Emergent, Crowd-scale Programming Practice in the IDE

Ethan Fast, Daniel Steffee, Lucy Wang, Michael Bernstein, Joel Brandt
Stanford HCI, Adobe Research
Emergent behaviors, or the ways people adapt to a system, can be just as informative as a system’s design.
Many norms for programming systems aren’t codified in documentation or on the web.
Developers can have unanswered questions

What is the **best idiom** or library to use for a certain kind of task?

Does my code follow **common practice**?

How is a language being used **today**?
A Ruby Idiom

How does this code work? What is the block doing?
A Ruby Idiom

How does this code work? What is the block doing?

Extracting an options hash from a function that takes any number of arguments

```ruby
def complicated_function(*args)
  some_var =
  if Hash === args.last
    args.pop
  else
    {}
  end
  # The function does some
  # other important things...
  ret_result
end
```
Codex is a knowledge base that records emergent practice for the Ruby programming language.
Codex normalizes code structure to identify common functions, blocks, and syntactic patterns.
Codex enables new data-driven interfaces for programming

- Detect unlikely code
- Annotate common idioms
- Create a living library
Part 1: Building the Knowledge Base

Building the Codex Knowledge Base
The goal: identify emergent patterns that good programmers would use
Each record in the Codex knowledge base is an AST node.
Each record in the Codex knowledge base is an AST node

Are these snippets equivalent?

```
novels.map { |title| title.downcase + "!" }
movies.map { |name| name.downcase + "?" }
```
Part 1: Building the Knowledge Base

# Snippet 1
uist_hash = Hash.new do |hash, key|
  hash[key] = {}
end
my_hash[:UIST][“2014”] = “Hawaii”

# Snippet 2
chi_hash = Hash.new do |h, k|
  h[k] = {}
end
chi_hash[:CHI][“2014”] = “Toronto”
Part 1: Building the Knowledge Base

# Snippet 1
```
uiist_hash = Hash.new do |hash, key|
  hash[key] = {}
end
my_hash[:UIST][“2014”] = “Hawaii”
```

# Snippet 2
```
chi_hash = Hash.new do |h, k|
  h[k] = {}
end
chi_hash[:CHI][“2014”] = “Toronto”
```
Part 1: Building the Knowledge Base

# Snippet 1
var0 = Hash.new do |var1, var2|
  var1[var2] = {}
end
var0[:UIST][“2014”] = “Hawaii”

# Snippet 2
var0 = Hash.new do |var1, var2|
  var1[var2] = {}
end
var0[:CHI][“2014”] = “Toronto”
# Snippet 1
var0 = Hash.new do |var1, var2|
  var1[var2] = {}
end
var0[:UIST]["2014"] = "Hawaii"

# Snippet 2
var0 = Hash.new do |var1, var2|
  var1[var2] = {}
end
var0[:CHI]["2014"] = "Toronto"
# Snippet 1

```ruby
var0 = Hash.new do |var1, var2|
  var1[var2] = {}
end

var0[:SYM0]["2014"] = "Hawaii"
```

# Snippet 2

```ruby
var0 = Hash.new do |var1, var2|
  var1[var2] = {}
end

var0[:SYM0]["2014"] = "Toronto"
```
Part 1: Building the Knowledge Base

# Snippet 1
```ruby
var0 = Hash.new do |var1, var2|
  var1[var2] = {} 
end
var0[:SYM0]["2014"] = "Hawaii"
```

# Snippet 2
```ruby
var0 = Hash.new do |var1, var2|
  var1[var2] = {} 
end
var0[:SYM0]["2014"] = "Toronto"
```
Part 1: Building the Knowledge Base

# Snippet 1
var0 = Hash.new do |var1, var2|
  var1[var2] = {}
end
var0[:SYM0]["STR0"] = "STR1"

# Snippet 2
var0 = Hash.new do |var1, var2|
  var1[var2] = {}
end
var0[:SYM0]["STR0"] = "STR1"
Part 2: Statistical Linting

Statistical Linting
Part 2: Statistical Linting

Statistical linting: detecting code that is unlikely to occur in practice
Codex observes var0 = var1.downcase more than 200 times, but var0 = var1.downcase! only 1 time.

