On the biodiversity of source code: rigid or plastic repair?

Benoit Baudry & Martin Monperrus
University of Lille & INRIA
Dagstuhl #13061 - Feb, 2013

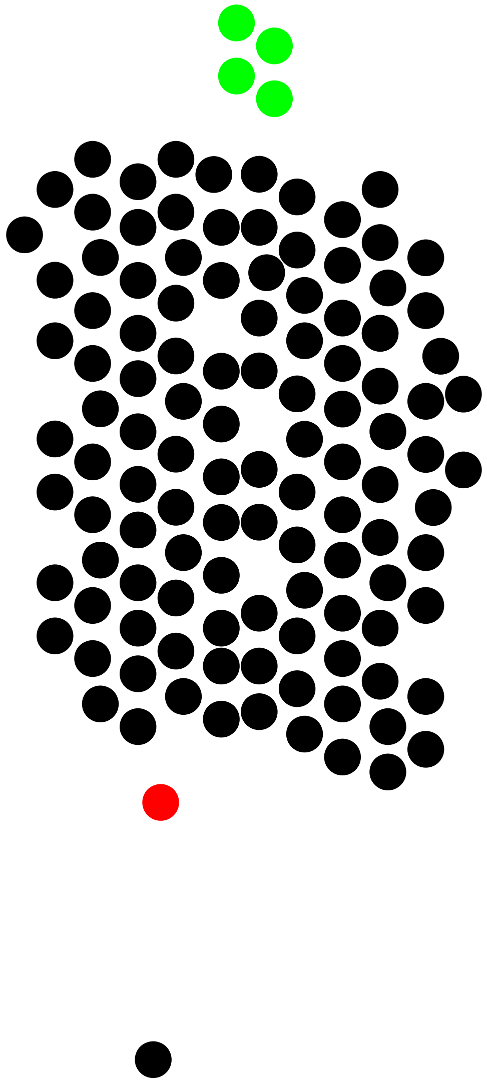
Preface

Intention:

- presenting emerging results talk
- getting feedback for crystallization of concepts

Content:

- technical content (Detecting missing method calls, TOSEM, to appear)
- new results
- conceptual interpretation and discussion



Intuition: A "buggy' code is:
- Close to the majority
- Alone

"the redundancy defines the anomaly"

Abstracting over code

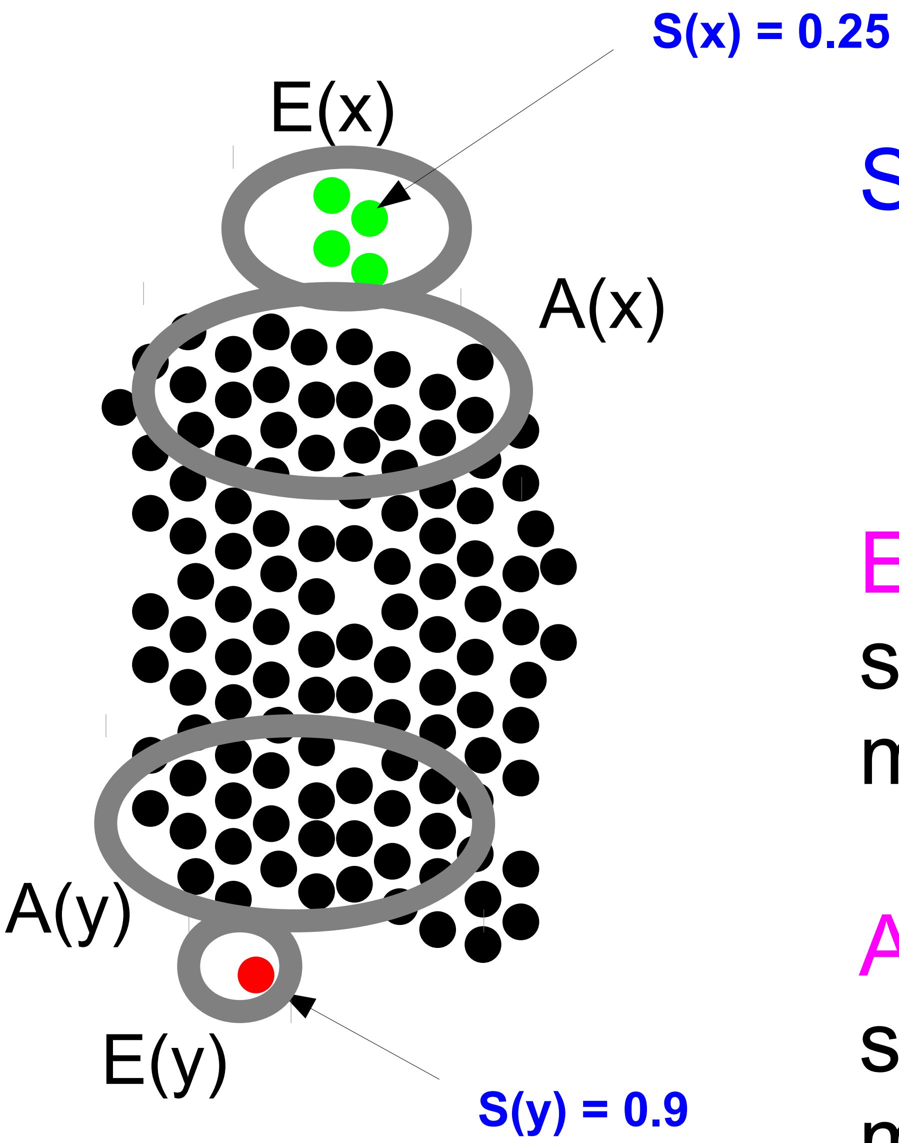
```
class A extends Page {
  Button b;

  Button createButton(){
    b = new Button();
    b.setText("hello");
    b.setColor(GREEN);
    ...(other code)
    return b;
  }
}
```

