The Coolest Projects Ever

The First Steps toward founding your “Big Startup”

J’adore le concept !

没开玩笑

Earl T. Barr
e.barr@ucl.ac.uk
earlbarr.com

My Research Venn Diagram

Software Engineering

SE

The Test Oracle Problem

Earl T. Barr, UCL

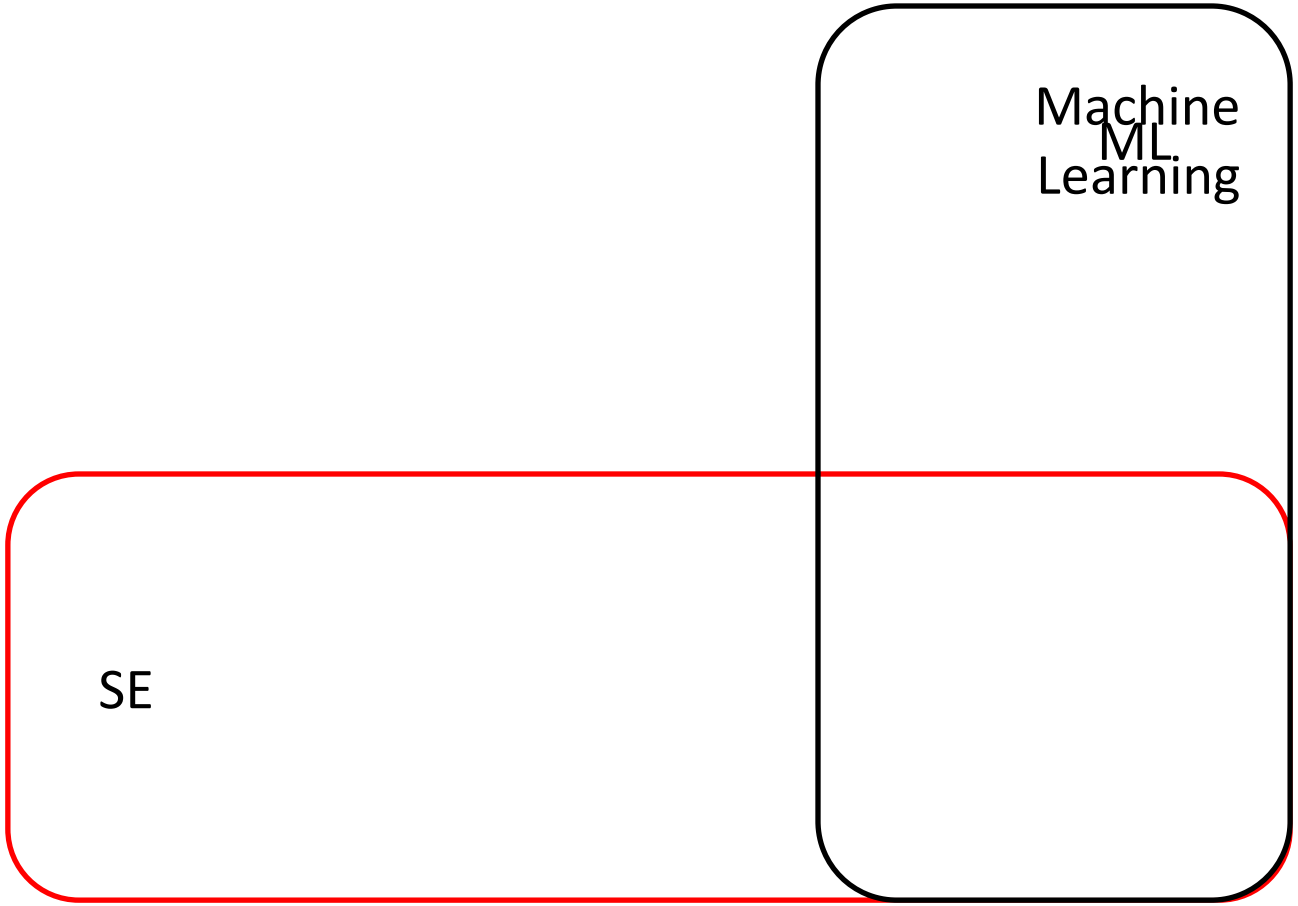
Automatic Software Transplantation

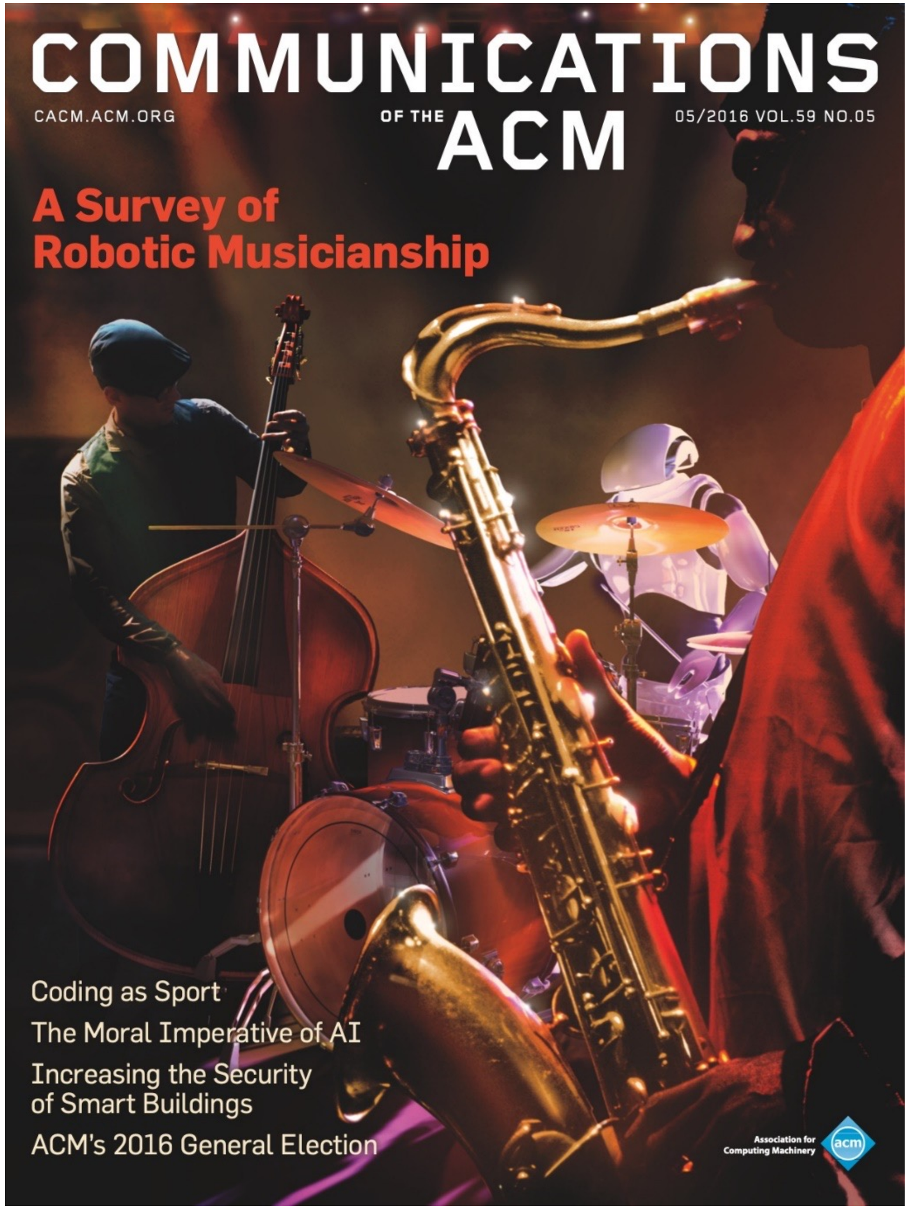
Best Paper, ISSTA 2015

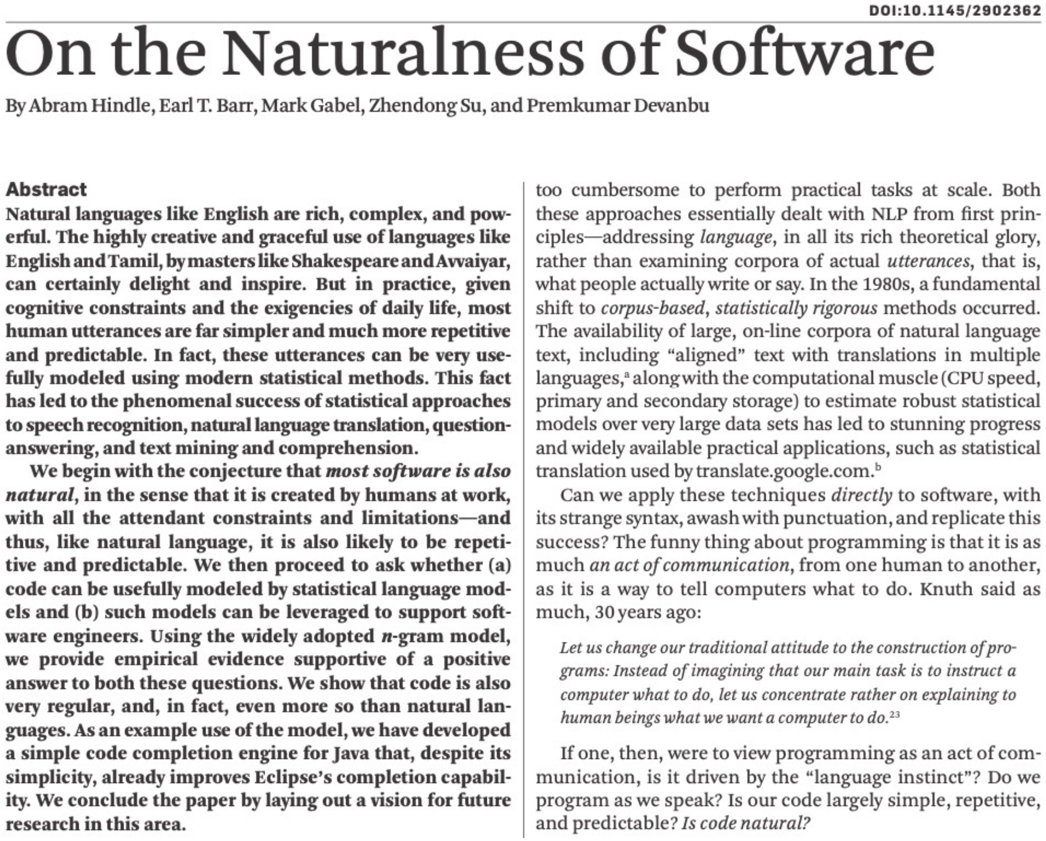
```
QModelIndex start;
if (currentIndex().isValid())
    start = currentIndex();
else
    start = d->model->index(0, 0, d->root);

bool skipRow = false;
bool keyboardTimeWasValid = d->keyboardInputTime.isValid();
qint64 keyboardInputTimeElapsed = d->keyboardInputTime.restart();
if (search.isEmpty() || !keyboardTimeWasValid
    || keyboardInputTimeElapsed > QApplication::keyboardInputInterval()) {
    d->keyboardInput = search;
    skipRow = currentIndex().isValid(); //if it is not valid we should reall
} else {
    d->keyboardInput += search;
}
```

Earl T. Barr, UCL

My Research Venn Diagram




COMMUNICATIONS OF THE ACM
CACM.ACM.ORG
05/2016 VOL.59 NO.05
A Survey of Robotic Musicianship
Coding as Sport
The Moral Imperative of AI
Increasing the Security of Smart Buildings
ACM's 2016 General Election
Association for Computing Machinery
acm


DOI:10.1145/2902362

On the Naturalness of Software

By Abram Hindle, Earl T. Barr, Mark Gabel, Zhendong Su, and Premkumar Devanbu

Abstract

Natural languages like English are rich, complex, and powerful. The highly creative and graceful use of languages like English and Tamil, by masters like Shakespeare and Avvaiyar, can certainly delight and inspire. But in practice, given cognitive constraints and the exigencies of daily life, most human utterances are far simpler and much more repetitive and predictable. In fact, these utterances can be very usefully modeled using modern statistical methods. This fact has led to the phenomenal success of statistical approaches to speech recognition, natural language translation, question-answering, and text mining and comprehension.

We begin with the conjecture that *most software is also natural*, in the sense that it is created by humans at work, with all the attendant constraints and limitations—and thus, like natural language, it is also likely to be repetitive and predictable. We then proceed to ask whether (a) code can be usefully modeled by statistical language models and (b) such models can be leveraged to support software engineers. Using the widely adopted *n*-gram model, we provide empirical evidence supportive of a positive answer to both these questions. We show that code is also very regular, and, in fact, even more so than natural languages. As an example use of the model, we have developed a simple code completion engine for Java that, despite its simplicity, already improves Eclipse's completion capability. We conclude the paper by laying out a vision for future research in this area.

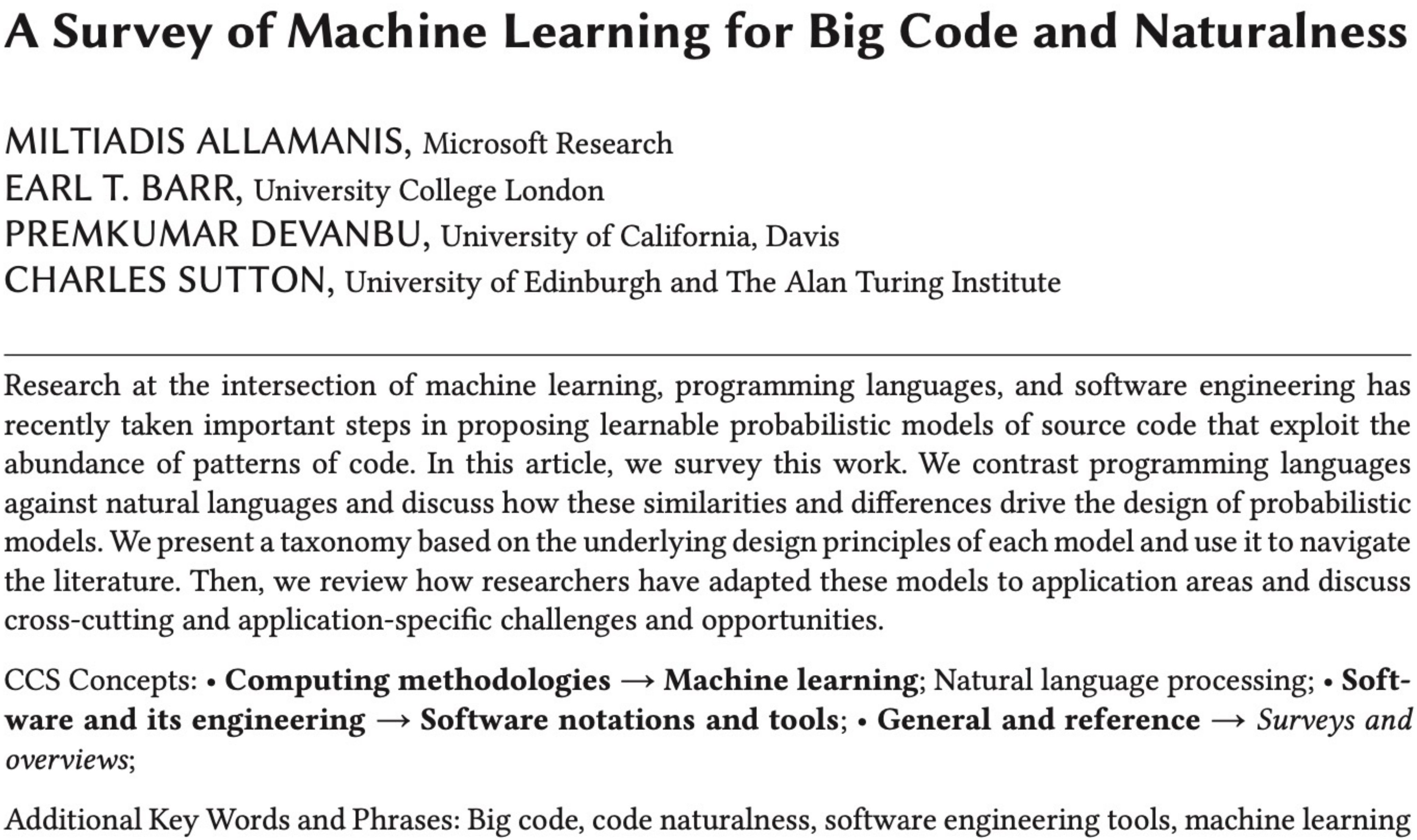
too cumbersome to perform practical tasks at scale. Both these approaches essentially dealt with NLP from first principles—addressing *language*, in all its rich theoretical glory, rather than examining corpora of actual *utterances*, that is, what people actually write or say. In the 1980s, a fundamental shift to *corpus-based*, *statistically rigorous* methods occurred. The availability of large, on-line corpora of natural language text, including "aligned" text with translations in multiple languages,[a] along with the computational muscle (CPU speed, primary and secondary storage) to estimate robust statistical models over very large data sets has led to stunning progress and widely available practical applications, such as statistical translation used by translate.google.com.[b]

