Cleaning Up Your Giant Codebase

Building Linters and Codemods for Better Code

Sarah Dapul-Weberman - Pinterest

August 9, 2017

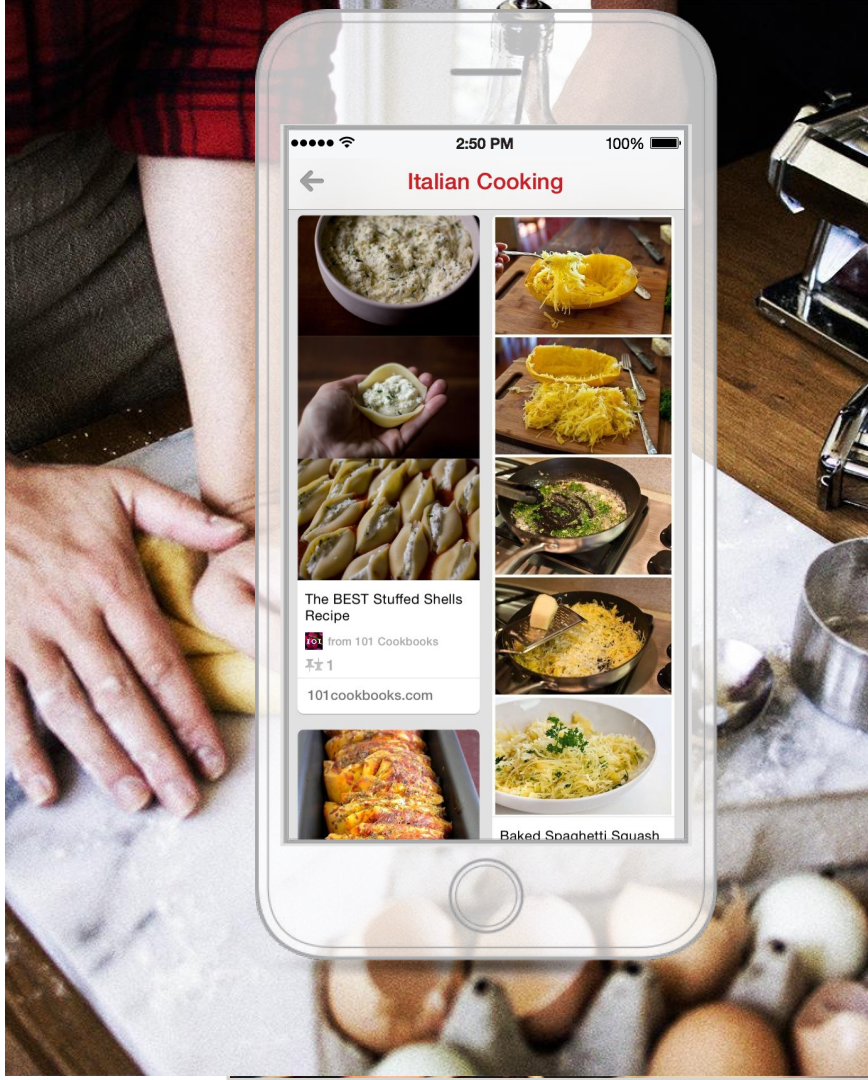
Today's Workshop:

- What we did at Pinterest & why it's important (Presentation)
- A little programming language theory and explanation (Presentation)
- Applying this to your own codebase (Workshop)

Who am I?

What is Pinterest?

- *A place for inspiration*
- *Discover new things for use in your day-to-day life*
- *Great for cooking, decorating, DIY, fashion, beauty, cute animal pictures, etc*
- *Also: a really huge site*




50+ Billion Pins
categorized by people into more than
1 Billion Boards

A few numbers...

175M+ Monthly Active Users

Engineers	RPS
400+ (~50 in 2012)	**150k** on web

What happens when you grow extremely fast?

Things get messy

how to clean up messy room code

All Pins

Home Explore Sarah Bag 2

How to Clean Your House When You Feel Paralyzed by "The…

110

I know what it's like to stand in the middle of a messy room…

Kellie Zollars
Cleaning Tips

How to Clean Up a Messy Bedroom Quickly

473

How to Clean Up a Messy Bedroom Quickly thumbnail I just…

Julie Renshaw South…
Cleaning

How to keep a child's room clean

7.2k

After lots of fighting with my daughter about keeping her…

Donna Chapin
Parenting

How to Clean A Very Messy Room Fast

102

How to Clean Up a Really Messy Room

Linters

Custom Linters

Custom Linters...
Under the hood

A little programming language theory

Abstract Syntax Trees

From Wikipedia:

In computer science, an **abstract syntax tree** (**AST**), or just **syntax tree**, is a **tree** representation of the **abstract syntactic** structure of source code written in a programming language. Each node of the **tree** denotes a construct occurring in the source code.

