Making communicating with computers more accessible:

easier, faster, and safer
Welcome!

• This is a graduate course; undergrads are welcome.
  • can have taken 152 or 179 and be just fine, not necessarily both
• You (as a student presenter) will present and lead discussion for at least one paper
• You (as a non-presenting student) will post questions and a summary of the design arguments by Friday of the previous week
• Key learning outcomes:
  • (279r) to look at scientific publications, identify the core design arguments, write new design arguments, and evaluate them
  • (252r) understand, design and implement language abstractions for solving a task
• Group projects will be composed of both “HCI folks” and “PL folks”
Welcome!

• In undergraduate courses, you consume knowledge and practice applying it.

• In graduate courses like this one, you attempt to generate new knowledge.
I’m an HCI person.
I build novel interfaces and evaluate them in studies.

Since ~2015, I have found **PL technology** useful for providing the *magic* behind the screen.
Evaluation is Hard.

But also, evaluation with respect to what?
Outline

1. Design Arguments
2. Some HCI evaluation techniques
3. Means of communicating with computers
Design Arguments

Need Thesis

Stakeholders + Domain
Person P [in setting S] wants to achieve goal G but obstacles O1-N get in the way.

Core tension
Any solution also has to:
satisfy constraints X1-N,
minimize costs Y1-N,
and avoid obstacles Z1-N.

Axioms
As designers, we bring the following principles and constraints A1-N.

Approach Thesis
Our approach, ____________, has characteristics C1-N that help stakeholders achieve their goal G while avoiding obstacles O1-N.

Evidence
How do you know?
How do existing approaches fail?
What characteristics have you borrowed from solutions that succeeded in analogous settings?
What differentiates your approach from previous solutions that failed?
How have stakeholders responded to/been able to use your approach?
Visualizing API Usage Examples at Scale

Elena L. Glassman*, Tianyi Zhang†, Björn Hartmann‡, Miryung Kim§
† UC Berkeley, Berkeley, CA, USA
‡ UC Los Angeles, Los Angeles, CA, USA
{eglassman, bjoern}@berkeley.edu, {tianyi.zhang, miryung}@cs.ucla.edu

Figure 1. EXAMPLERE takes a focal API call that a programmer is interested in, locates uses of that API call in a large corpus of mined code examples, and then produces an interactive visualization that lets programmers explore common usage patterns of that API across the corpus.

ABSTRACT
Using existing APIs properly is a key challenge in programming, given that libraries and APIs are increasing in number and complexity. Programmers often search for online code examples in Q&A forums and read tutorials and blog posts to learn how to use a given API. However, there are often a massive number of related code examples and it is difficult for a user to understand the commonalities and variances among them, while being able to drill down to concrete details. We introduce an interactive visualization for exploring a large collection of code examples mined from open-source repositories at scale. This visualization summarizes hundreds of code examples in one synthetic code skeleton with statistical distributions for canonicalized statements and structures enclosing an API call. We implemented this interactive visualization for a set of Java APIs and found that, in a lab study, it helped users (1) answer significantly more API usage questions correctly and comprehensively and (2) explore how other programmers have used an unfamiliar API.

INTRODUCTION
Learning how to correctly and effectively use existing APIs is a common task — and a core challenge — in software development. It spans all expertise levels from novices to professional software engineers, and all project types from prototypes to production code. The landscape of publicly available APIs is massive and constantly changing, as new APIs are created in response to shifting programmer needs. Within companies, the same is true, perhaps even more so: joining a company can require learning a whole new set of proprietary APIs before a developer becomes an effective contributor to the company codebase. Developers often are stymied by various learning barriers, including overly specific or overly general explanations of API usage, lack of understanding about the interaction between multiple APIs, lack of alternative uses, and difficulty identifying program statements and structures related to an API [11, 19, 5].