Can we apply these techniques *directly* to software, with its strange syntax, awash with punctuation, and replicate this success? The funny thing about programming is that it is as much *an act of communication*, from one human to another, as it is a way to tell computers what to do. Knuth said as much, 30 years ago:

> *Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.*[23]

If one, then, were to view programming as an act of communication, is it driven by the "language instinct"? Do we program as we speak? Is our code largely simple, repetitive, and predictable? *Is code natural?*

Research Highlight, 2016



A Survey of Machine Learning for Big Code and Naturalness

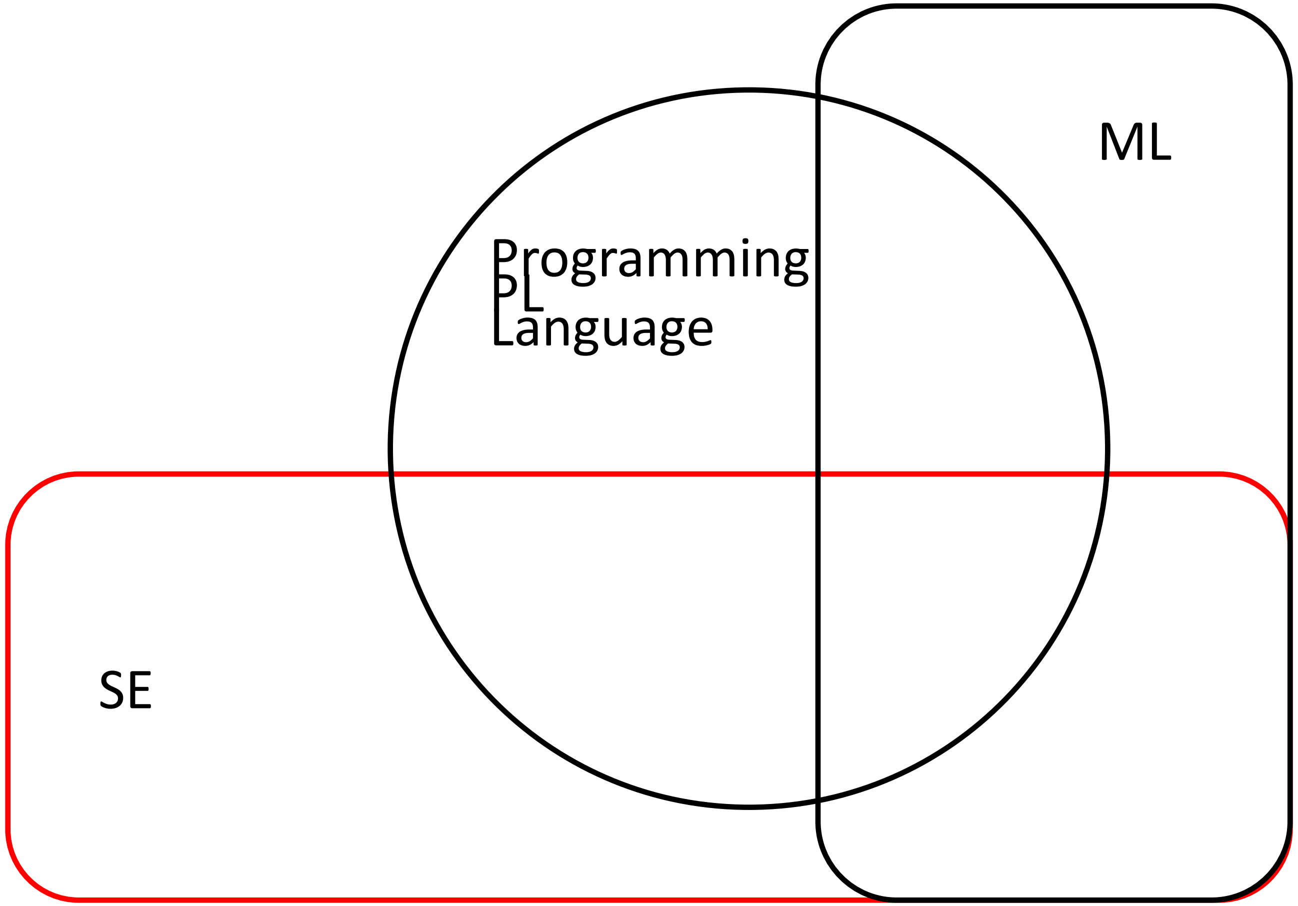
MILTIADIS ALLAMANIS, Microsoft Research
EARL T. BARR, University College London
PREMKUMAR DEVANBU, University of California, Davis
CHARLES SUTTON, University of Edinburgh and The Alan Turing Institute

Research at the intersection of machine learning, programming languages, and software engineering has recently taken important steps in proposing learnable probabilistic models of source code that exploit the abundance of patterns of code. In this article, we survey this work. We contrast programming languages against natural languages and discuss how these similarities and differences drive the design of probabilistic models. We present a taxonomy based on the underlying design principles of each model and use it to navigate the literature. Then, we review how researchers have adapted these models to application areas and discuss cross-cutting and application-specific challenges and opportunities.

CCS Concepts: • **Computing methodologies → Machine learning**; Natural language processing; • **Software and its engineering → Software notations and tools**; • **General and reference →** *Surveys and overviews*;

Additional Key Words and Phrases: Big code, code naturalness, software engineering tools, machine learning

My Research Venn Diagram



Earl T. Barr, UCL

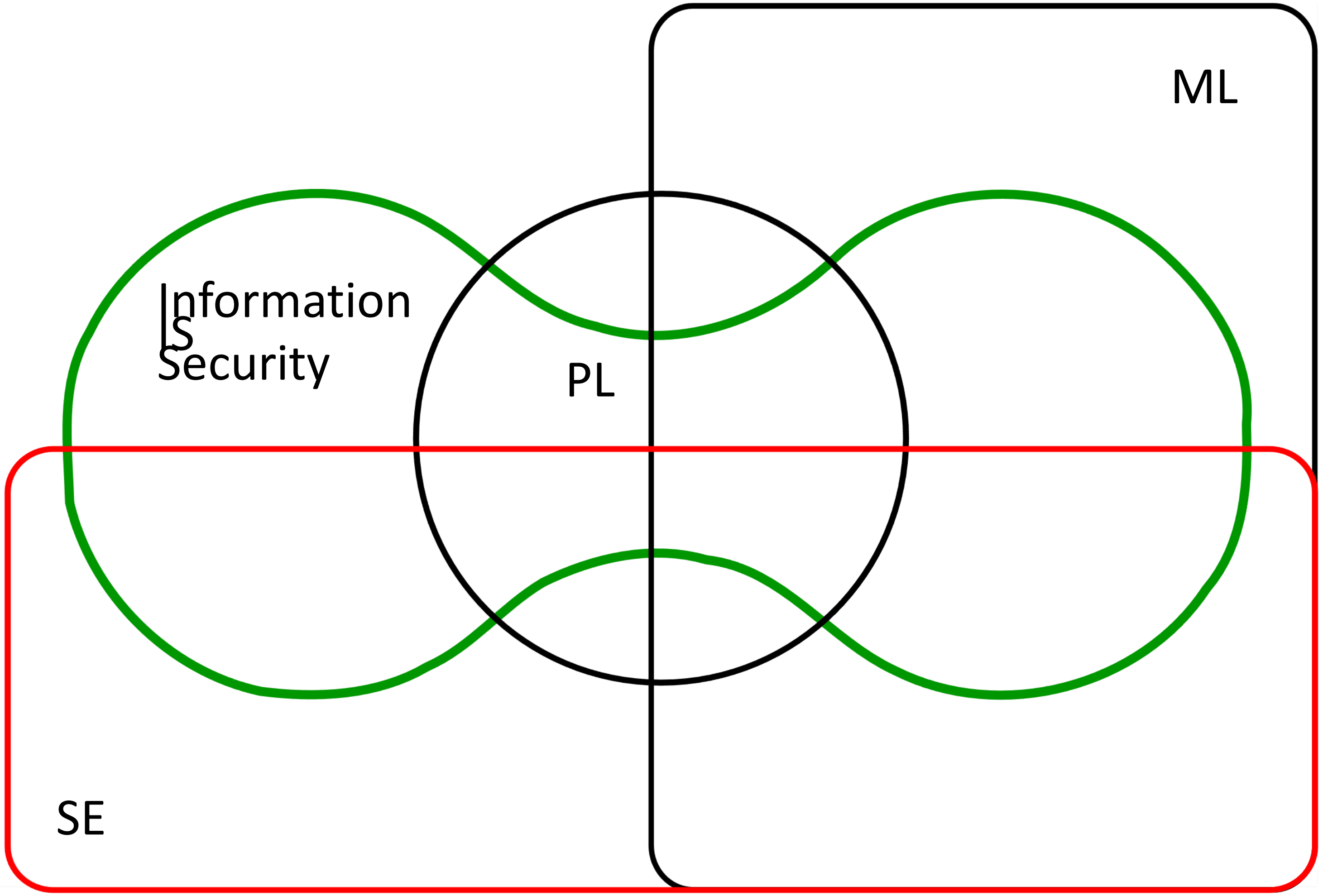
Automatic Detection of Floating-Point Exceptions