A "type-usage" is composed of:
Type(b) = 'Button'
Context(b) = 'Page.createButton()'
Calls(b) = {<init>, setText, setColor}

Can we use the intrinsic redundancy of code to detect missing method calls?

(ECOOP'10, ACM TOSEM to appear)

Co-authored with Marcel Bruch, Mira Mezini

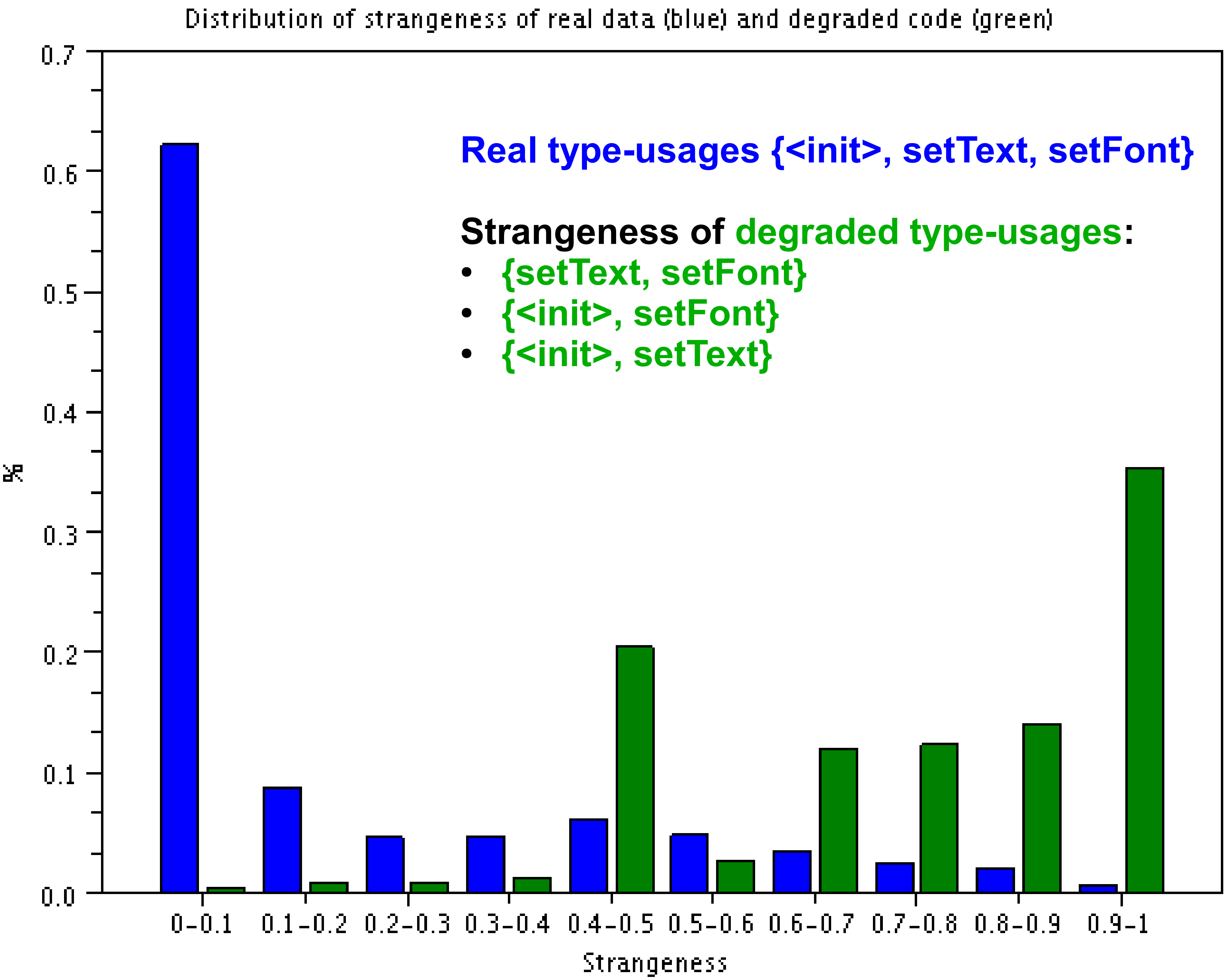


Strangeness measure
0 is fine
1 is suspicious

E(X) - Exact-similarity:
same type and context, same method calls

A(X) - Almost-similarity:
same type and context, same method calls - 1

41193 type usages in Eclipse 3.5 Classic related to SWT



Real type-usages {<init>, setText, setFont}

Strangeness of degraded type-usages:

- **{setText, setFont}**
- **{<init>, setFont}**
- **{<init>, setText}**

Redundancy for Prediction

Method calls that are not in x but in A(x) are missing:

Example:
let x = {<init>, setText }
setColor 98 times in A(x), |A(x)|=100

Redundancy used for missing method calls

Predicting removed method calls:
Precision: 77%
Recall: 68%
(42845 queries)

Patches accepted by lead developers of OSS

Step back - exploratory research:

what is the topology of type usages across software package boundaries?