???

Abstract Syntax Tree Example

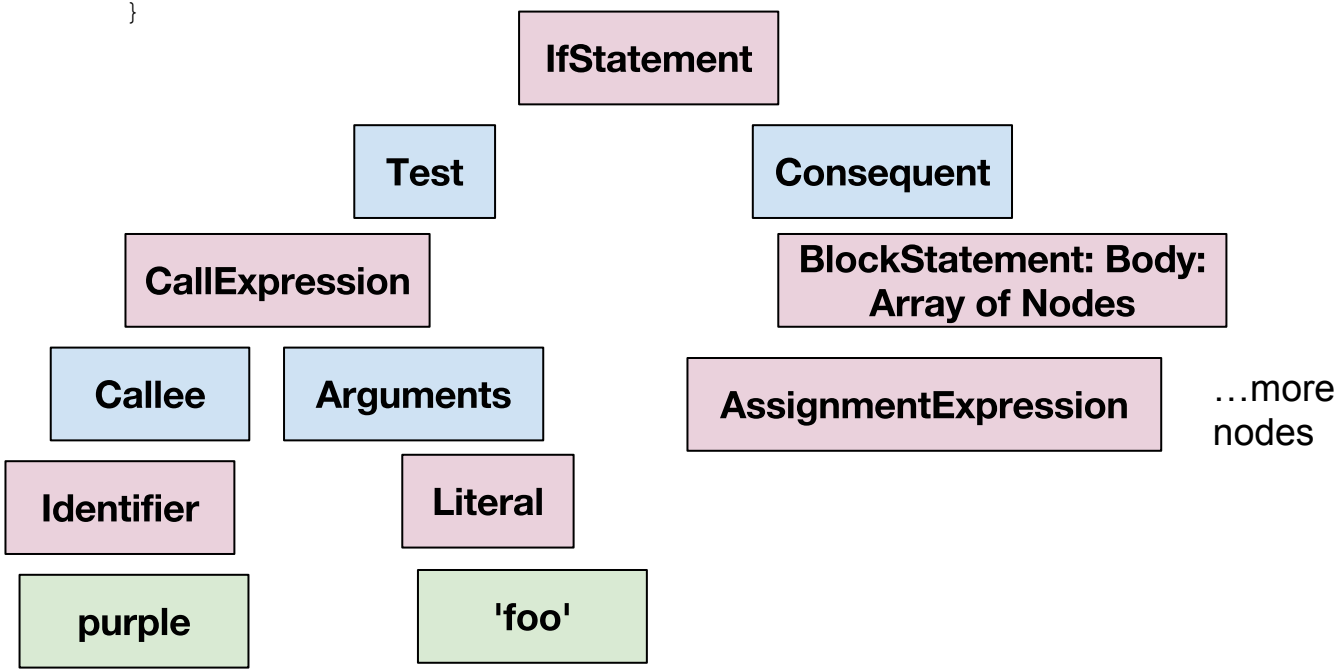
```
if (purple('foo')) {
      bar = baz || boo;
}
```

```
- body: [
   - IfStatement {
        type: "IfStatement"
      - test: CallExpression {
           type: "CallExpression"
         - callee: Identifier {
              type: "Identifier"
              name: "purple"
           }
         - arguments: [
            - Literal = $node {
                 type: "Literal"
                 value: "foo"
                 rawValue: "foo"
                 raw: "'foo'"
              }
           ]
        }
```

https://astexplorer.net/

Abstract Syntax Tree Example

```
if (purple('foo')) {
      bar = baz ||
boo;
}
```




```
-body: [
 -IfStatement {
   type: "IfStatement"
  -test: CallExpression {
    type: "CallExpression"
   -callee: Identifier {
     type: "Identifier"
     name: "purple"
    }
   -arguments: [
    -Literal {
      type: "Literal"
      value: "foo"
      rawValue: "foo"
      raw: "'foo'"
     }
    ]
   }
  -consequent: BlockStatement {
    type: "BlockStatement"
   -body: [
    -ExpressionStatement {
      type: "ExpressionStatement"
     -expression: AssignmentExpression {
       type: "AssignmentExpression"
       operator: "="
      -left: Identifier {
        type: "Identifier"
        name: "bar"
       }
      -right: LogicalExpression {
        type: "LogicalExpression"
       -left: Identifier {
         type: "Identifier"
         name: "baz"
        }
        operator: "||"
       -right: Identifier = $node {
         type: "Identifier"
         name: "boo"
        }
       }
      }
     }
    ]
```


