Computer glitch hampers IMPD communications for 4 days

Computer Glitch Leads to $1 Gas (Sweet!)

ICANN Extends New Domain Deadline Because of Bug

Computer Glitch Means No Licenses, IDs

Computer Glitch Dashed High School Hopes for Five Queens Girls

Computer glitch causes hospital billing errors

Bats CEO Says Computer Glitch “Unfortunate”

State Panel Wants Answers about Prison Computer Glitch

Computer Glitch Delays NJ Jobless Claims
developers use the wrong languages
teams lack effective methodologies
CS education fails to adequately prepare
tools fail to compensate for human fallibility
developers use the wrong languages
teams lack effective methodologies
CS education fails to adequately prepare
tools fail to compensate for human fallibility

ALL OF THESE ARE HUMAN PROBLEMS
because

PROGRAMMING LANGUAGES ARE USER INTERFACES
PROGRAMMING LANGUAGES ARE USER INTERFACES

Some history on this viewpoint
Research on the topic
Open questions
IN THE BEGINNING
(the early 1940's anyway)

Programmers Betty Jean Jennings (left) and Fran Bilas (right) operate the ENIAC's main control panel at the Moore School of Electrical Engineering
SEPARATING HARDWARE AND SOFTWARE

the IBM punchcard
INTERACTIVE COMPUTING

Douglas Engelbart, 1968
what made this different was the speed with which the computer reacted to human input

no longer necessary to write and wait

feedback loops between people and computers were reduced to milliseconds

the result of ones commands could be seen immediately, allowing people to engage in the rapid exchange of information
BATCH COMPUTING
programming

INTERACTIVE COMPUTING
GUIs
web sites
mobile apps
Kinect
....
BATCH COMPUTING

manipulate a computer’s future behavior through abstract notation

INTERACTIVE COMPUTING

manipulate the computer’s present behavior through concrete notations

researchers started to ask...

“why can’t code be interactive like every other kind of document?”
InterLisp: syntax highlighting, spell checking, auto-complete, version control, integrated debugger, etc.

A vision for writing, executing, and understanding code interactively.
INTERACTIVE CODE 1980–2000

these ideas go mainstream

Turbo Pascal 1983

Eclipse 2004
THE PRESENT AND FUTURE

What’s hard about making programming environments more usable?

What progress have we made?
Think of programming platforms as a collection of programming interfaces:

- Language constructs, functions, classes, libraries, APIs, types, etc.

I claim that all barriers in programming arise from:

- **Problem solving challenges** inherent to devising algorithms and data structures to solve a problem (which I called DESIGN barriers)

- **Usability problems** with the programming interfaces necessary to express these solutions
SIX BARRIERS IN PROGRAMMING

Discuss with your neighbor:

What was useful about the paper?
What was surprising?
What was less useful?
SELECTION barriers

Finding programming interfaces that implement a particular behavior

- Reading API documentation
- Asking a friend
- Using a code search engine
- Searching Stack Overflow
USE barriers

Discovering the intended way to use a programming interface (syntax, inputs, outputs, side effects, preconditions, postconditions, etc.)

Reading documentation about a function, class, or method, writing test cases
COORDINATION barriers

Discovering usage rules that govern how programming interfaces can be composed

Reading Stack Overflow, searching for error messages on Google, reading documentation
UNDERSTANDING barriers

Difficulties interpreting the unexpected behavior of a programming interface

Searching Google for an error message, test case minimization, guessing

CrypticUndocumentedException
INFORMATION barriers

Difficulties observing the internal behavior of a programming interface

Finding a better debugging tool, writing the perfect print statement, selecting the perfect breakpoint
PROGRESS

addressing these barriers
solutions to USE barriers
Alice (2007)

what if syntax and type errors were impossible
(removes USE barriers)

Scratch (2008)


same idea as Alice: drag and drop prevents syntax and type errors (*removes USE barriers*)

what if you could embed anything in a source file, in context? (removes USE barriers)
solutions to SELECTION barriers

what if programs could be guessed from natural language? (removes SELECTION barriers)

discussion paper!
CoScripter (2008)

what if web interactions could be recorded and replayed? (removes SELECTION barriers)

Reform (2009)

web mashups through interactive web scraping (removes SELECTION barriers)

what if web service mashups could be constructed by selecting examples? (removes SELECTION barriers)
Mica (2006)