Co-authored with Diego Mendez

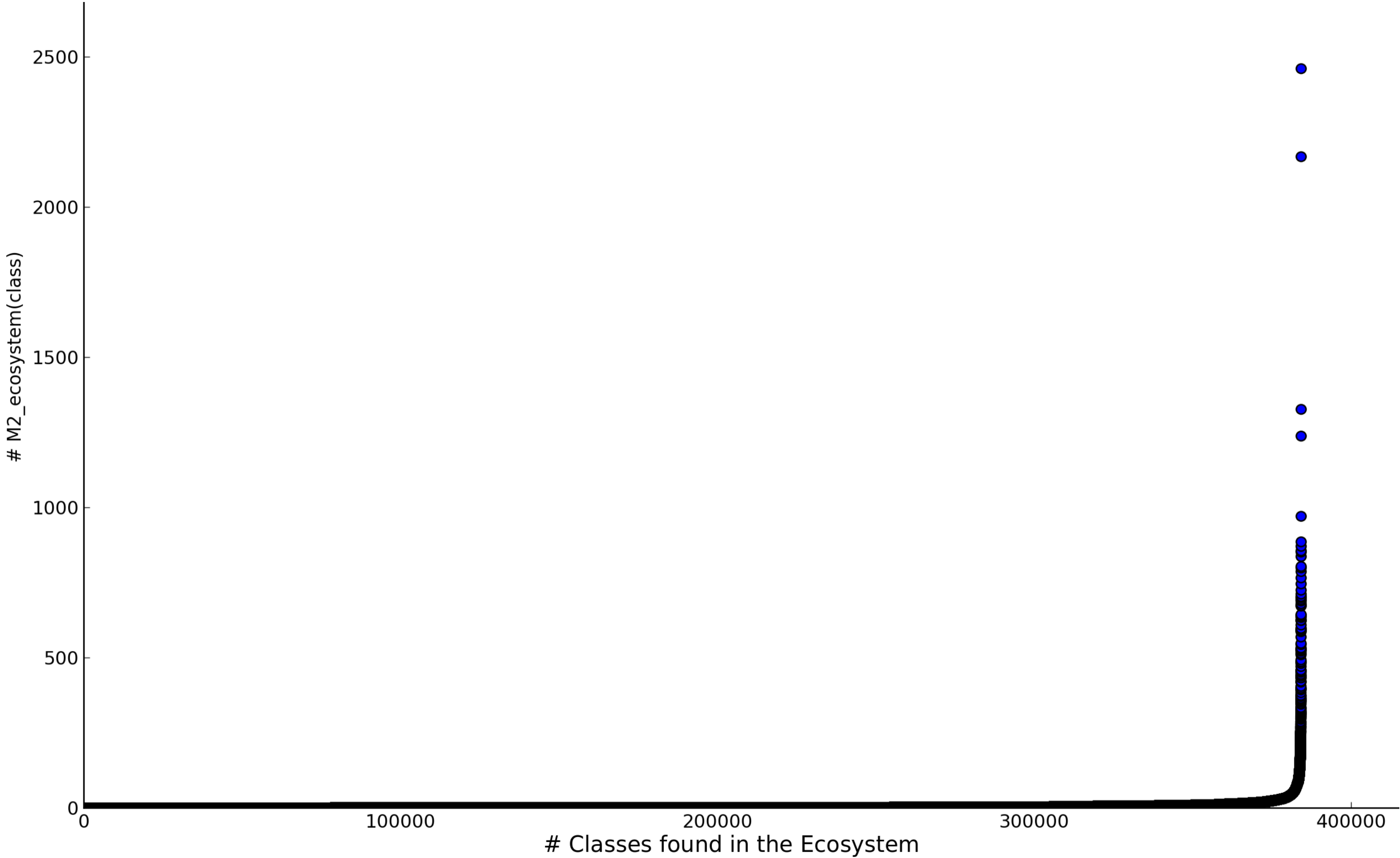
4343 Jar files, 386183 classes

Type-usages of Jlabel:

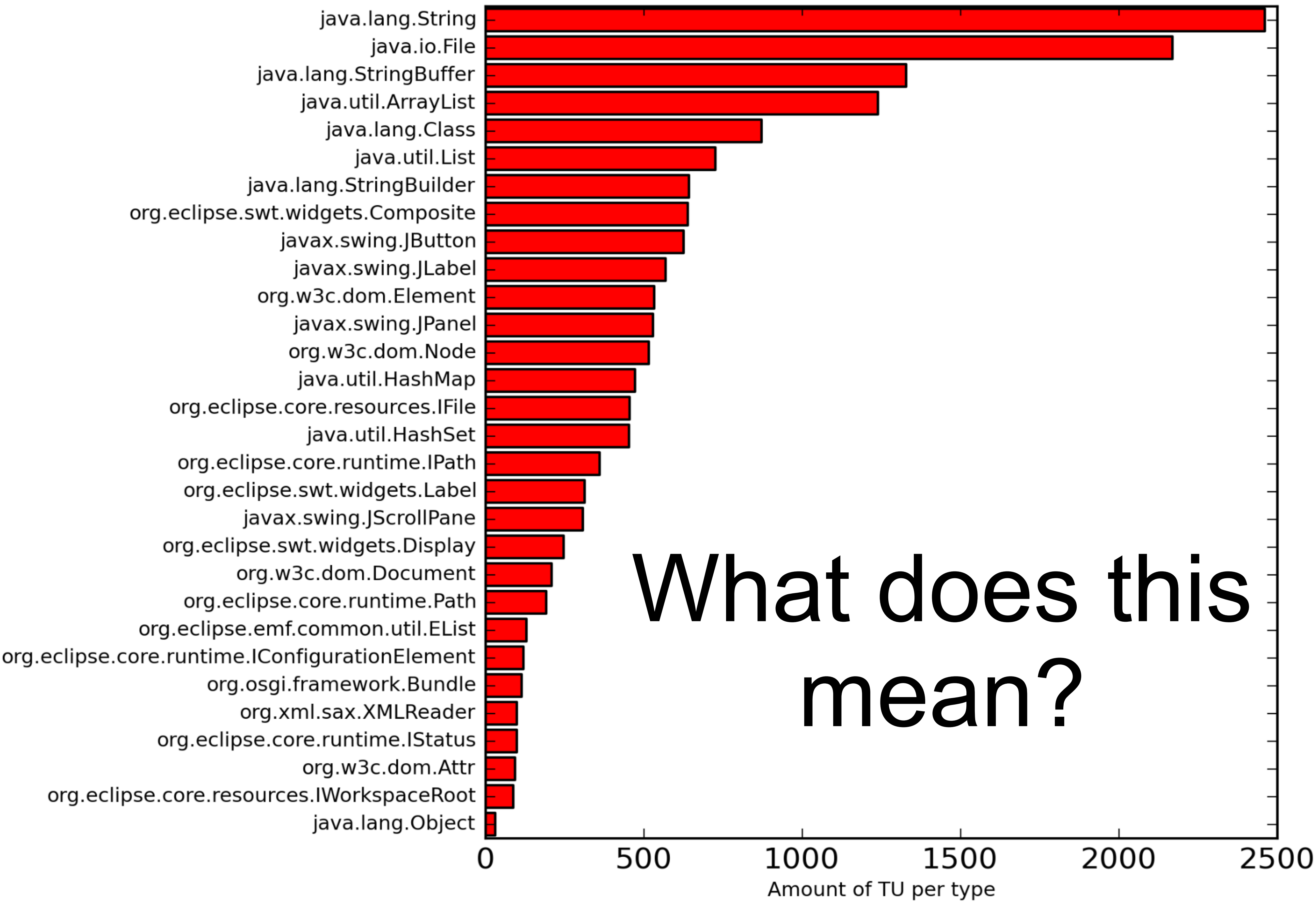
```
call:<void setMinimumSize(java.awt.Dimension)> call:<void setText(java.lang.String)>
call:<void <init>()> call:<void setHorizontalAlignment(int)> call:<void setText(java.lang.String)> call:<void
setBorder(javax.swing.border.Border)>
call:<void setVisible(boolean)> call:<void setLabelFor(java.awt.Component)>
call:<void <init>()> call:<void setMinimumSize(java.awt.Dimension)> call:<void setText(java.lang.String)>
call:<void setForeground(java.awt.Color)>
call:<void <init>(java.lang.String
call:<void <init>()> call:<java.awt.Font getFont()> call:<void setFont(java.awt.Font)> call:<void
setBorder(javax.swing.border.Border)> call:<void setForeground(java.awt.Color)>
call:<void setFont(java.awt.Font)> call:<void setBackground(java.awt.Color)> call:<void
setForeground(java.awt.Color)>
call:<void removeMouseListener(java.awt.event.MouseListener)>
call:<void <init>()> call:<void setText(java.lang.String)>
call:<void <init>(java.lang.String)> call:<void setPreferredSize(java.awt.Dimension)> call:<void
setBackground(java.awt.Color)> call:<void setHorizontalAlignment(int)> call:<void setOpaque(boolean)>
call:<void <init>(java.lang.String)> call:<void setFont(java.awt.Font)> call:<void setForeground(java.awt.Color)>
call:<void setHorizontalTextPosition(int)>
call:<java.awt.Color getForeground()> call:<void setBorder(javax.swing.border.Border)> call:<java.awt.Font
getFont()> call:<void setFont(java.awt.Font)> call:<void setLabelFor(java.awt.Component)>
call:<void <init>(java.lang.String)> call:<void setToolTipText(java.lang.String)> call:<void setFont(java.awt.Font)>
call:<void setOpaque(boolean)> call:<void setHorizontalAlignment(int)>