Earl T. Barr, UCL

Time-Travel Debugging

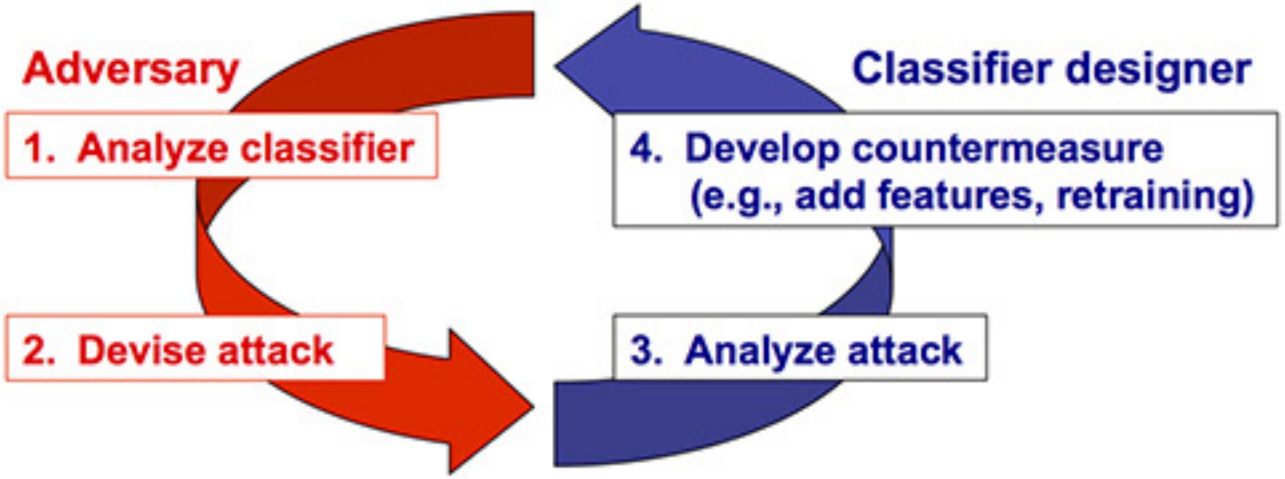
Tardis

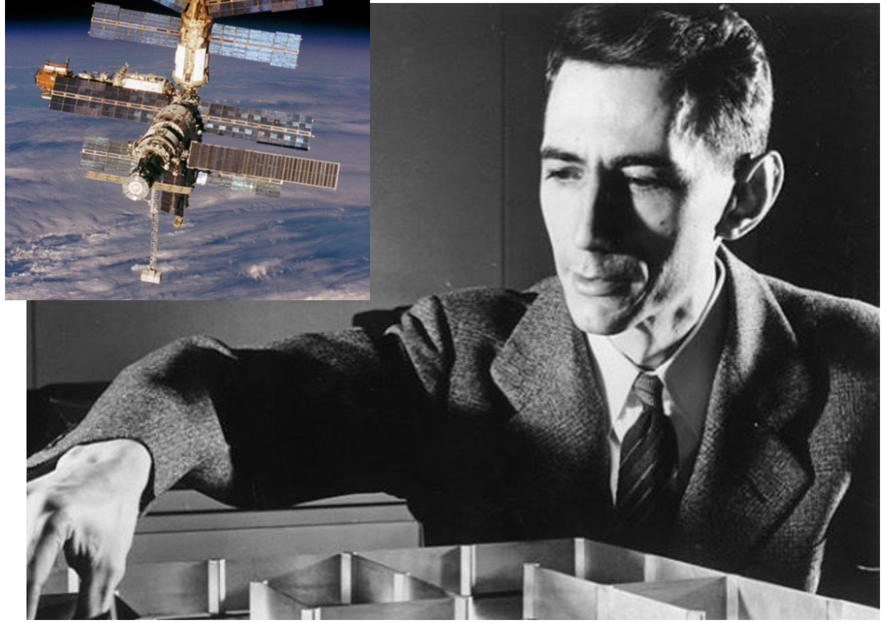
My Research Venn Diagram



Earl T. Barr, UCL

The Arms Race: Adversarial Search Defeats Entropy Used to Detect Malware

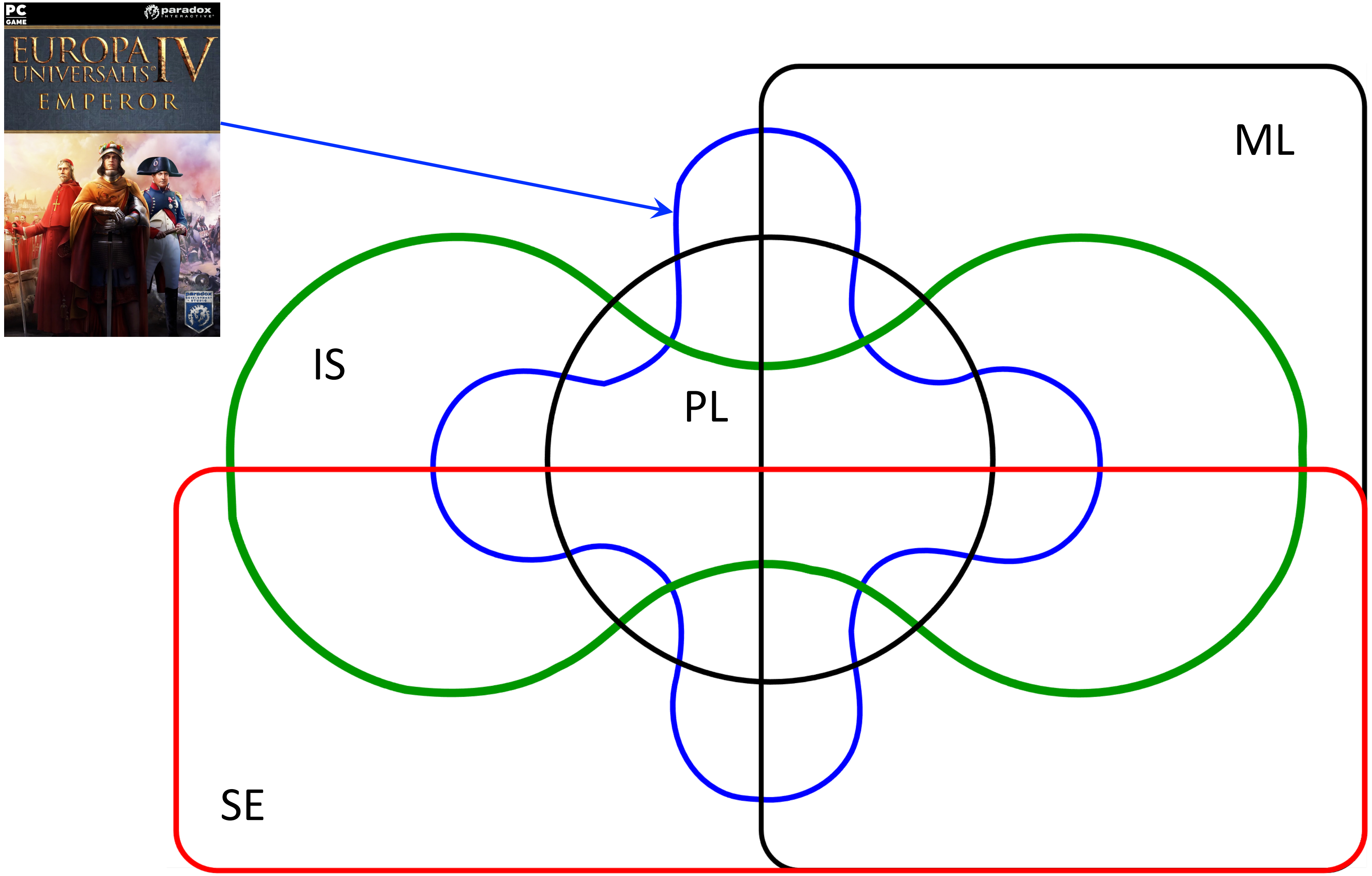




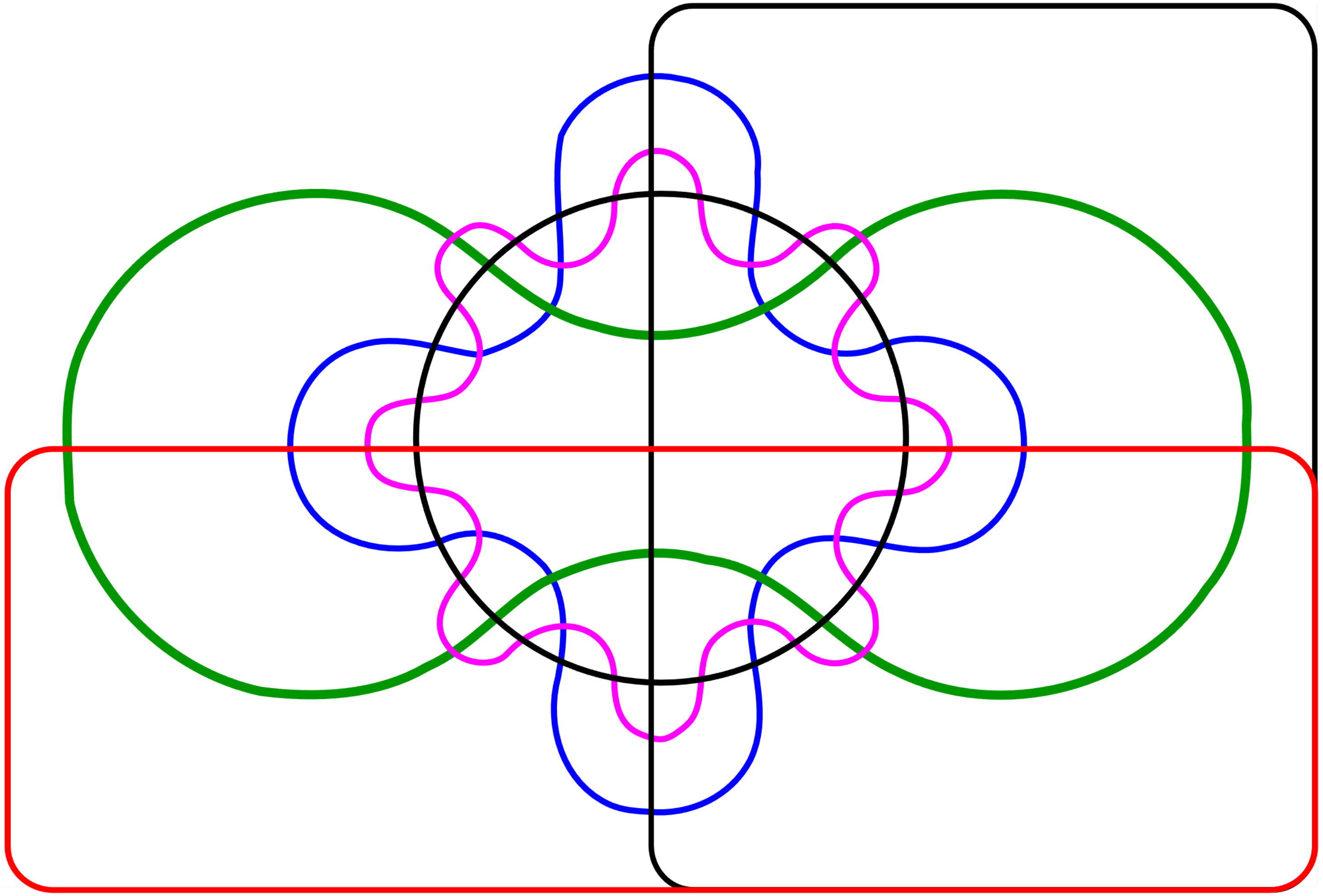
$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_b P(x_i)$$

Earl T. Barr, UCL

Ever Wonder about a 5 Set Venn?



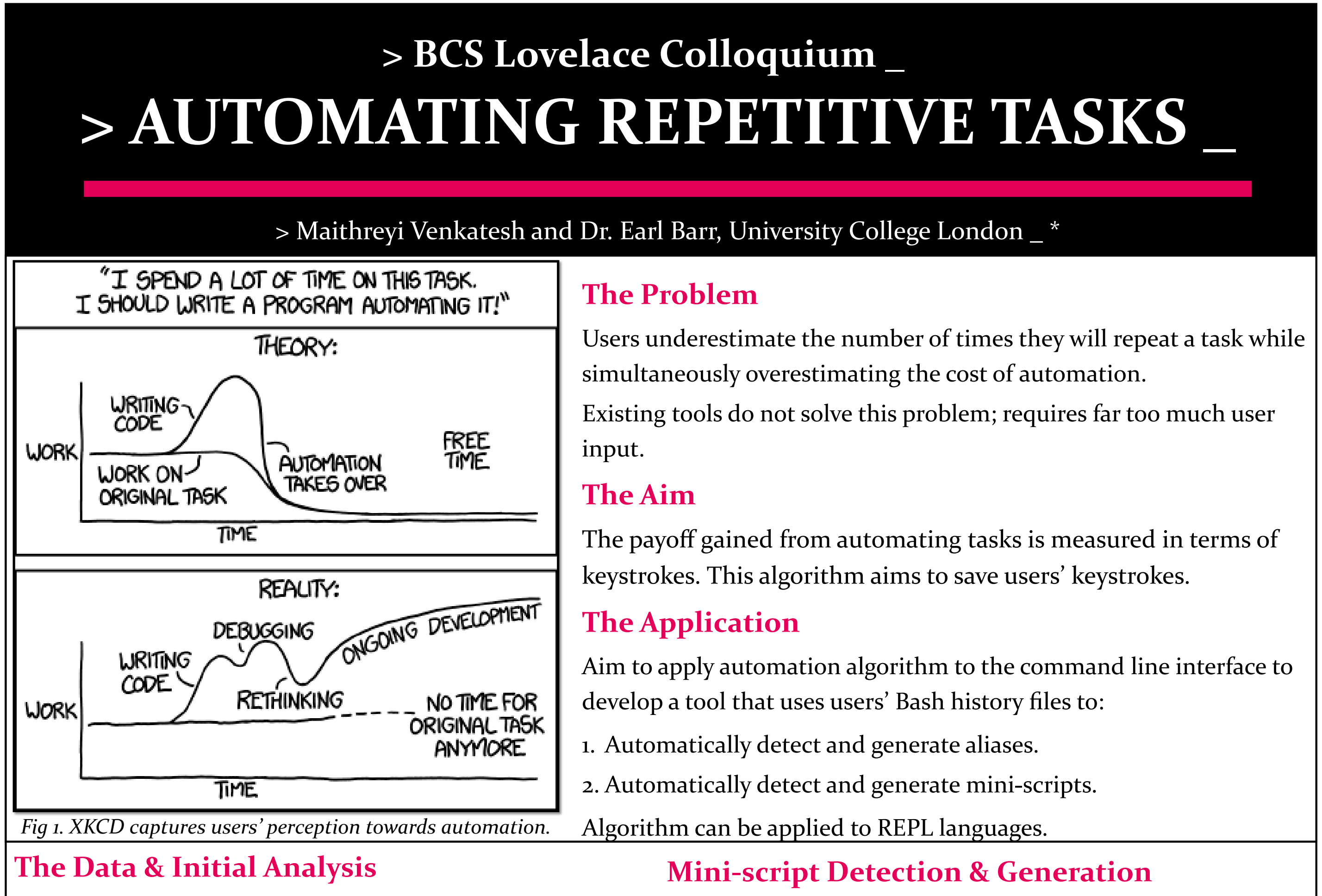
Or 6 Set?



Earl T. Barr, UCL

Previous Student Projects

Projects: Automating the Command Line



"I SPEND A LOT OF TIME ON THIS TASK. I SHOULD WRITE A PROGRAM AUTOMATING IT!"
THEORY:
WORK
WRITING CODE
WORK ON ORIGINAL TASK
AUTOMATION TAKES OVER
FREE TIME
TIME
REALITY:
WORK
WRITING CODE
DEBUGGING
RETHINKING
ONGOING DEVELOPMENT
NO TIME FOR ORIGINAL TASK ANYMORE
TIME

> BCS Lovelace Colloquium _

> AUTOMATING REPETITIVE TASKS _

> Maithreyi Venkatesh and Dr. Earl Barr, University College London _ *

Fig 1. XKCD captures users' perception towards automation.

The Problem

Users underestimate the number of times they will repeat a task while simultaneously overestimating the cost of automation.

Existing tools do not solve this problem; requires far too much user input.

The Aim

The payoff gained from automating tasks is measured in terms of keystrokes. This algorithm aims to save users' keystrokes.

The Application

Aim to apply automation algorithm to the command line interface to develop a tool that uses users' Bash history files to:

1. Automatically detect and generate aliases.
2. Automatically detect and generate mini-scripts.

Algorithm can be applied to REPL languages.

The Data & Initial Analysis

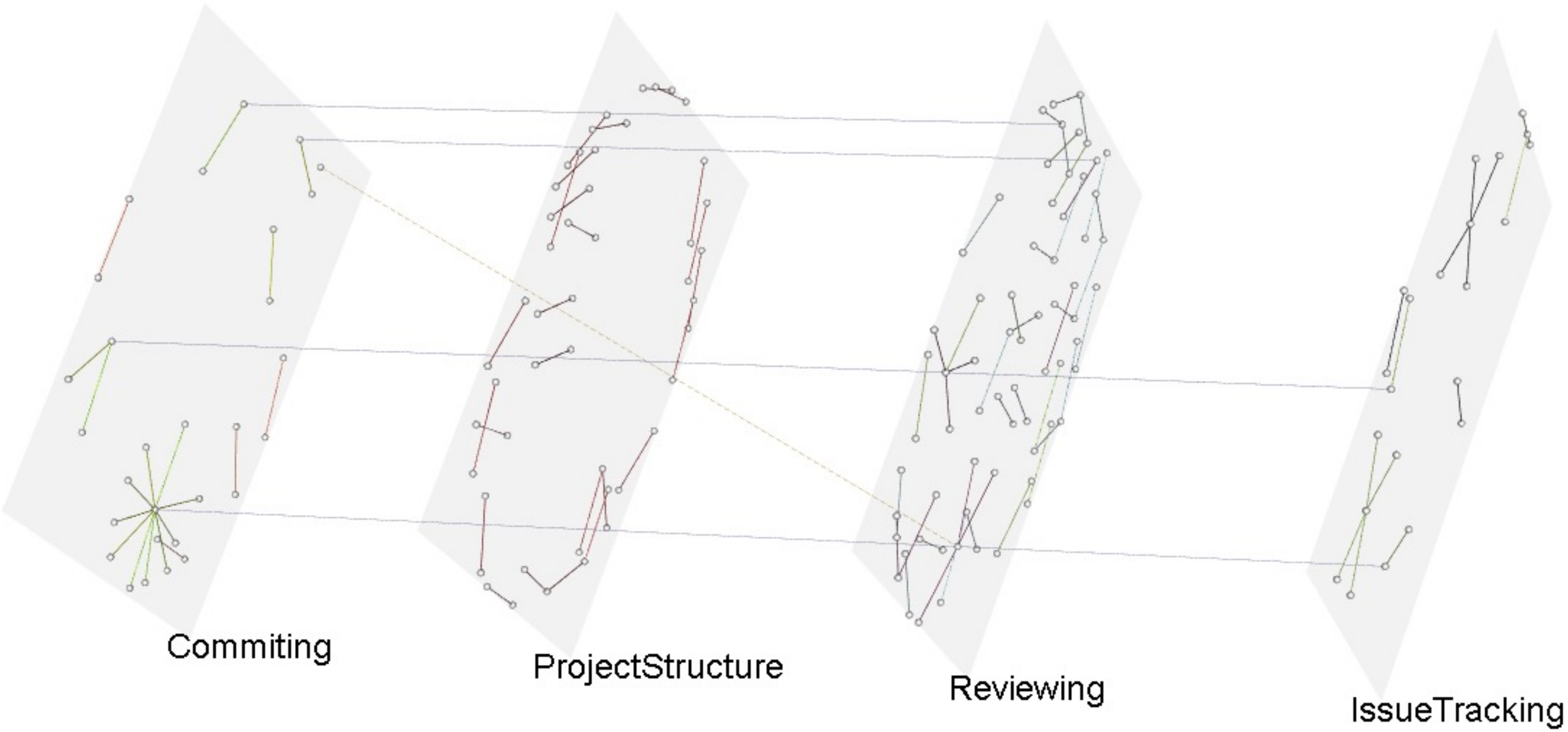
Mini-script Detection & Generation

Maithreyi Venkatesh

The BCSWomen Lovelace Colloquium is an annual one day conference for women students of Computing and related subjects.

https://bcswomenlovelace.bcs.org/

Internships: Multilayer Network Analysis

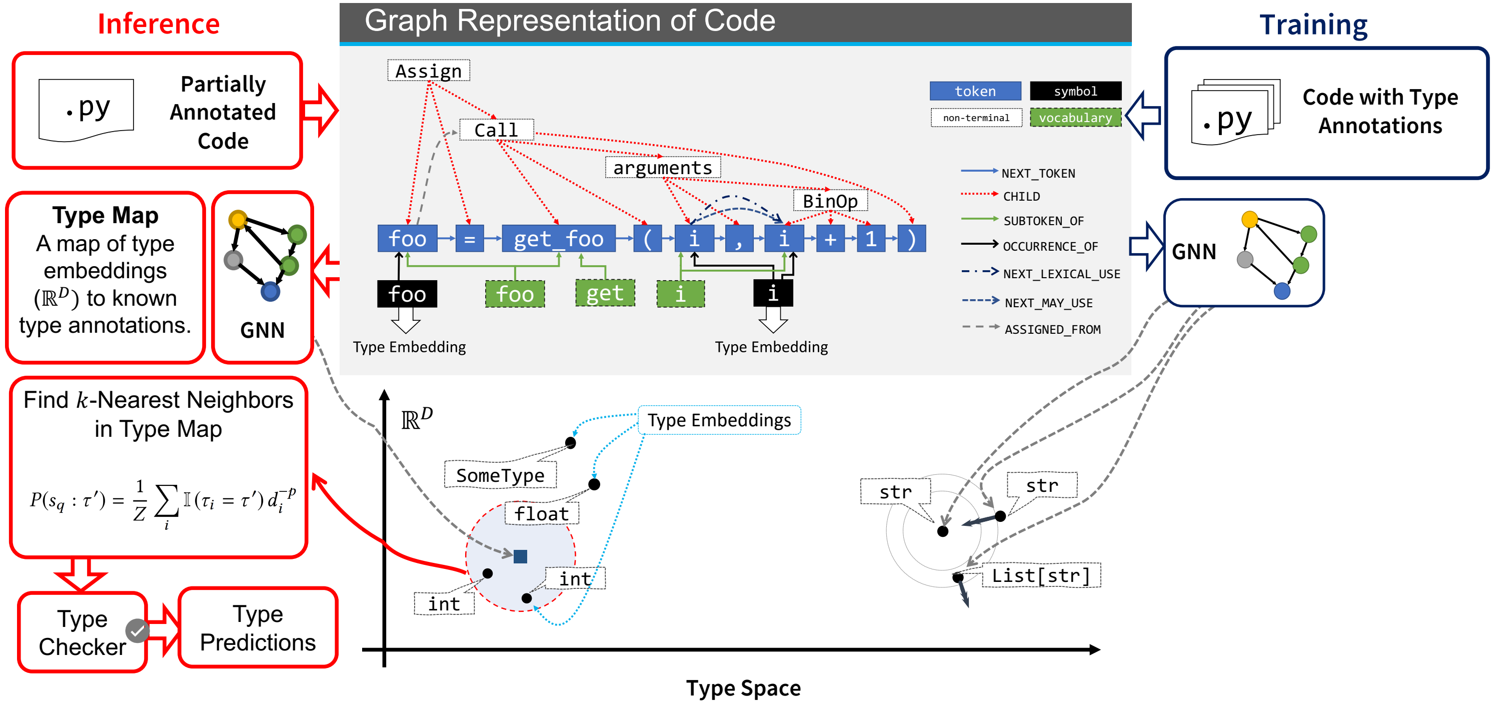


Soon Moon Kang

Aga Kapros

Internships: Typilus

Programming Languages Design and Implementation (PLDI) 2020



Soline Ducousso

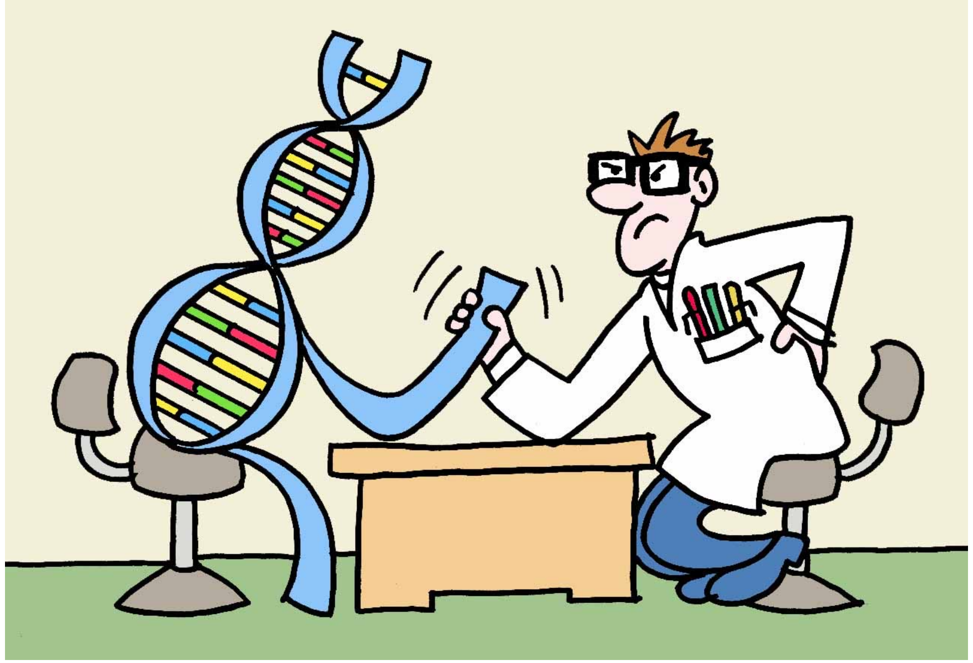
Automatic Software Transplantation

Best Paper, ISSTA 2015

```
QModelIndex start;
if (currentIndex().isValid())
    start = currentIndex();
else
    start = d->model->index(0, 0, d->root);

bool skipRow = false;
bool keyboardTimeWasValid = d->keyboardInputTime.isValid();
qint64 keyboardInputTimeElapsed = d->keyboardInputTime.restart();
if (search.isEmpty() || !keyboardTimeWasValid
    || keyboardInputTimeElapsed > QApplication::keyboardInputInterval()) {
    d->keyboardInput = search;
    skipRow = currentIndex().isValid(); //if it is not valid we should real
} else {
    d->keyboardInput += search;
}
```

Earl T. Barr, UCL

Projects: Software Transplantation



Gold Medal 2016 Hummies

Dr. Alexandru
Marginean

Meta

Tangra: Slaying Immortal Bugs

In a block chain, code is immutable.