Warning: Line 3

Codex observes var0 = var1.downcase more than 200 times, but var0 = var1.downcase! only 1 time.
Chaining & Composition

The function `downcase!` has a side-effect and changes name

```ruby
name = "Ethan Fast"

lc_name = name.downcase!
 #=> "ethan fast"

# But downcase! has a side-effect.
# It changes the value of name.

name
 #=> "ethan fast"
```

Warning: Line 3
Codex observes `var0 = var1.downcase` more than 200 times, but `var0 = var1.downcase!` only 1 time.
Unlikely variable names

```ruby
array = {}
array["CHI"] = "Toronto"
array["UIST"] = "Hawaii"
array.keys.each do |k|
  puts "#{k} is a conference."
end
```

Warning: Line 2
Codex observes variables named `array` 116 times and variables assigned a Hash value 1248 times, but has never seen the two together.
Unlikely variable names

You might wonder: does an Array really have a method named `keys`?

```
array = {}
array["CHI"] = "Toronto"
array["UIST"] = "Hawaii"
array.keys.each do |k|
  puts "#{k} is a conference."
end
```

Warning: Line 2
Codex observes variables named `array` 116 times and variables assigned a Hash value 1248 times, but has never seen the two together.
Other kinds of analysis

# Function chaining: .join, not .to_s
"my string".split.to_s

# Type analysis: reversed arguments
arr = Array.new({}, 10)

# Block checks: puts returns nil
nums.map do |x|
  puts x
end
var0.split.to_s #=> Error: Array => String

“Function split has appeared 29 times and to_s has appeared 12 times, but they’ve never been chained together.”
Pattern annotation: finds common idioms, then annotates them using crowds.
Query for snippets with sufficient commonality and complexity

```python
mongo_query = {
    'project_count': {'gt': .02},
    'total_count': {'lt': 0.9},
    'file_count': {'lt': 0.2},
    'token_count': {'lt': 0.8},
    'function_count': {'gt': 2.0}
}
```
Next we crowdsource a title, description, and vote of usefulness from oDesk workers.
Nested Hashes

Creating a Nested Hash

```ruby
# Creating a nested Hash
my_hash = Hash.new { |h,k| h[k] = {} }

my_hash[:CHI][:Toronto] = true
```

# Naive way:
`Hash.new({})` # This is a bug!

**Total count:** 66  **Project count:** 10

Creates a Hash with a new empty Hash object as a default key value.
Nested Hashes

Creating a Nested Hash

```ruby
# Creating a nested Hash
my_hash = Hash.new { |h,k| h[k] = {} }

my_hash[:CHI][:Toronto] = true

# Naive way:
Hash.new({}) # This is a bug!
```

Creates a Hash with a new empty Hash object as a default key value

This simple idiom is easy to mess up!
By setting this to false, you can turns off caching for the Rails web framework.
Raise StandardError

```
raise StandardError.new("CHI")
```

#=> StandardError: Failed for CHI

Raise Custom Error

Total count: 66  Project count: 10

Raise a new StandardError using a custom message, passed as a string value
Library generation constructs a utility package that reflects common practice.
This idiom occurred 10 times across 5 different projects.
Create a helper method for nested Hashes

```ruby
# Modifying the Hash class to
# create a helper for nested Hashes
class Hash
  def self.nested
    Hash.new do |h, k|
      h[k] = {}
    end
  end
end

# In practice!
my_hash = Hash.nested
```

This idiom occurred **66** times across **12** different projects.
Hit-rate after **500k** LOC

Figure 3. A plot of Codex’s hit rate as it indexes code over four random samples of file orderings. The y-axis plots the database hit rate, and the x-axis plots the number of lines of code indexed.
Part 5: Evaluation: Pattern Annotation

Snippet categories

- Standard Library: 76%
- External Library: 9%
- Data / Control Flow: 14%
A survey of expert crowdworkers