call:<void <init>(javax.swing.Icon)> call:<void setMinimumSize(java.awt.Dimension)>
call:<void <init>(java.lang.String)> call:<javax.accessibility.AccessibleContext getAccessibleContext()>
call:<void <init>(java.lang.String
call:<void <init>(java.lang.String)> call:<void setHorizontalAlignment(int)> call:<void setVerticalAlignment(int)>
....
```

570 different type-usages

Biodiversity: Number of species / Number of type-usages



Biodiversity: Number of species / Number of type-usages



What does this mean?

What does this mean?

For some classes:

"Usage diversity may not be encoded into rigid rules"

"Usage diversity may reflect a high class plasticity"

Rigid repair

- Is based on a correctness envelope
  - Object protocol / typestate
  - Pre/Postconditions, assertions
- Uses this correctness envelope to define anomalies and fixes
  - Wasylkowski et al. (FSE'07), Bodden (ICSE'11), Wei et al. (ISSTA'10), Nguyen et al. (ICSE'13)
- Repair means entering back the correctness envelope

Plastic repair

- Uses acceptability envelopes to define anomalies
  - controlled uncertainty (Locasto et al. NDSS'06), loop perforation (Misailovic et al. ICSE'10, Zhu et al. POPL'12)
- Is founded on intrinsic redundancy to find fixes
  - Genprog (Weimer et al., ICSE'09 etc.), Naturalness of Software (Hindle et al. ICSE'12), Forrest 2002, Gorla et al. (ICSE'13)
- How to characterize and recognize plastic problem domains and zones of plasticity?

"Embed the nature of code into the repair techniques"

For certain domains, the presence of a correctness envelope allows rigid repair.

In others, the presence of redundancy and diversity calls for plastic repair.