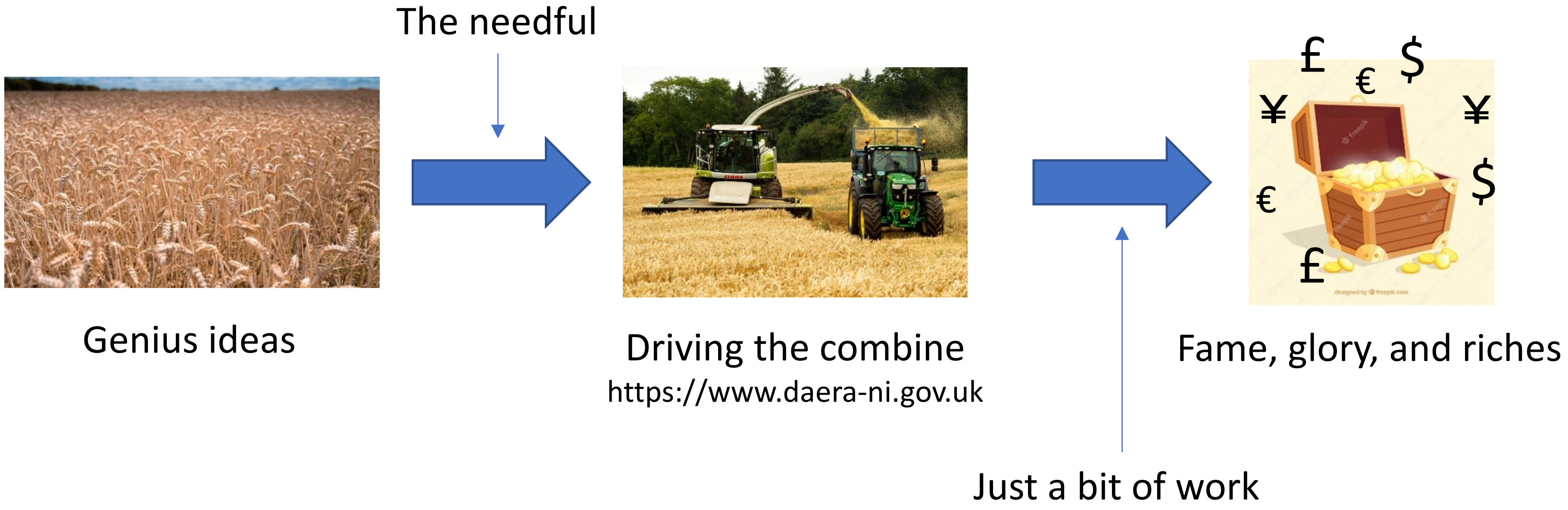
But even Gods can be slain.

Joint work with Zvezdin Besarabov

Earl T. Barr, UCL

Project Ideas

Aka this year's crop of genius idea
and life-transforming opportunities!



The needful

Genius ideas

Driving the combine
https://www.daera-ni.gov.uk

Fame, glory, and riches

Just a bit of work

Earl T. Barr, UCL

Why an Individual Research Project?

- “Individual” is a misnomer.

- Don’t get lost in the crowd!

- I do companies too!

Fame, glory, and riches

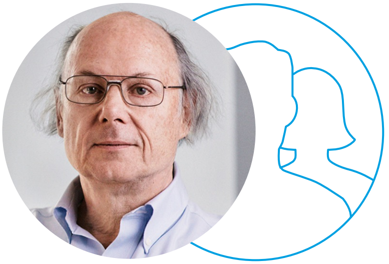
Individual Research Project I with Morgan Stanley

Earl T. Barr, UCL

Technology at Morgan Stanley

11,000+

engineers in technology - largest department in the firm



38

Distinguished Engineers including Bjarne Stroustrup

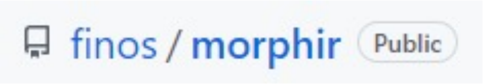
~$4.5B

invested in technology and innovation each year

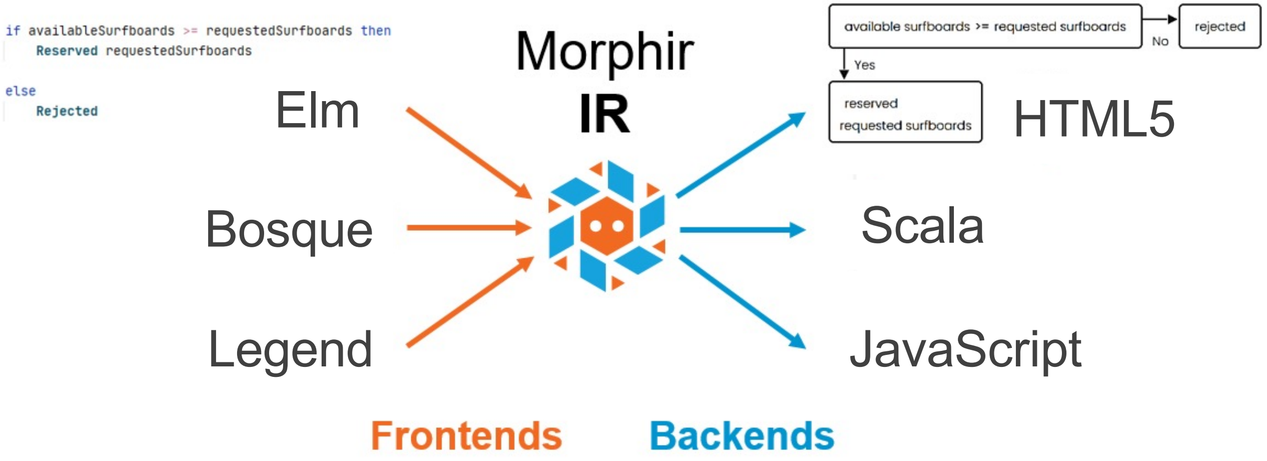
About Morphir

multi-language system built on a data format that captures an application's domain model and business logic in a technology agnostic manner

GitHub



finos / morphir Public



```
if availableSurfboards >= requestedSurfboards then
    Reserved requestedSurfboards

else
    Rejected
```

Morphir
IR

Elm

Bosque

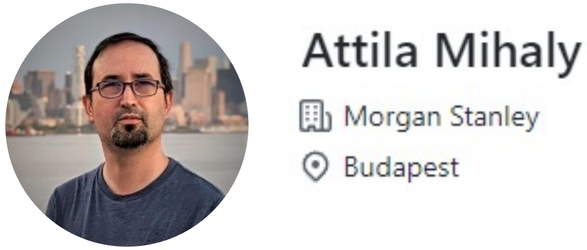
Legend

HTML5

Scala

JavaScript

Frontends **Backends**

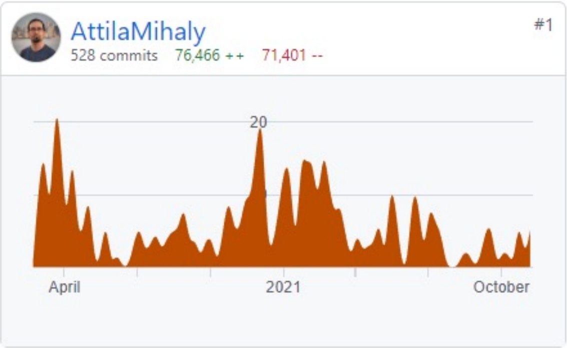


Attila Mihaly

Morgan Stanley

Budapest

- 14 years at Morgan Stanley
- Built multiple trading systems
- Co-created Morphir
- Now maintaining Morphir



Open-source Momentum

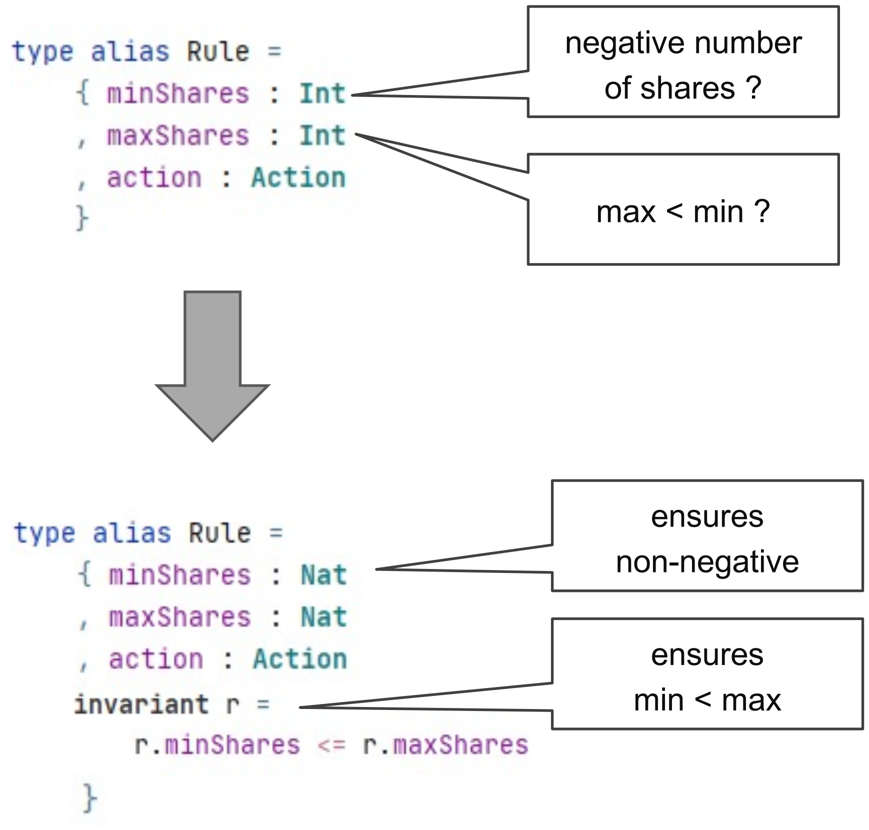
- We want to change the whole approach to developing software in Finance to make it both more efficient and less risky.
- Contributions
  - Goldman Sachs → Legend
  - Morgan Stanley → Morphir
  - Microsoft Research → Bosque

FINOS

Legend → morphir → Bosque

The Project: Add Data Constraints to Morphir

- Goals
  - Extend the Morphir IR with invariants
  - Add more specific types to the SDK (like Nat)
  - Extend type inference to retain invariant information
  - Pass invariants to Bosque checker for verification
- Skills used
  - Functional programming: Elm
    - Designed to get JS programmers hooked on FP
    - Learning Elm ≈ Learning Morphir
  - Type theory



```
type alias Rule =
    { minShares : Int
    , maxShares : Int
    , action : Action
    }
```

negative number of shares ?

max < min ?

```
type alias Rule =
    { minShares : Nat
    , maxShares : Nat
    , action : Action
    invariant r =
        r.minShares <= r.maxShares
    }
```

ensures non-negative

ensures min < max

Individual Research Project II with Bloomberg

Earl T. Barr, UCL

Project Admin

Title: Converting Spreadsheets to Python Functions

Main Supervisor: Prof. Earl Barr

Industry Sponsor: Alex Brisan, Software Engineer, Bloomberg

Background

- Historical dependency on spreadsheets (Excel)

- Complex row/column/cell level formulas

- Using Excel to connect to external systems (databases, Bloomberg API, etc.)

Challenges

- Excel does not scale for big data
- Extremely error-prone/hard to automatically verify
- Lack of transparency into how results are produced
- Suboptimal for collaboration

Industry is moving towards Python

Ruthless automation is the key to success!

Your Project:

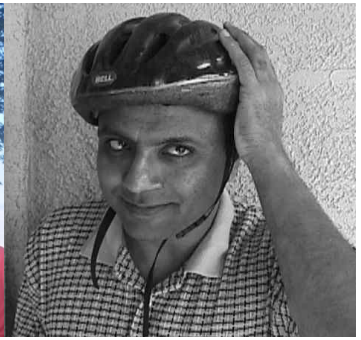
- Research existing solutions, gaps in functionality
- Explore how translation could work for simple formulas; key is to build an extensible framework
- Output would be a tool that can translate simple formulas

Potential for further collaboration beyond the scope of the project

Individual Research Project III with University of California Davis

Earl T. Barr, UCL

UCDAVIS
UNIVERSITY OF CALIFORNIA

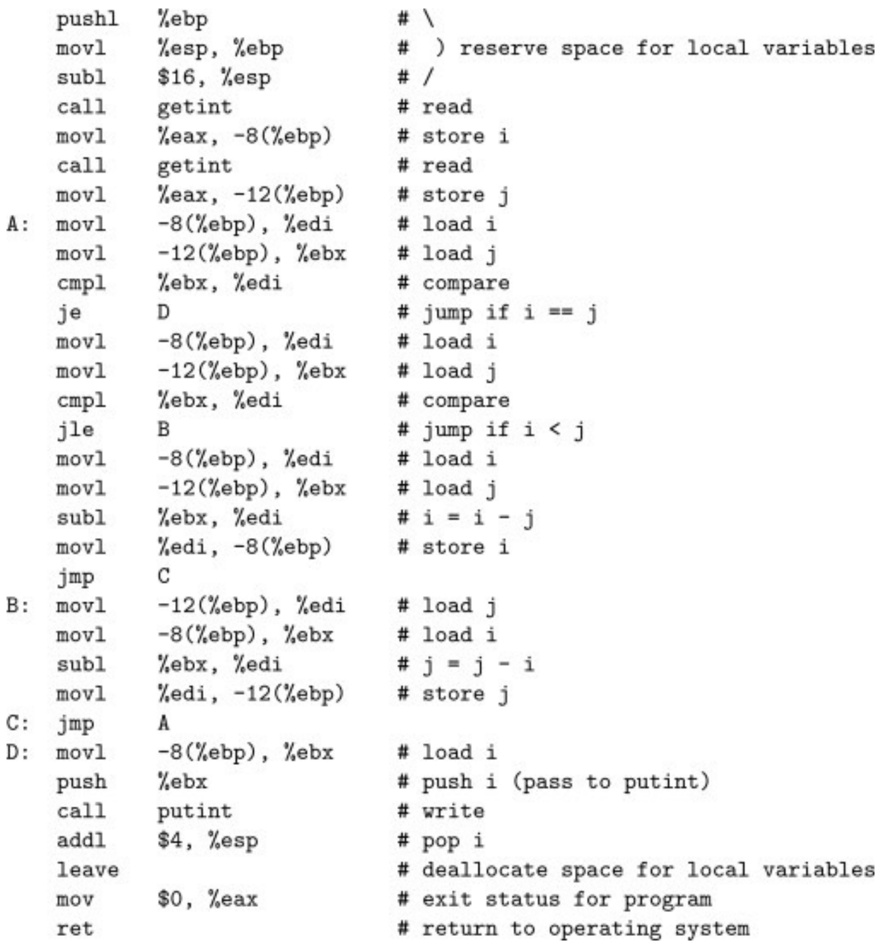


“The Wilderness holds answers to more questions than we have yet learned to ask.“