One study found that the greatest obstacle to learning an API is "insufficient or inadequate examples." [19] Official documentation...
ABSTRACT
Using existing APIs properly is a key challenge in programming, given that libraries and APIs are increasing in number and complexity. Programmers often search for online code examples in Q&A forums and read tutorials and blog posts to learn how to use a given API. However, there are often a massive number of related code examples and it is difficult for a user to understand the commonalities and variances among them, while being able to drill down to concrete details. We introduce an interactive visualization for exploring a large collection of code examples mined from open-source repositories at scale. This visualization summarizes hundreds of code examples in one synthetic code skeleton with statistical distributions for canonicalized statements and structures enclosing an API call. We implemented this interactive visualization for a set of Java APIs and found that, in a lab study, it helped users (1) answer significantly more API usage questions correctly and comprehensively and (2) explore how other programmers have used an unfamiliar API.

INTRODUCTION
Learning how to correctly and proficiently use an API is a common task — and a core component of program development. It spans all expertise levels, from hobbyists to professional software engineers, and all phases of the software development life cycle, from development to production code. The landscape of APIs is massive and constantly changing, and developers must respond to shifting programming needs. Our research shows that the same is true, perhaps even more so, for learning new APIs. Even if developers do not require learning a whole new API, using APIs often becomes an essential part of building software systems. Developers often face many challenges, including the need to select the right APIs from a vast selection of options, API documentation, and specifications of API usage, lack of understanding of the context in which an API is used, and the need to understand the relationships between multiple APIs, lack of understanding of the interactions between different APIs, and identifying program statements that invoke the API [11, 19, 5].

One study found that the greatest barrier to using APIs is the availability of “insufficient or inadequate examples.”
Outline

1. Design Arguments
2. Some HCI formative design and evaluation techniques
3. Means of communicating with computers
Some HCI Formative Design Techniques

• Survey

• Interview

• Contextual inquiry
  • Observation in context
  • Requests for explanation

• Wizard of Oz

• Technology probe
Some HCI Evaluation Techniques

• User study
  • Task design
  • Metrics

• Deployment

• Interview or survey of deployment participants

• Crowdsourcing, i.e., Amazon Mechanical Turk
Outline

1. Design Arguments
2. Some HCI formative design and evaluation techniques
3. Means of communicating with computers
Communicating with Computers

• Human intent
  • Examples
  • Statement(s) in a programming language
  • Natural language
  • …

• Computer’s interpretation
  • Program
  • Behavior
    • Action in response to a human request
    • Results of running understood program on additional data
Human

capabilities

capabilities w/ an interface

Synthesizer (++)
capabilities (++)
Approach

Human

capabilities

capabilities w/ an interface

bounded by psychological limits, learning rate

design intervention

Synthesizer (++)

capabilities (++)

theoretically bounded
Approach

Human

Capabilities

Capabilities w/ an interface

bounded by psychological limits, learning rate

capabilities (++)

design intervention

capabilities (++)

Simplifier (++)

programming language examples specifications serving as an oracle

theoretically bounded
Approach

Human

Co-optimize

programming lang
natural lang examples
specifications serving as an oracle

bounded by psychological limits,
learning rate

capabilities w/ on interface

capabilities (++)

capabilities (++)

theoretically bounded
<table>
<thead>
<tr>
<th>Blue</th>
<th>Green</th>
<th>Yellow</th>
</tr>
</thead>
<tbody>
<tr>
<td>12. A. Chlipala. Modular development of</td>
<td>A. Chlipala. Modular development of</td>
<td>A. Chlipala. Modular development of</td>
</tr>
</tbody>
</table>
PROSE Architecture

Example-based intent spec $\varphi$ → Program Synthesizer

Program Synthesizer

Ranked program set $\tilde{N}$ → Debugging

Debugging

Test inputs $\hat{\sigma}$ → Intended program $P \in \mathcal{L}$

Intended program in Python/C#/C++/...

Credit: Alex Polozov
“FlashFill in Excel is designed to cater to users that care not about the program but about its behavior on the small number of input rows in the spreadsheet.

Such users can simply eye-ball the outputs of the synthesized program and provide another example if they are incorrect.

However, this becomes much more cumbersome (or impossible) with a larger spreadsheet.”

“We have observed that inspecting the synthesized program directly also does not establish enough confidence in it even if the user knows programming.

Two main reasons for this are
(i) program readability, and
(ii) the users’ uncertainty in the desired intent due to hypothetical unseen corner cases in the data.”