Mines an API to augment Google search results with classes and methods
solutions to COORDINATION barriers
Intelligent API tutors

Generates instructional tasks from online FAQs and open source code providing more explanation and context about API usage rules

To check if an element is an array in JavaScript, I have always used Crockford's function (pg 61 of The Good Parts):

```javascript
var is_array = function (value) {
    return value &&
        typeof value === 'object' &&
        typeof value.length === 'number' &&
        typeof value.splice === 'function' &&
        !(value.propertyIsEnumerable('length'));
};
```

But if I'm not mistaken, recently some guy from Google had found a new way on how to test for a JavaScript array, but I just can't remember from where I read it and how the function went.

Can anyone point me to his solution please?

[Update]
The person from Google who apparently discovered this is called Mark Miller.

Now I've also read that from this post that his solution can easily break as well:

```javascript
// native prototype overloaded, some js libraries extends them
Object.prototype.toString= function(){
    return 'Object Array';
}

function isArray ( obj ) {
    return Object.prototype.toString.call(obj) === 'Object Array';
}

var a = [];
alert(isArray(a)); // returns true, expecting false;
```

So, I ask, Is there any way that we can truly check for array validity?
solutions to UNDERSTANDING barriers
Stack Overflow

A searchable repository of human readable explanations of error messages and other strange behavior.
HelpMeOut (2010)

what if fixes to error messages could come from everyone who’d fixed the error before?

(removes UNDERSTANDING barriers)

what if you could test spreadsheets by simply marking which values are right and wrong?

(removes UNDERSTANDING barriers)
solutions to INFORMATION barriers
DuctileJ (2011)

what if programmers could run their programs whenever they wanted to, regardless of compiler errors? (removes INFORMATION barriers)

TimeLapse
precise deterministic replay of web applications
/removes INFORMATION barriers/
Code Canvas (2010)  

what if you could see all of your code and its dependencies on a single screen? (removes INFORMATION barriers)
A Working Set Interface (2006)

A design sketch I created

Code Bubbles (2010) what if IDEs sliced code up into snippets instead of files? (removes INFORMATION barriers)
Debugger Canvas

6 years from idea to Visual Studio plug-in
WHAT’S NEXT?

010100100100101001010010
100101001010010101001010
101010010101010010101010
100100010010111001010101
001010010101010010010100
101010001010101000101000
101010001010101000101001
0010101010010101100100
PRODUCTIVITY IS DONE

New dev tools are fine, but they’re increasingly incremental, niche and irrelevant to industry

Productivity is not the problem, it’s learning, expertise, design, iteration, scale, domains

Look ahead 20 years…

What will we be coding?
Who will be coding it?
Who will they coding it for?
How should they be coding it?
NEW KINDS OF CODE

Machine-learned
How do we code against uncertainty?

Crowd-powered
How do we code against human cognition?

Biological
How do we code against anatomy and physiology?

Cloud-powered
How do we code against data centers, social networks, and massive data sets?
BETTER DEVELOPERS

Instead of making better tools, why not make better developers?

Training end-users
How can we insert education into end-user programming tools?

Teaching novices
How can we teach learners more efficiently and effectively?

Facilitating experts
How can we help engineers make more effective decisions?

Structuring teams
How can we help teams coordinate work more effectively?
Teaching Problem Solving (2016)

What if we taught novice programmers how to structure and reflect on their programming efforts?

One hour of instruction on six stages:

1) interpreting problem prompt,
2) search for analogous problems,
3) search for solutions,
4) evaluate solutions,
5) implement solution,
6) evaluate implementation

Upon help requests, prompt for reflection: “What are you doing, why are you doing it, and is it working”?

Teaching Problem Solving (2016), cont.

Two camps, two weeks, 25 students each

20 requirements to implement for a web application

Campers with the instruction were more productive, more creative, more independent, more confident in their ability to code and learn other non-coding skills.
CS Ed for All

President Obama just announced a $4 billion initiative to:

Prepare and place 10,000 CS teachers in U.S. public schools

Fund $125 million in CS ed research per year, including NSF graduate fellowships, CAREER grants, basic research funding, faculty positions, etc.

The computing education research community will grow from ~50 researchers now to ~500 researchers in the next twenty years