—Nancy Newhall

A Brief History of Types

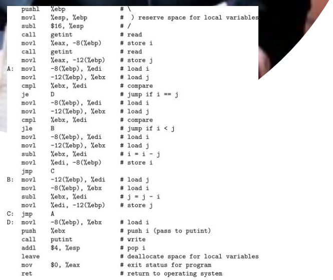
- Assembly



```
    pushl   %ebp                # \
    movl    %esp, %ebp          #  ) reserve space for local variables
    subl    $16, %esp           # /
    call    getint              # read
    movl    %eax, -8(%ebp)      # store i
    call    getint              # read
    movl    %eax, -12(%ebp)     # store j
A:  movl    -8(%ebp), %edi      # load i
    movl    -12(%ebp), %ebx     # load j
    cmpl    %ebx, %edi          # compare
    je      D                   # jump if i == j
    movl    -8(%ebp), %edi      # load i
    movl    -12(%ebp), %ebx     # load j
    cmpl    %ebx, %edi          # compare
    jle     B                   # jump if i < j
    movl    -8(%ebp), %edi      # load i
    movl    -12(%ebp), %ebx     # load j
    subl    %ebx, %edi          # i = i - j
    movl    %edi, -8(%ebp)      # store i
    jmp     C
B:  movl    -12(%ebp), %edi     # load j
    movl    -8(%ebp), %ebx      # load i
    subl    %ebx, %edi          # j = j - i
    movl    %edi, -12(%ebp)     # store j
C:  jmp     A
D:  movl    -8(%ebp), %ebx      # load i
    push    %ebx                # push i (pass to putint)
    call    putint              # write
    addl    $4, %esp            # pop i
    leave                       # deallocate space for local variables
    mov     $0, %eax            # exit status for program
    ret                         # return to operating system
```

A Brief History of Types

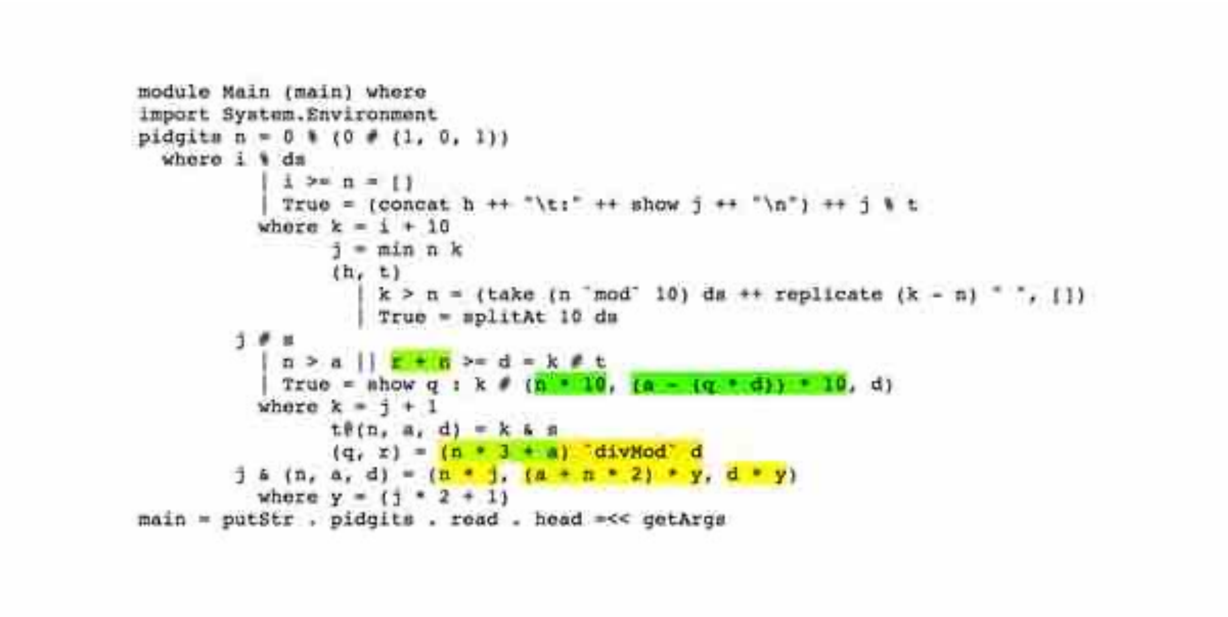
- Assembly



```
    pushl   %ebp               # \
    movl    %esp, %ebp         #  ) reserve space for local variables
    subl    $16, %esp          # /
    call    getint             # read
    movl    %eax, -8(%ebp)     # store i
    call    getint             # read
    movl    %eax, -12(%ebp)    # store j
A:  movl    -8(%ebp), %edi     # load i
    movl    -12(%ebp), %ebx    # load j
    cmpl    %ebx, %edi         # compare
    je      D                  # jump if i == j
    movl    -8(%ebp), %edi     # load i
    movl    -12(%ebp), %ebx    # load j
    cmpl    %ebx, %edi         # compare
    jle     B                  # jump if i < j
    movl    -8(%ebp), %edi     # load i
    movl    -12(%ebp), %ebx    # load j
    subl    %ebx, %edi         # i = i - j
    movl    %edi, -8(%ebp)     # store i
    jmp     C
B:  movl    -12(%ebp), %edi    # load j
    movl    -8(%ebp), %ebx     # load i
    subl    %ebx, %edi         # j = j - i
    movl    %edi, -12(%ebp)    # store j
C:  jmp     A
D:  movl    -8(%ebp), %ebx     # load i
    push    %ebx               # push i (pass to putint)
    call    putint             # write
    addl    $4, %esp           # pop i
    leave                      # deallocate space for local variables
    mov     $0, %eax           # exit status for program
    ret                        # return to operating system
```

A Brief History of Types

- Assembly
- Let’s constrain things: Types → Haskell

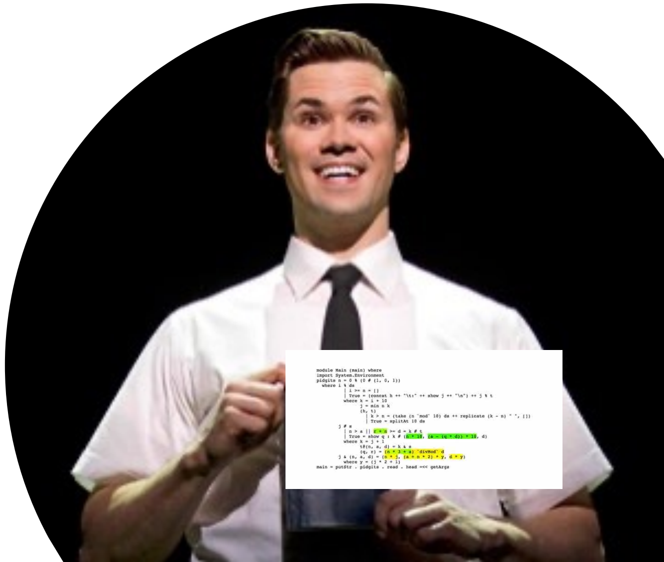


```
module Main (main) where
import System.Environment
pidgits n = 0 % (0 # (1, 0, 1))
  where i % ds
          | i >= n = []
          | True = (concat h ++ "\t:" ++ show j ++ "\n") ++ j % t
          where k = i + 10
                j = min n k
                (h, t)
                  | k > n = (take (n `mod` 10) ds ++ replicate (k - n) " ", [])
                  | True = splitAt 10 ds
        j # s
          | n > a || r + n >= d = k # t
          | True = show q : k # (n * 10, (a - (q * d)) * 10, d)
          where k = j + 1
                t@(n, a, d) = k & s
                (q, r) = (n * 3 + a) `divMod` d
        j & (n, a, d) = (n * j, (a + n * 2) * y, d * y)
          where y = (j * 2 + 1)
main = putStr . pidgits . read . head =<< getArgs
```

A Haskellite

A Brief History of Types

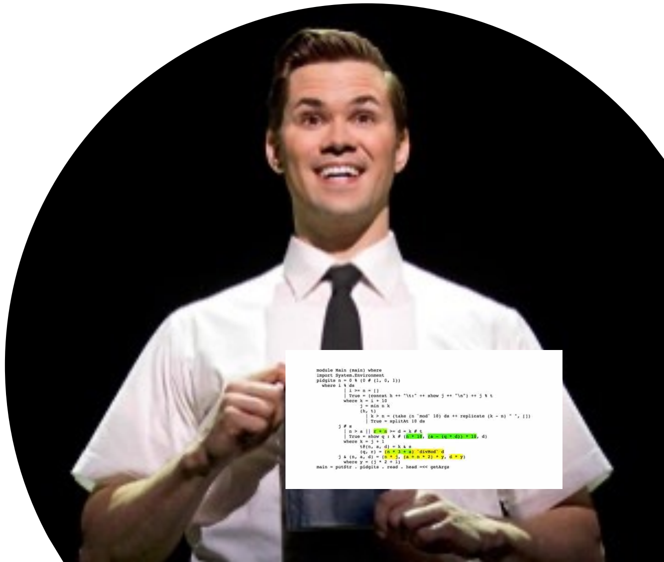
- Assembly
- Let’s constrain things: Types → Haskell



```
module Main (main) where
import System.Environment
pidgits n = 0 % (0 # (1, 0, 1))
  where i % ds
          | i >= n = []
          | True = (concat h ++ "\t:" ++ show j ++ "\n") ++ j % t
          where k = i + 10
                j = min n k
                (h, t)
                  | k > n = (take (n `mod` 10) ds ++ replicate (k - n) " ", [])
                  | True = splitAt 10 ds
        j # s
          | n > a || r + n >= d = k # t
          | True = show q : k # (n * 10, (a - (q * d)) * 10, d)
          where k = j + 1
                t@(n, a, d) = k & s
                (q, r) = (n * 3 + a) `divMod` d
        j & (n, a, d) = (n * j, (a + n * 2) * y, d * y)
          where y = (j * 2 + 1)
main = putStr . pidgits . read . head =<< getArgs
```

A Brief History of Types

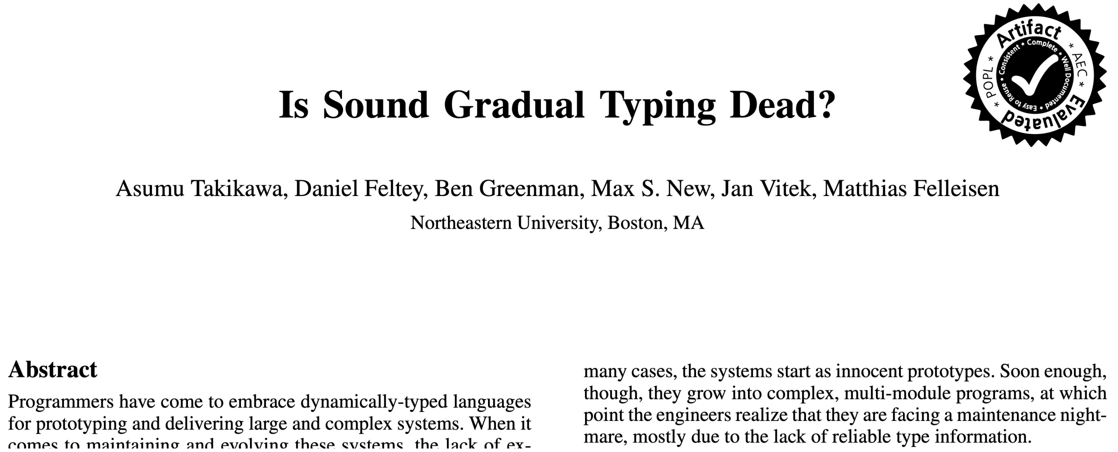
- Assembly
- Let’s constrain things: Types → Haskell
- Gradual typing: purity rituals performed at runtime
    - Gradual guarantee



```
module Main (main) where
import System.Environment
pidgits n = 0 % (0 # (1, 0, 1))
  where i % ds
          | i >= n = []
          | True = (concat h ++ "\t:" ++ show j ++ "\n") ++ j % t
          where k = i + 10
                j = min n k
                (h, t)
                  | k > n = (take (n `mod` 10) ds ++ replicate (k - n) " ", [])
                  | True = splitAt 10 ds
        j # s
          | n > a || r + n >= d = k # t
          | True = show q : k # (n * 10, (a - (q * d)) * 10, d)
          where k = j + 1
                t@(n, a, d) = k & s
                (q, r) = (n * 3 + a) `divMod` d
        j & (n, a, d) = (n * j, (a + n * 2) * y, d * y)
          where y = (j * 2 + 1)
main = putStr . pidgits . read . head =<< getArgs
```

A Brief History of Types

- Assembly
- Let’s constrain things: Types → Haskell
- Gradual typing: purity rituals performed at runtime
  - Gradual guarantee
- Gradual typing is dead



Is Sound Gradual Typing Dead?

Asumu Takikawa, Daniel Feltey, Ben Greenman, Max S. New, Jan Vitek, Matthias Felleisen

Northeastern University, Boston, MA

Abstract

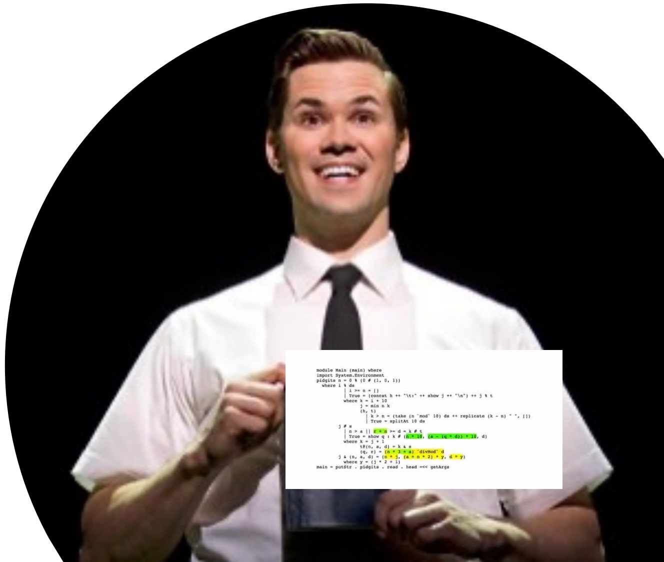
Programmers have come to embrace dynamically-typed languages for prototyping and delivering large and complex systems. When it comes to maintaining and evolving these systems, the lack of ex-

many cases, the systems start as innocent prototypes. Soon enough, though, they grow into complex, multi-module programs, at which point the engineers realize that they are facing a maintenance nightmare, mostly due to the lack of reliable type information.

A Brief History of Types

- Assembly
- Let’s constrain things: Types → Haskell
- Gradual typing: purity rituals performed at runtime
  - Gradual guarantee
- Gradual typing is dead
  - Optimisation
  - And…

```
    pushl   %ebp                # \
    movl    %esp, %ebp          #  ) reserve space for local variables
    subl    $16, %esp           # /
    call    getint              # read
    movl    %eax, -8(%ebp)      # store i
    call    getint              # read
    movl    %eax, -12(%ebp)     # store j
A:  movl    -8(%ebp), %edi      # load i
    movl    -12(%ebp), %ebx     # load j
    cmpl    %ebx, %edi          # compare
    je      D                   # jump if i == j
    movl    -8(%ebp), %edi      # load i
    movl    -12(%ebp), %ebx     # load j
    cmpl    %ebx, %edi          # compare
    jle     B                   # jump if i < j
    movl    -8(%ebp), %edi      # load i
    movl    -12(%ebp), %ebx     # load j
    subl    %ebx, %edi          # i = i - j
    movl    %edi, -8(%ebp)      # store i
    jmp     C
B:  movl    -12(%ebp), %edi     # load j
    movl    -8(%ebp), %ebx      # load i
    subl    %ebx, %edi          # j = j - i
    movl    %edi, -12(%ebp)     # store j
C:  jmp     A
D:  movl    -8(%ebp), %ebx      # load i
    push    %ebx                # push i (pass to putint)
    call    putint              # write
    addl    $4, %esp            # pop i
    leave                       # deallocate space for local variables
    mov     $0, %eax            # exit status for program
    ret                         # return to operating system
```



```
module Main (main) where
import System.Environment
pidgits n = 0 % (0 # (1, 0, 1))
  where i % ds
          | i >= n = []
          | True = (concat h ++ "\t:" ++ show j ++ "\n") ++ j % t
         where k = i + 10
               j = min n k
               (h, t)
                 | k > n = (take (n `mod` 10) ds ++ replicate (k - n) " ", [])
                 | True = splitAt 10 ds
        j # s
          | n > a || r + n >= d = k # t
          | True = show q : k # (n * 10, (a - (q * d)) * 10, d)
         where k = j + 1
               t@(n, a, d) = k & s
               (q, r) = (n * 3 + a) `divMod` d
        j & (n, a, d) = (n * j, (a + n * 2) * y, d * y)
         where y = (j * 2 + 1)
main = putStr . pidgits . read . head =<< getArgs
```