What is particularly hard about evaluating methods for communicating with computers?
Evaluation

Why is it hard:
some puzzlers from programming languages
puzzler 1

easy vs safe
Array<Cat> <= Array<Animal>
puzzler 2
semantics
let x = 1 in
let f = let x = 2 in (\y -> y+x)
let x = 3 in
f 0

Is the result 1, 2 or 3?
puzzler 3
cognitive overhead
Programmer

Hardware

general purpose compiler

(illustration: Markus Püschel)
Programmer

Hardware

- horizontal and vertical extensibility
- generic optimizations at each level (cse, dce, ...)

Matrix, Graph, ...
Array, Struct, Loop, ...
SIMD, GPU, cluster, ...
Staging?

• multi-level language
  \[ n \mid x \mid e \ \alpha^b e \mid \lambda^b x. e \mid \ldots \]

• MetaML / MetaOCaml
  \[ n \mid x \mid e e \mid \lambda x. e \mid <e> \mid \sim e \mid \text{run } e \]

• Lightweight Modular Staging (LMS) in Scala
  driven by types: \( T \) vs \( \text{Rep}[T] \)
“People confuse the familiar for the simple. For new features, people insist on LOUD explicit syntax. For established features, people want terse notation.”

–Bjarne Stroustrup
Power in Scala

```scala
def square(x: Int): Int = x*x

def power(b: Int, n: Int): Int =
  if (n == 0) 1
  else if (n % 2 == 0) square(power(b, n/2))
  else b * power(b, n-1)

// power(2, 7) == 128
```
def square(x: Rep[Int]): Rep[Int] = x*x

def power(b: Rep[Int], n: Int): Rep[Int] =
    if (n == 0) 1
    else if (n % 2 == 0) square(power(b, n/2))
    else b * power(b, n-1)

def snippet(b: Rep[Int]) = power(b, 7)
class Snippet extends ((Int)=>(Int)) {
    def apply(x0: Int): Int = {
        val x1 = x0 * x0
        val x2 = x0 * x1
        val x3 = x2 * x2
        val x4 = x0 * x3
        x4
    }
}
Power in OCaml

let square x = x * x

let rec power n x =
    if n = 0 then 1
    else if n mod 2 = 0 then square (power (n/2) x)
    else x * (power (n-1) x)

(* val power : int -> int -> int = <fun> *)
Staged Power in MetaOCaml

let square x = x * x

let rec spower n x =
    if n = 0 then 1.
    else if n mod 2 = 0 then square ~ (spower (n/2) x).
    else ~ x * ~ (spower (n-1) x).

(* val spower : int -> int code -> int code = <fun> *)
let spower7_code = <fun x -> ~(spower 7 <x>.)>.;;

(*
val spower7_code : (int -> int) code = <
  fun x_1 ->
    x_1 * ((CSP square *) (x_1 * 
      ((CSP square *) (x_1 * 1))))>.
*)
Program generically...

... and run specialized!
Program generically...  
(save human time)  

... and run specialized!  
(save computer time)
A matrix vector product, where the matrix is known (static) but the vector is unknown (dynamic),
e.g., for a Hidden Markov Model (HMM) where a single transition matrix is multiplied by many different observation vectors.

–Shonan Challenge for Generative Programming (PEPM’13)
Unstaged

def matrix_vector_prod(a: Array[Array[Int]],
v: Array[Int]) = {
  val n = a.length
  val v1 = new Array[Int](n)
  for (i <- (0 until n)) {
    for (j <- (0 until n)) {
      v1(i) = v1(i) + a(i)(j) * v(j)
    }
  }
  v1
}
def matrix_vector_prod(a0: Array[Array[Int]],
                      v: Rep[Array[Int]]) = {
    val n = a0.length
    val a = staticData(a0)
    val v1 = NewArray[Int](n)
    for (i <- (0 until n):Range) {
        val sparse = a0(i).count(_ != 0) < 3
        for (j <- unrollIf(sparse, 0 until n)) {
            v1(i) = v1(i) + a(i).apply(j) * v(j)
        }
    }
    v1
}
def matrix_vector_prod(a0: Array[Array[Int]],
                      v: Rep[Array[Int]]) = {
    val n = a0.length
    val a = staticData(a0)
    val v1 = NewArray[Int](n)
    for (i <- (0 until n):Range) {
        val sparse = a0(i).count(_ != 0) < 3
        for (j <- unrollIf(sparse, 0 until n)) {
            v1(i) = v1(i) + a(i).apply(j) * v(j)
        }
    }
    v1
}
Example Matrix