86% of snippets are useful

96% are recomposable

91% have no more common form
Part 5: Evaluation: Statistical Linting

Statistical linting and false positives

We find 1,248 warnings over 49,735 lines, a rate of 2.5%.
Common false positives

```
# The functions nodes and uri are part of
# an HTML parsing library, only seen in
# a few files

@system.nodes.uri
```
Part 5: Evaluation: Statistical Linting

Ambiguous false positives

```ruby
# Some libraries override standard library
# functions like map

@network.map({:uri => :localhost})
```
Conclusion
Mining emergent practice can support a broad set of software engineering interfaces.
Programming languages can be living artifacts

Libraries self-update to the latest idioms

IDEs offer suggestions to suit new coding styles

Languages evolve to better support their users
Emergent, Crowd-scale Programming Practice in the IDE

Ethan Fast, Daniel Steffee, Lucy Wang, Michael Bernstein, Joel Brandt
Stanford HCI, Adobe Research
Conventions emerge among many different kinds of domains.
Chaining & Composition

```ruby
1 name = "Ethan Fast"
2 lc_name = name.downcase!
3 #=> "ethan fast"
4 # But downcase! has a side-effect.
5 # It changes the value of name.
6 name
7 #=> "ethan fast"
```
Chaining & Composition

The function `downcase!` has a side-effect and changes `name`.
Chaining & Composition

Codex observes \texttt{var0 = var1.downcase} more than 200 times, but \texttt{var0 = var1.downcase!} only 1 time.

The function \texttt{downcase!} has a side-effect and changes \texttt{name}.
Unlikely variable names

```ruby
array = {}
array["CHI"] = "Toronto"
array["UIST"] = "Hawaii"
array.keys.each do |k|
  puts "#{k} is a conference."
end
```
Unlikely variable names

You might wonder: does an Array really have a method named `keys`?
Unlikely variable names

Codex observes variables named array 116 times and variables assigned a Hash value many thousands of times, but we never see the two together.

You might wonder: does an Array really have a method named keys?
# Creating a nested Hash

```ruby
my_hash = Hash.new { |h, k|
  h[k] = {}
}

my_hash[:CHI][:Toronto] = true

# Naive way:
Hash.new({}) # This is a bug!
```
Nested Hashes

Assigns an empty Hash as the default key value

```ruby
# Creating a nested Hash
my_hash = Hash.new { |h,k| h[k] = {} }

my_hash[:CHI][:Toronto] = true

# Naive way:
Hash.new({}) # This is a bug!
```
Nested Hashes

This simple idiom is easy to mess up!

```
# Creating a nested Hash
my_hash = Hash.new { |h,k| h[k] = {} }
my_hash[:CHI][:Toronto] = true

# Naive way:
Hash.new({}) # This is a bug!
```

Assigns an empty Hash as the default key value
Turn off Rails Caching

```ruby
Rails.application.configure do
  config.cache_classes = true
  config.eager_load = false
  config.serve_static_assets = true
  config.static_cache_control = 'public, max-age=3600'
  config.consider_all_requests_local = true
  config.action_controller.perform_caching = false
  config.action_dispatch.show_exceptions = false
  config.action_controller.allow_forgery_protection = false
  config.action_mailer.delivery_method = :test
  config.active_support.deprecation = :stderr
end
```
Raise StandardError

```ruby
raise StandardError.new("Failed for CHI")
```

Raise a new StandardError message using a custom message.
Data mining for Codex

1. Gather Ruby code from Github
2. Parse the code into AST representation
3. Normalize the ASTs (rename variables, strings, symbols, and numbers)
4. Collapse normalized ASTs
Data mining for Codex

1. Gather Ruby code from Github

2. Parse the code into AST representation

3. Normalize the ASTs (rename variables, strings, symbols, and numbers)

4. Collapse normalized ASTs
An AST node $s$ must occur fewer than $t$ times, and its children $c_i$ must occur more then $t_i$ times

E.g., the snippet `var0.split.to_s` is composed of `.split` and `.to_s`