The Optional Type Revolution

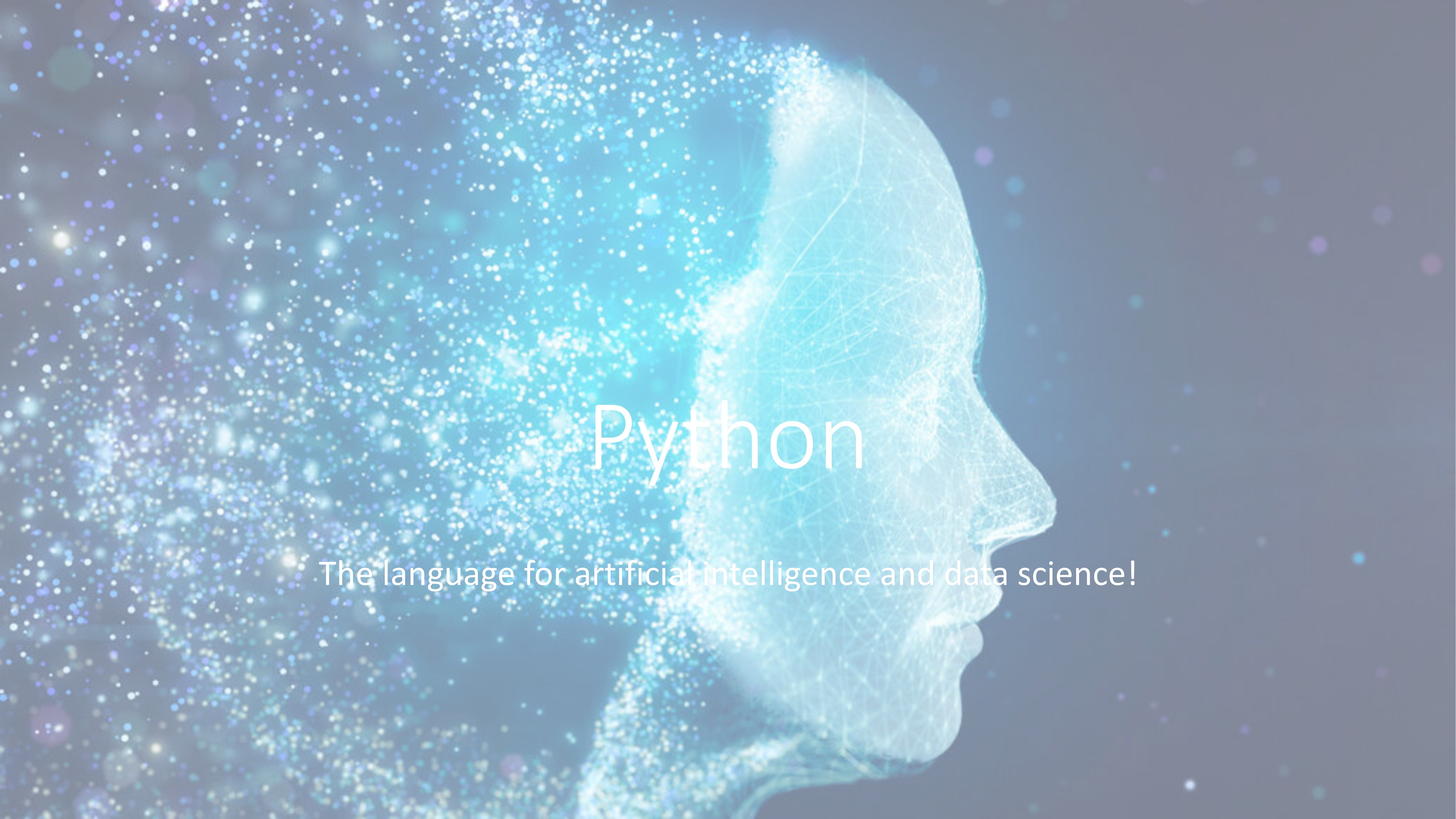
Optional type inference checks types only locally and statically; it has no runtime footprint

+ Fast

+ Finds local inconsistencies,

+ Supports navigation and completion

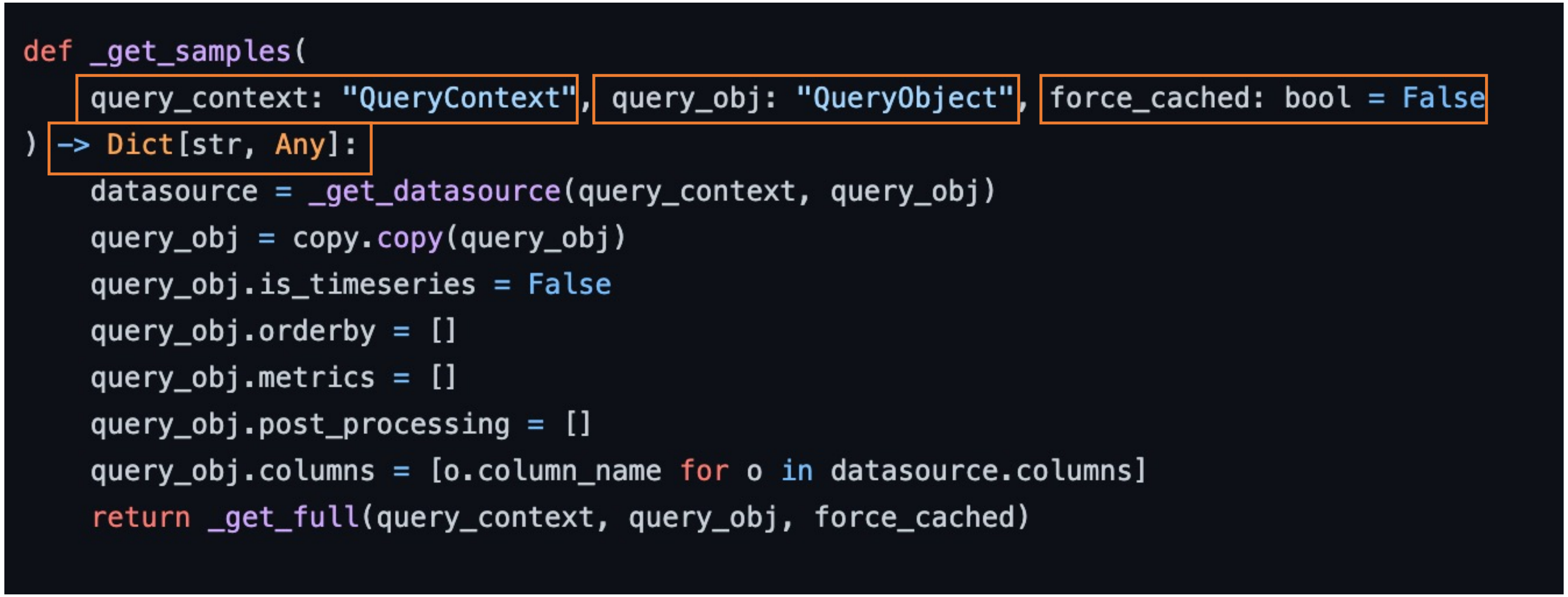
- Partial guarantees; can give a false sense of security



Python

The language for artificial intelligence and data science!

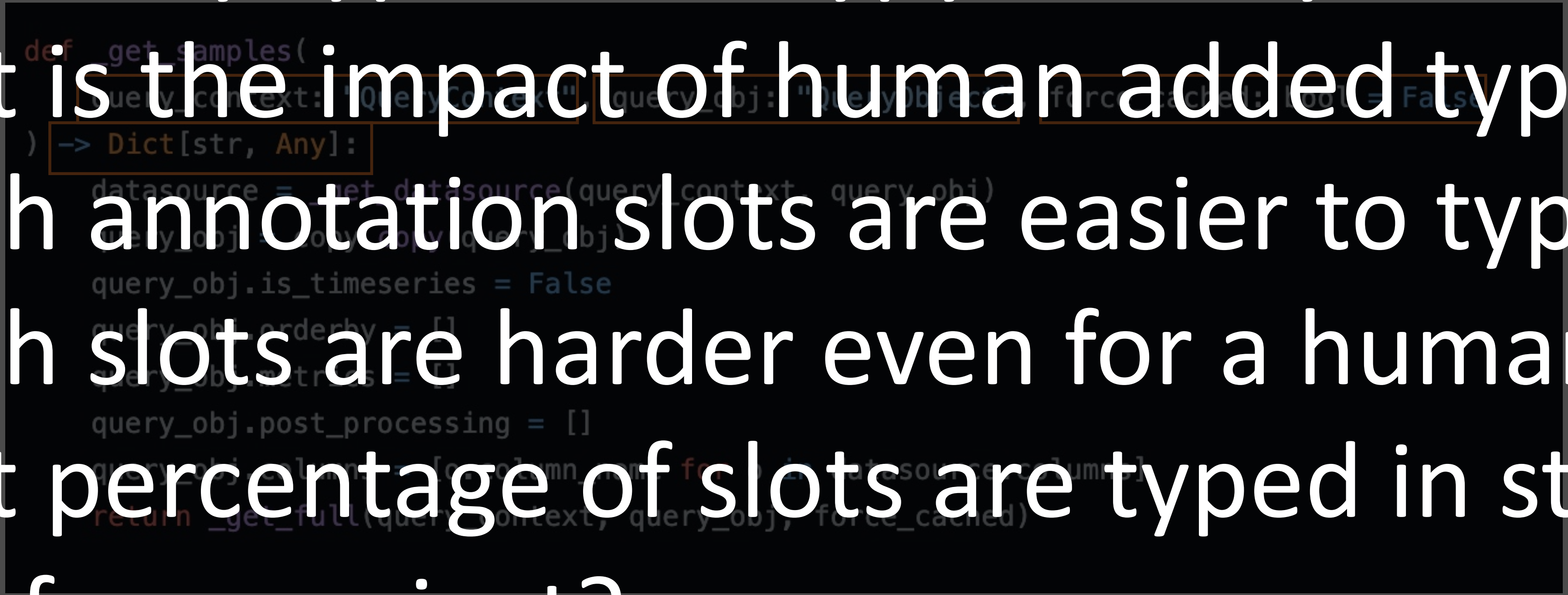
Natural Experiment on Optional Types



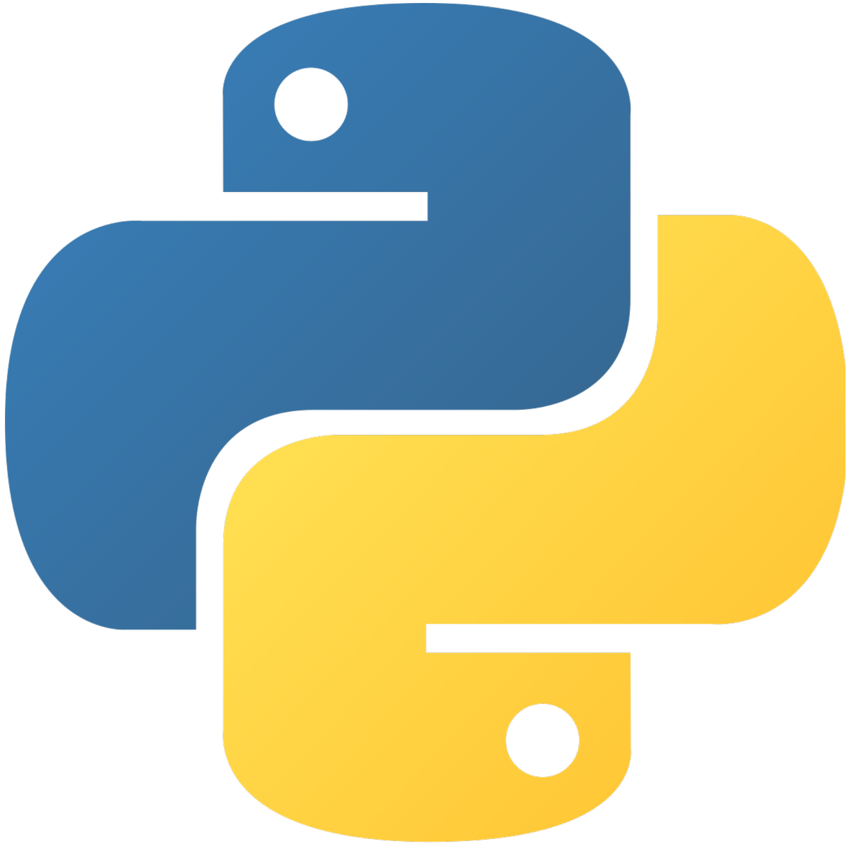
```
def _get_samples(
    query_context: "QueryContext", query_obj: "QueryObject", force_cached: bool = False
) -> Dict[str, Any]:
    datasource = _get_datasource(query_context, query_obj)
    query_obj = copy.copy(query_obj)
    query_obj.is_timeseries = False
    query_obj.orderby = []
    query_obj.metrics = []
    query_obj.post_processing = []
    query_obj.columns = [o.column_name for o in datasource.columns]
    return _get_full(query_context, query_obj, force_cached)
```

Natural Experiment on Optional Types

- How many types can mypy trivially infer?
- What is the impact of human added types?
- Which annotation slots are easier to type?
- Which slots are harder even for a human?
- What percentage of slots are typed in steady state for a project?



```
def _get_samples(
    query_context: "QueryContext", query_obj: "QueryObject", force_cached: bool = False
) -> Dict[str, Any]:
    datasource = _get_datasource(query_context, query_obj)
    query_obj = copy.copy(query_obj)
    query_obj.is_timeseries = False
    query_obj.orderby = []
    query_obj.metrics = []
    query_obj.post_processing = []
    query_obj.columns = [o.column_name for o in datasource.columns]
    return _get_full(query_context, query_obj, force_cached)
```



: my[py]

Into the Wild: Type Usage in Python

Task

- Data analytics and analysis scripts

Skills

- Python, Continuous Integration, git, Basic statistics
- R is a nice to have (if you don't know it, you'll learn it)

Wins

- A chance to shape the evolution of Python
- $$$ A research paper $$$

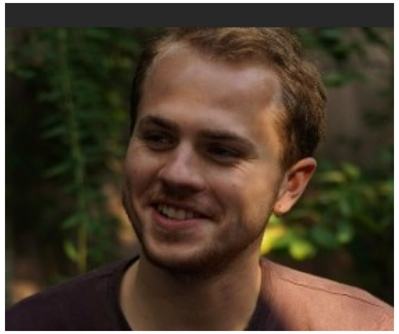
Individual Research Project IV with Y Combinator Startup Bloop

Earl T. Barr, UCL

bloop.

https://bloop.ai/

Y Combinator



Gabriel, founder and CTO

It should be easier for developers to find and share code

So we’re building a search engine

Small team of ex-Huawei, Imperial and Yale ML and compiler engineers

Backed by some of the top VCs, including Khosla Ventures and Y Combinator

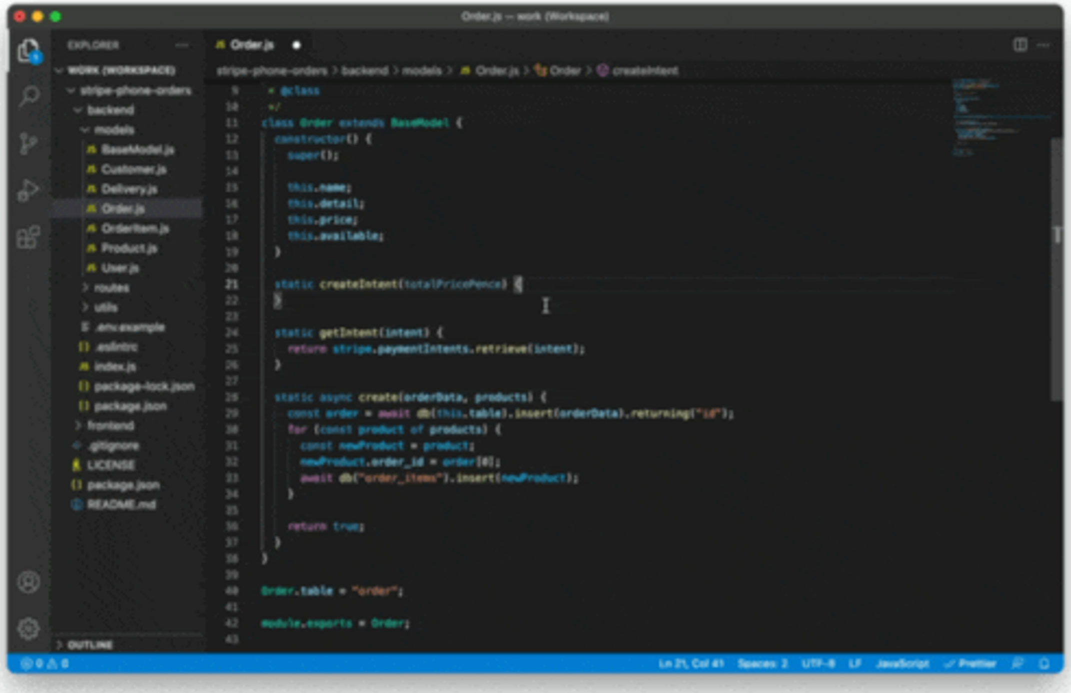
bloop.