https://astexplorer.net/

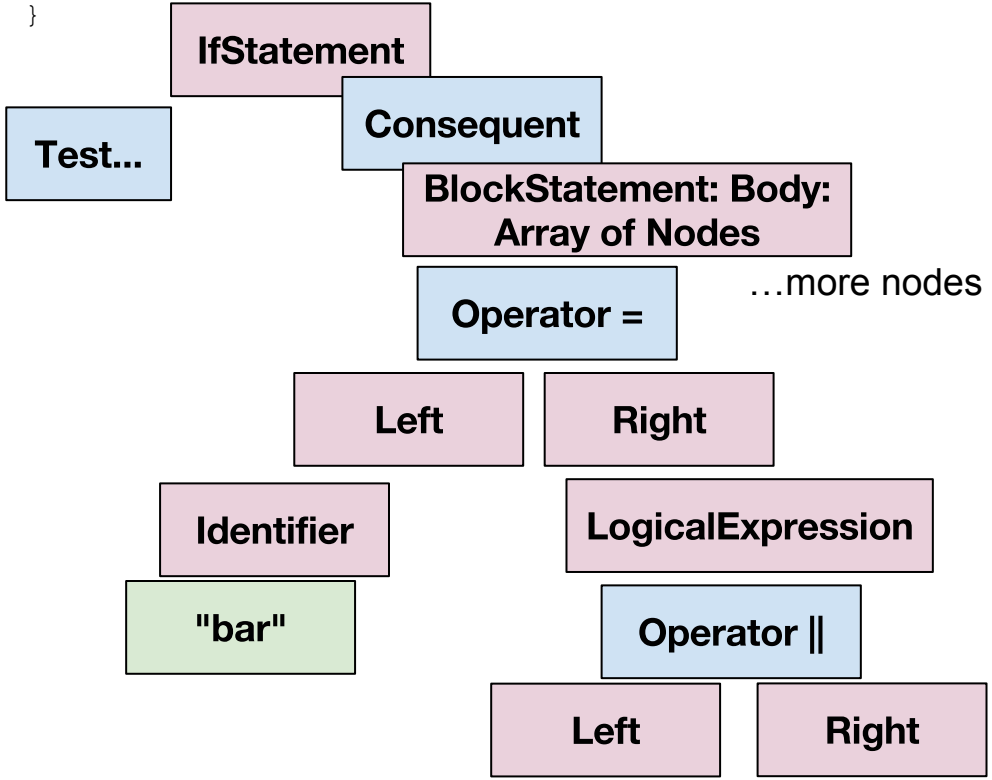
Abstract Syntax Tree Example

```
if (purple('foo')) {
      bar = baz || boo;
}
```

https://astexplorer.net/

Abstract Syntax Tree Example

```
if (purple('foo')) {
      bar = baz ||
boo;
}
```



…more nodes if more lines in body


```
-body: [
 -IfStatement {
   type: "IfStatement"
  -test: CallExpression {
    type: "CallExpression"
   -callee: Identifier {
     type: "Identifier"
     name: "purple"
    }
   -arguments: [
    -Literal {
      type: "Literal"
      value: "foo"
      rawValue: "foo"
      raw: "'foo'"
     }
    ]
   }
  -consequent: BlockStatement {
    type: "BlockStatement"
   -body: [
    -ExpressionStatement {
      type: "ExpressionStatement"
     -expression: AssignmentExpression {
       type: "AssignmentExpression"
       operator: "="
      -left: Identifier {
        type: "Identifier"
        name: "bar"
       }
      -right: LogicalExpression {
        type: "LogicalExpression"
       -left: Identifier {
         type: "Identifier"
         name: "baz"
        }
        operator: "||"
       -right: Identifier = $node {
         type: "Identifier"
         name: "boo"
        }
       }
      }
     }
    ]
```


https://astexplorer.net/

Abstract Syntax Tree Example

```
if (purple('foo')) {
      bar = baz || boo;
}
```

https://astexplorer.net/

Linter:

every time we see a pattern we don't like, throw an error. This can be anything! e.g.: "No one letter variable names"

Codemod:

every time you see a pattern you don't like, rewrite the code to fit the pattern you want

Transpiler:

Rewrite output of AST into entirely different programming language

Sneak Peek: Automating Migrations at Scale (talk tomorrow, 9:20AM)

Questions?

Workshop

1) Take your codebase + language of choice: run a simple file through an AST Parser and examine what you get
2) Write a program that takes this output and does something with it
    a) This could be building a linter or transforming your code!
        i) Linter: every time we see a pattern we don't like, throw an error. This can be anything! e.g.: "No one letter variable names"
        ii) Codemod: every time you see a pattern you don't like, rewrite the code to fit the pattern you want
3) Run this program on all your files

sarahddubs.com/workshop