val a0 =

A(A(1, 1, 1, 1, 1), // dense
   A(0, 0, 0, 0, 0), // null
   A(0, 0, 1, 0, 0), // sparse
   A(0, 0, 0, 0, 0),
   A(0, 0, 1, 0, 1))
class Snippet(px6:Array[Int]) extends ((Array[Int])=>(Array[Int])) {
  def apply(x0:Array[Int]): Array[Int] = {
    val x2 = new Array[Int](5)
    val x6 = px6 // static data: Array(1,1,1,1,1)
    var x4 : Int = 0
    val x13 = while (x4 < 5) {
      val x5 = x2(0); val x7 = x6(x4); val x8 = x0(x4)
      val x9 = x7 * x8; val x10 = x5 + x9;
      val x11 = x2(0) = x10
      x4 = x4 + 1 }
    val x14 = x2(1); val x17 = x2(1) = x14
    val x22 = x2(2); val x24 = x2(2) = x22
    val x19 = x0(2); val x25 = x22 + x19
    val x26 = x2(2) = x25; val x27 = x2(3); val x29 = x2(3) = x27
    val x30 = x2(4); val x32 = x2(4) = x30
    val x33 = x30 + x19; val x34 = x2(4) = x33
    val x21 = x0(4); val x35 = x33 + x21; val x36 = x2(4) = x35
    x2 }
}
Turning Interpreters into Compilers
Turning Interpreters into Compilers

Stage → Code
input

Code

interpret

result

stage

Code
Turning interpreters into compilers

in: Rep[String]
symbolic input
interpret stage

“abcd”
actual input
exec
result

RE: ^ab*
Scala: if (in(0) == ‘a’) ...
Usability Issues

• cognitive overhead of multi-level language
• notational overhead?
• multi-stage errors
puzzler 4
synthesis
Programming by Example

• $f(1)=2$, $f(2)=3$, ...

• $g(1)=2$, $g(2)=4$, ...

• $h(1)=3$, $h(2)=5$, ...
SMT solver underspecified

• (assert (and (= (f 1) 2) (= (f 2) 3)))

• Model:
  \[ f(x) = \text{if } x=2 \text{ then } 3 \text{ else if } x=2 \text{ then } 3 \text{ else } 2 \]

• Try it: https://rise4fun.com/Z3/bpMYx
SMT solver overspecified

• (assert (forall ((x Int) (y Int))
    (=> (or (and (= x 1) (= y 2))
        (and (= x 2) (= y 3)))
    (= y (+ (* a x) b)))))

• Solution: a=1, b=1, so f(x)=x+1.

• Try it: https://rise4fun.com/Z3/HcJY
SMT solver full program

(declare-const a Int)
(declare-const b Int)
(define-fun f ((x Int)) Int (+ (* a x) b))
(assert (and (= (f 1) 2) (= (f 2) 3)))
(check-sat)
(get-model)

• Try it: https://rise4fun.com/Z3/jhr8
Usability Issues

• Trade-offs, e.g. between expressivity and decidability.
• Need to be precise induces cognitive overhead.
• Precision is brittle.
• Many ways to encode.
Why evaluation is hard

- Many conflicting dimensions: faster, safer, easier
- Does a new toolkit enable new ways
  - of thinking?
  - of programming?
  - of creating?
Discussion

What is particularly hard about evaluating methods for communicating with computers?
Group Projects

- Systems HCI requiring heavy-duty PL
  - Humans modifying DSLs for PBD (programming by demonstration)
  - Examplore with interactively defined templates
- Generic human-centered PL
  - Pick language feature, design it in a human-friendly way
  - Pick a language, describe how—and to what extent—its features are being used in the wild
- Usable + X (PL technique)
  - Usable Generative Programming
  - Usable Probabilistic Programming
  - Usable Type System / Verification
  - Usable Synthesis
    - inductive bias alignment between human and machine
    - ranking function improvements
    - DSL improvements
    - expressing constraints on intermediate states, i.e., equivalence values or types
Thank you!