IDE-based search engine

Retrieves relevant snippets of code from open-source

Code contextual or natural language search

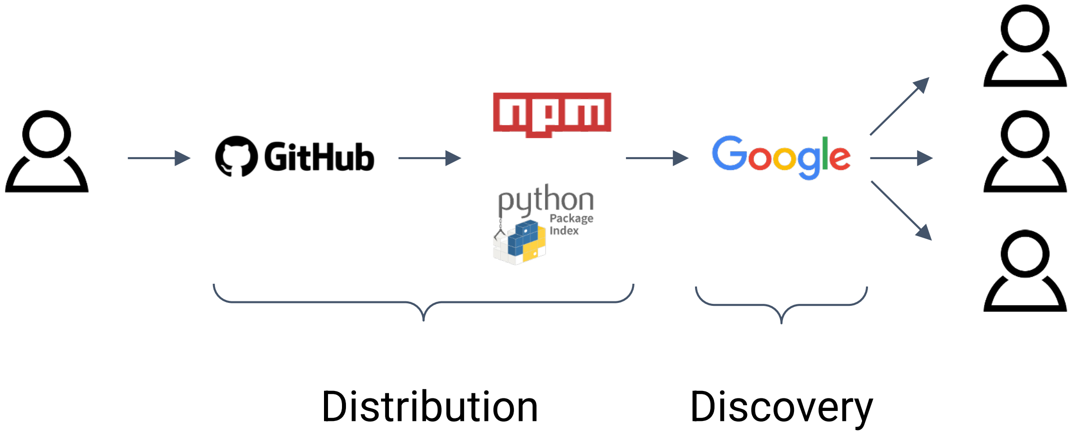
Search is powered by NLP and syntax analysis



```
 * @class
 */
class Order extends BaseModel {
  constructor() {
    super();

    this.name;
    this.detail;
    this.price;
    this.available;
  }

  static createIntent(totalPricePence) {
  }

  static getIntent(intent) {
    return stripe.paymentIntents.retrieve(intent);
  }

  static async create(orderData, products) {
    const order = await db(this.table).insert(orderData).returning("id");
    for (const product of products) {
      const newProduct = product;
      newProduct.order_id = order[0];
      await db("order_items").insert(newProduct);
    }

    return true;
  }
}

Order.table = "order";

module.exports = Order;
```

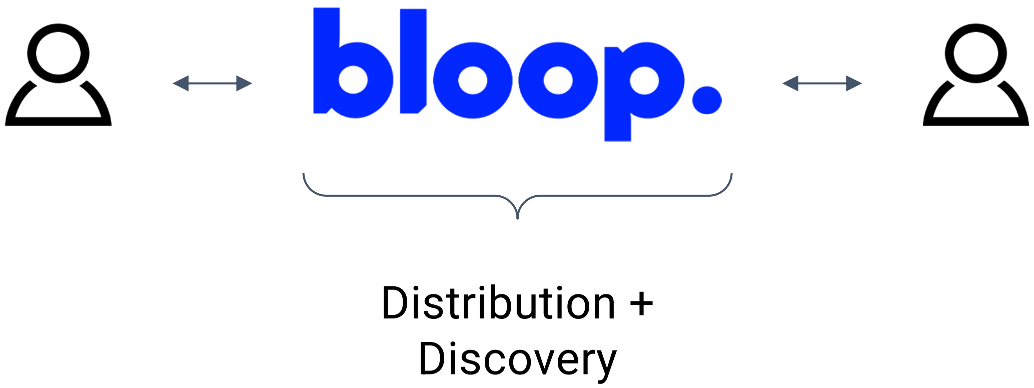
Sharing code today

(lots of friction)



Sharing code tomorrow

(no friction)



The Project: Build a Smart Copy-Paste System

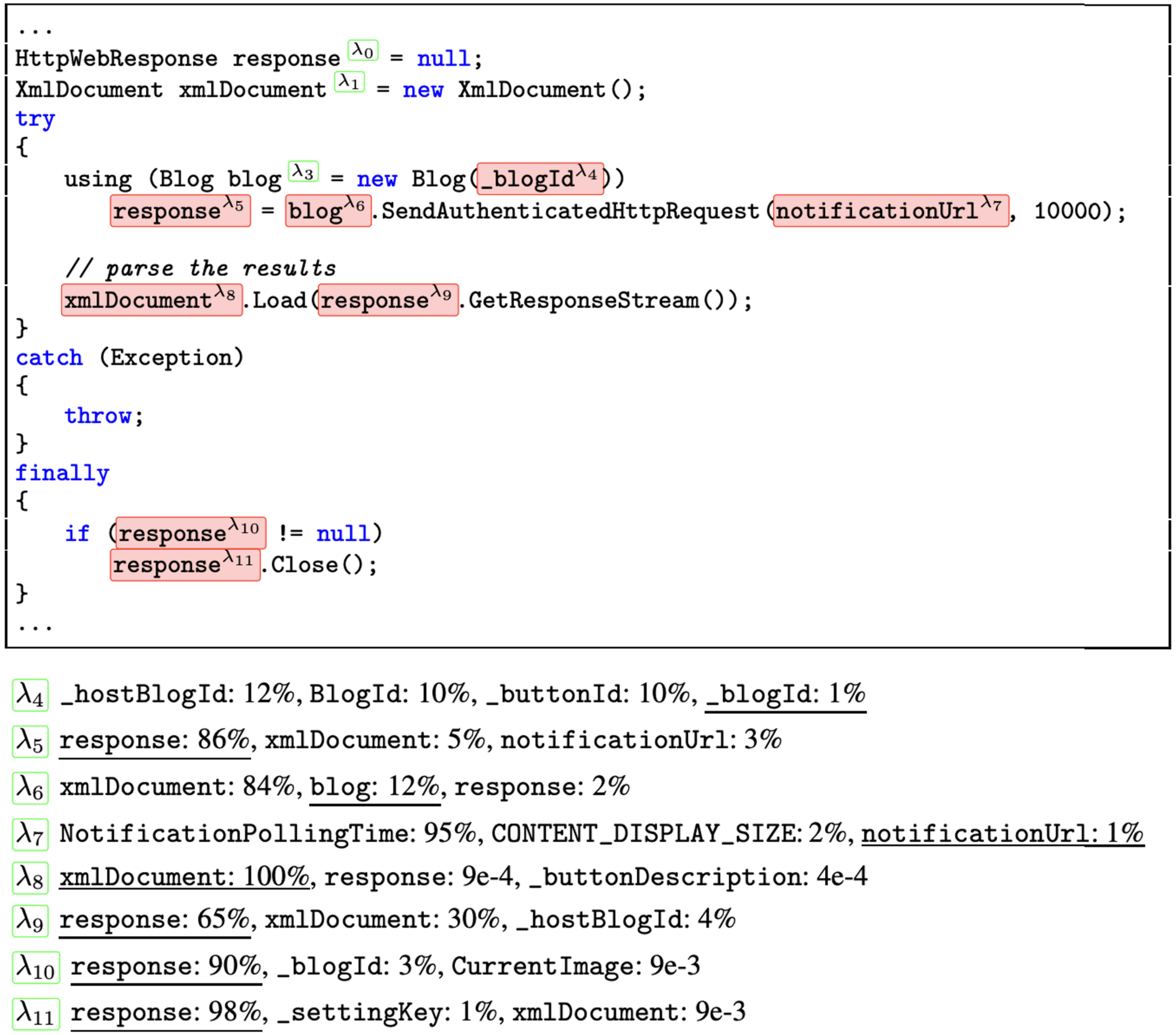
When users copy code from bloop they have to manually edit variable names to match their project

We want to automate this, renaming free variables with variables in-scope in the user's code

For each free variable find all valid replacements and predict the most likely

Skills:

- Static analysis
- Machine learning



```
...
HttpWebResponse response λ0 = null;
XmlDocument xmlDocument λ1 = new XmlDocument();
try
{
    using (Blog blog λ3 = new Blog(_blogId λ4))
        response λ5 = blog λ6.SendAuthenticatedHttpRequest(notificationUrl λ7, 10000);

    // parse the results
    xmlDocument λ8.Load(response λ9.GetResponseStream());
}
catch (Exception)
{
    throw;
}
finally
{
    if (response λ10 != null)
        response λ11.Close();
}
...
```

λ_4 _hostBlogId: 12%, BlogId: 10%, _buttonId: 10%, _blogId: 1%

λ_5 response: 86%, xmlDocument: 5%, notificationUrl: 3%

λ_6 xmlDocument: 84%, blog: 12%, response: 2%

λ_7 NotificationPollingTime: 95%, CONTENT_DISPLAY_SIZE: 2%, notificationUrl: 1%

λ_8 xmlDocument: 100%, response: 9e-4, _buttonDescription: 4e-4

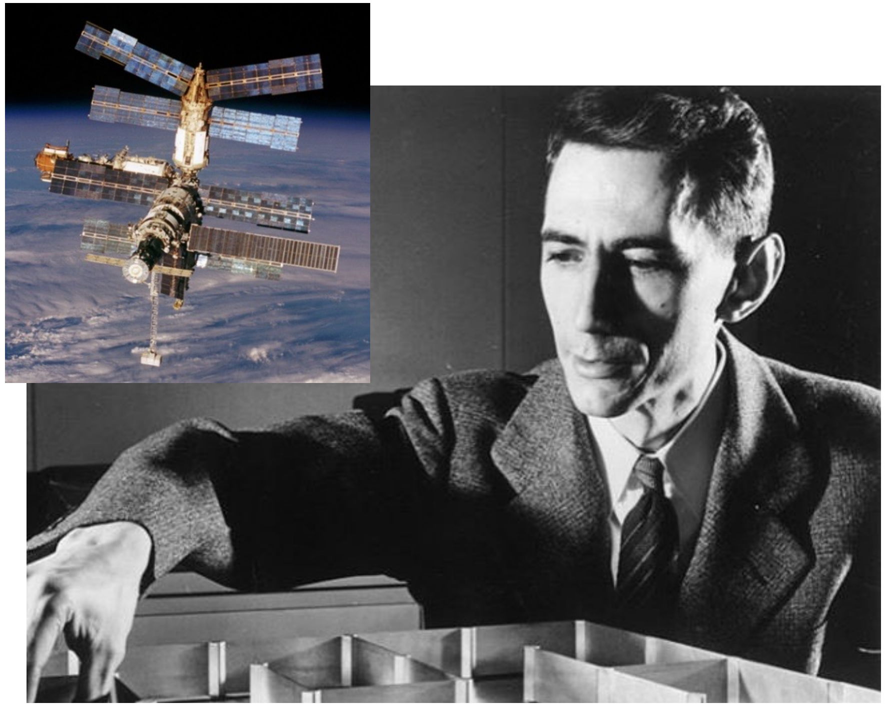
λ_9 response: 65%, xmlDocument: 30%, _hostBlogId: 4%

λ_{10} response: 90%, _blogId: 3%, CurrentImage: 9e-3

λ_{11} response: 98%, _settingKey: 1%, xmlDocument: 9e-3

Individual Research Project V at World Renowned UCL

A Pair of Information Theory Projects



$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}\left(x_i\right) \log_b \mathrm{P}\left(x_i\right)$$

David Kelly, UCL

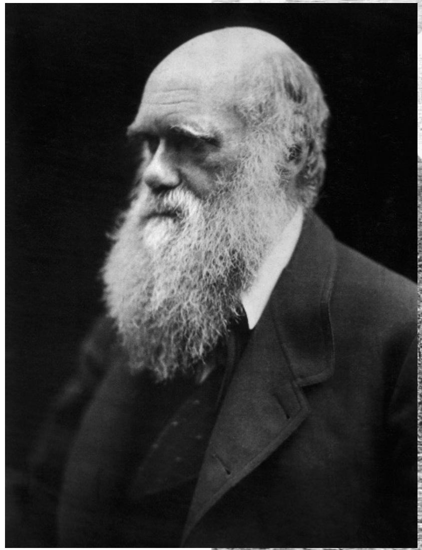
A Pair of Information Theory Projects

Sometimes, bird watching pays off

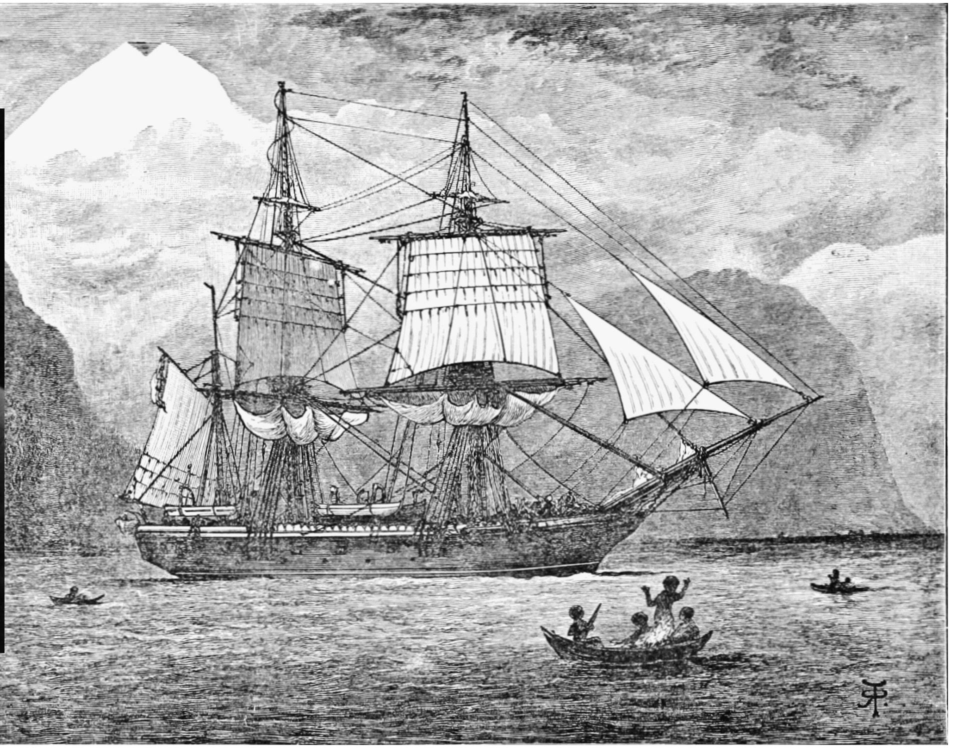
Britain's National Pastime in the 21st Century:
It's no longer about birds, it's all about flows!

Earl T. Barr, UCL

A Pair of Information Theory Projects



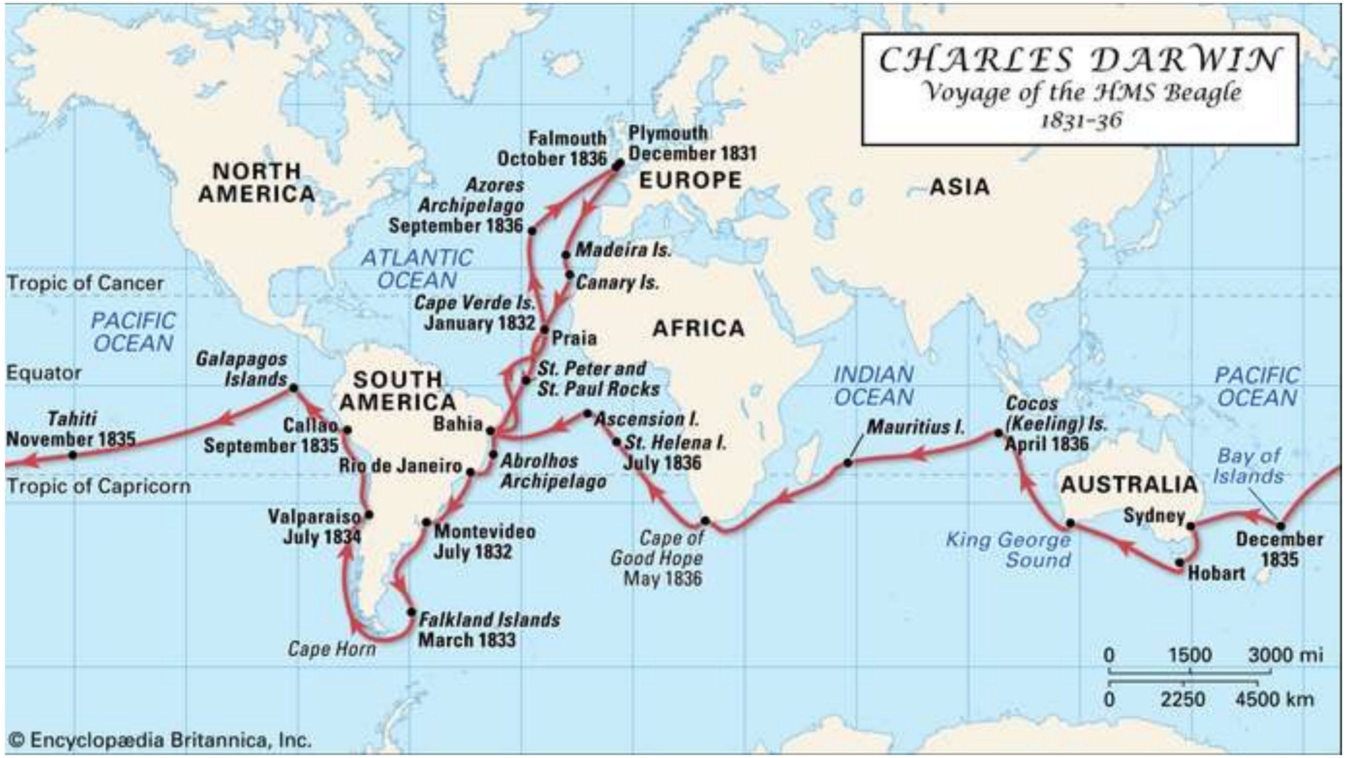
Julia Margaret Cameron, Public domain, via Wikimedia Commons



R. T. Pritchett, Public domain, via Wikimedia Commons

HMS *Beagle* in the Straits of Magellan at Monte Sarmiento, reproduction of R. T. Pritchett's frontispiece from the 1890 illustrated edition of *The Voyage of the Beagle*.

A Pair of Information Theory Projects

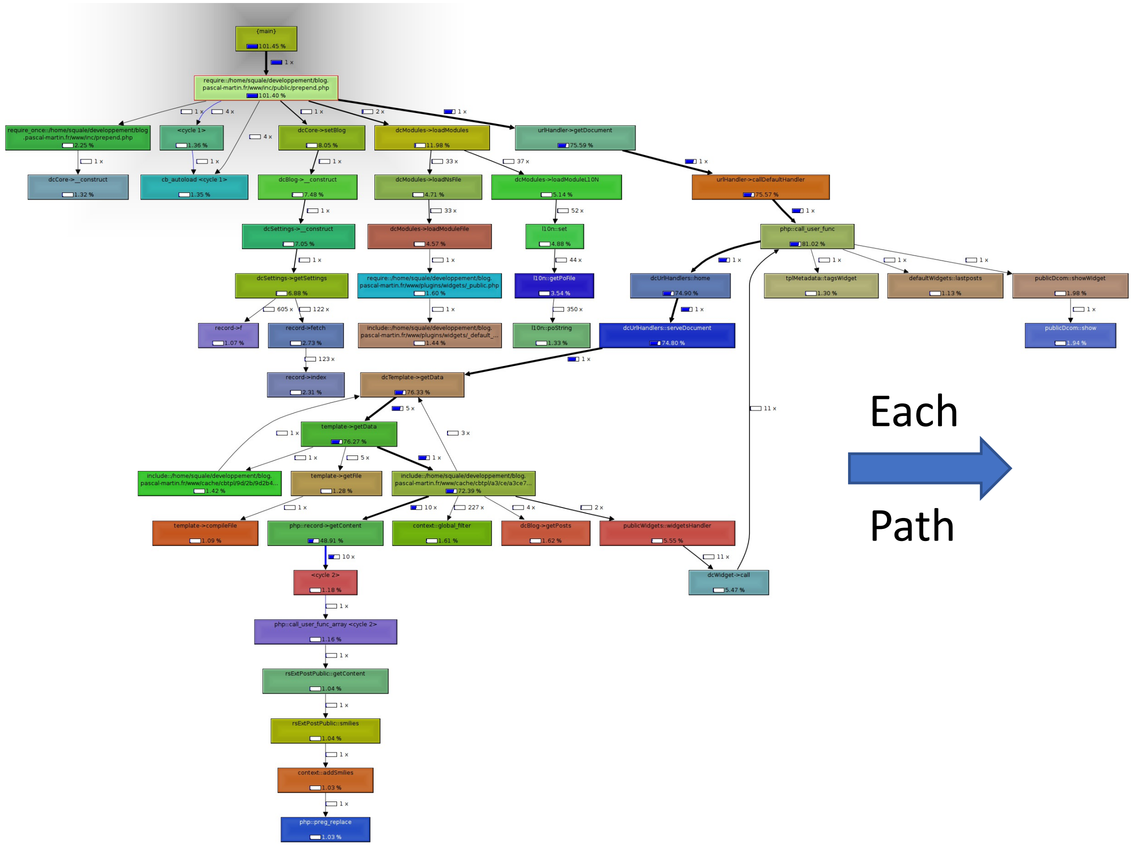


Charles Darwin: HMS *Beagle* voyage

A map of Charles Darwin's voyage on the HMS *Beagle* in 1831–36.

Encyclopædia Britannica, Inc.

What Birds Are We After?



(source: pascal-martin.fr)

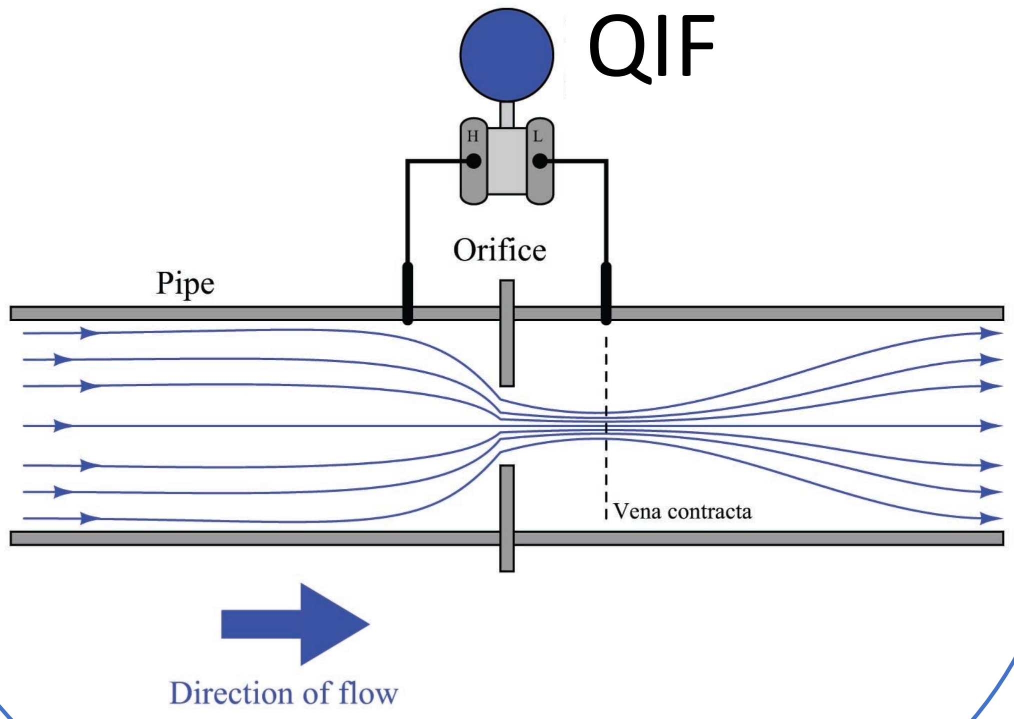
Each

Path

Quantified Information Flow

QIF

I



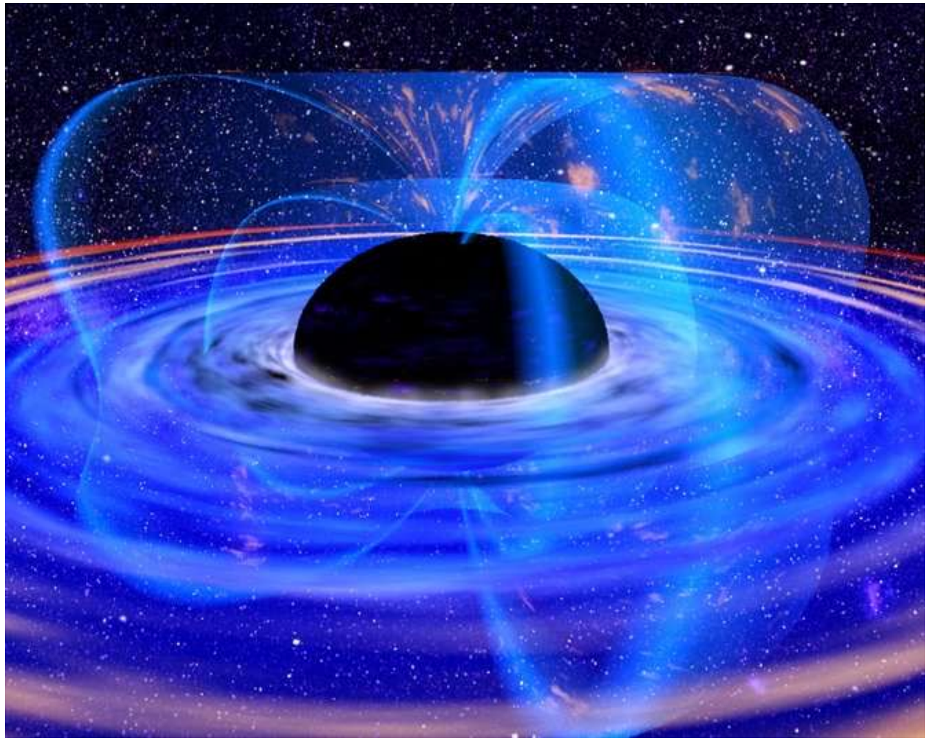
O

Random Variables

QIF and the End of the Universe

Problem: QIF is expensive!

Computing QIF can mean waiting to see how the universe ends



https://phys.org/news/2015-09-fate-universeheat-death-big-rip.html

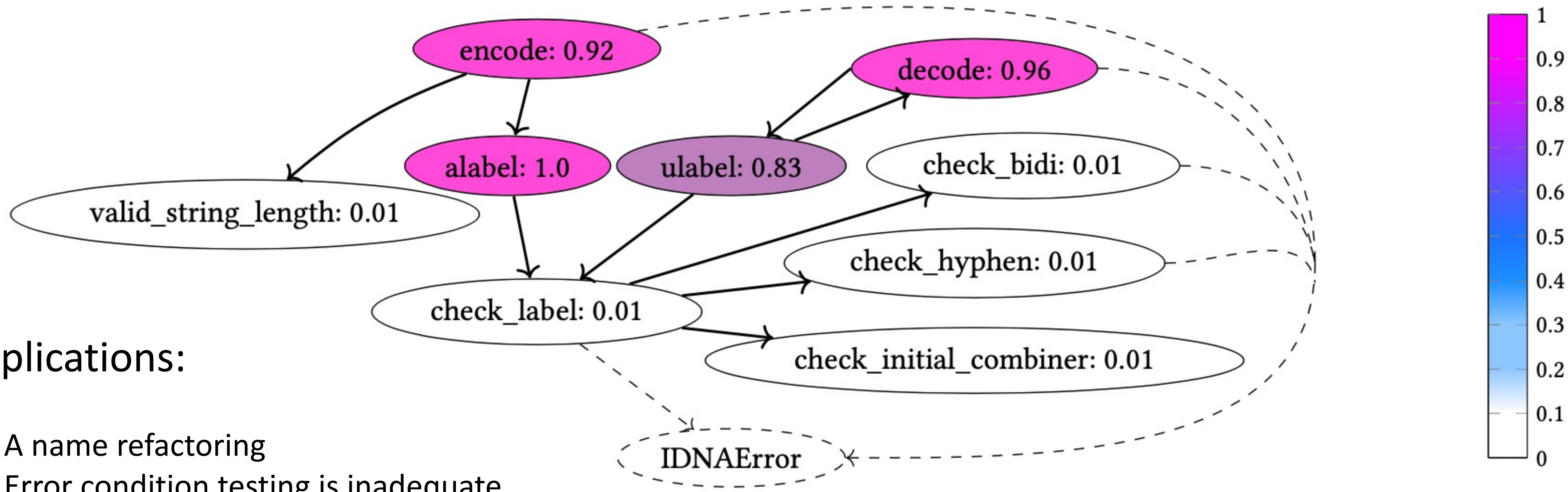
RIF Don’t Quantify

The solution is David Kelly’s PhD work on ranked information flow (RIF).

RIF ranks flows, it doesn’t measure them.

Steve Banks, CC BY-SA 4.0
<https://creativecommons.org/licenses/by-sa/4.0>,
via Wikimedia Commons

What Birds Are We After?



Implications:

(1) A name refactoring
(2) Error condition testing is inadequate
(3) a potential mismatch between function names — encode and decode — and their flows

IT Project I: Information Theoretic Surveying

The project will empirically map flows to programming constructs and idioms.

Open Problems Tackled

- Identifying paths that will be frequent in deployment
- Identifying core logic paths

Wins

- Focused optimization, testing or review

Tasks & Skills Needed: Python, basic statistics, imagination!

IT Project II: Finding Information Leaks

```
hbtype = *p++;
n2s(p, payload);
pl = p;
```

Heartbleed leaks about 14 bits of information, larger than the average password.

Solution 1

Write correct programs

But *“to error is human”*

Solution 2

Automatically detect and correct

But this is undecidable

IT Project II: Finding Information Leaks

```
hbtype = *p++;
n2s(p, payload);
pl = p;
```

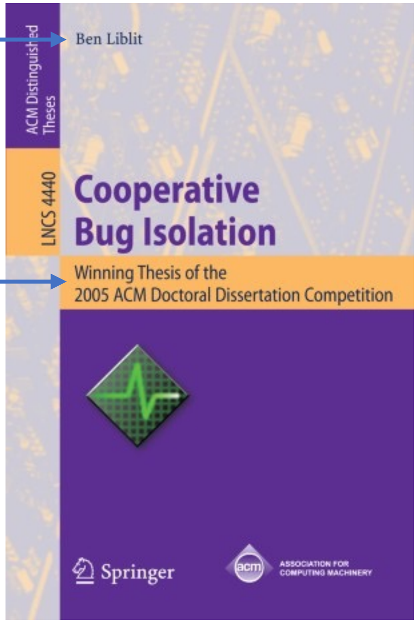
Heartbleed leaks about 14 bits of information, larger than the average password.

Solution 3

Detect and manually correct anomalies

Challenge: Efficient Observation

Ben Liblit



Ben Liblit

ACM Distinguished Theses

LNCS 4440

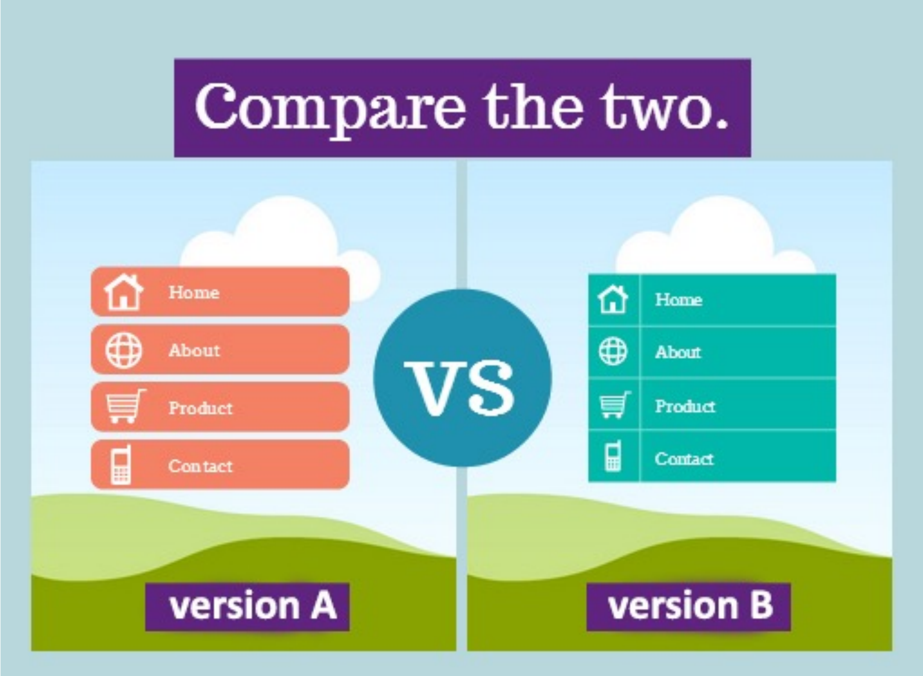
Cooperative Bug Isolation

Winning Thesis of the 2005 ACM Doctoral Dissertation Competition

Springer

ASSOCIATION FOR COMPUTING MACHINERY

+



A/B Testing: Dynamically give different versions to your customer base to test the effect of changes.

Solution: Use existing A/B frameworks to create CBI variants.

IT Project II: Brass Tacks

Tasks

- Identify a suitable web app
- Instrument a web app
- Mock up server for “realistic” testing

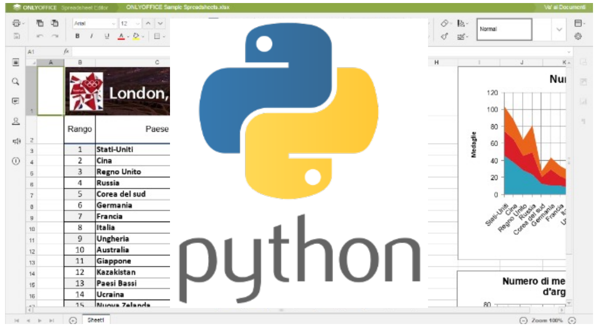
Skills: Python, basic stats, and at least as much imagination as any of the other projects

Wins: $$$ More secure software at little cost to the enduser $$$

Project Ideas

Morphir
IR

Attila Mihaly, Morgan Stanley



Rango	Paese
1	Stati-Uniti
2	Cina
3	Regno Unito
4	Russia
5	Corea del sud
6	Germania
7	Francia
8	Italia
9	Ungheria
10	Australia
11	Giappone
12	Kazakistan
13	Paesi Bassi
14	Ucraina
15	Nuova Zelanda

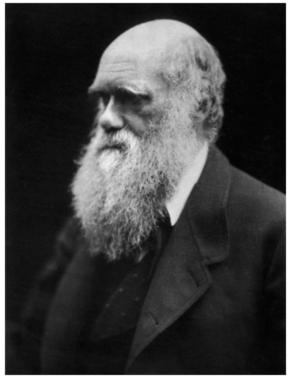
Alex Brisan, Bloomberg

Anand and Prem, UCD

bloop.

https://bloop.ai/

Gabriel Gordon-Hall, Founder



IT Survey

RIF-ing for Leaks

David Kelly, UCL

Join me and together we'll plunder the Dragon's Den!

Earl T. Barr, UCL