

Classifying Edits to Variability in Source Code

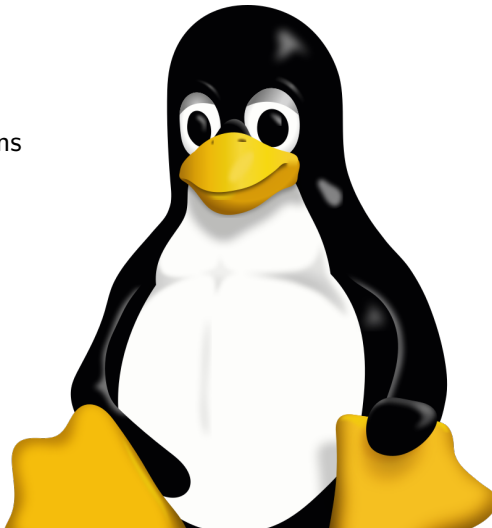
Paul Bittner, Christof Tinnes, Alexander Schultheiß, Sören Viegner, Timo Kehrer, and Thomas Thüm | Feb 24, 2023

Software Comprises Massive Evolving Variability

$\geq 4,000$
configuration options

$\geq 10^{725}$
different variants

[2007]

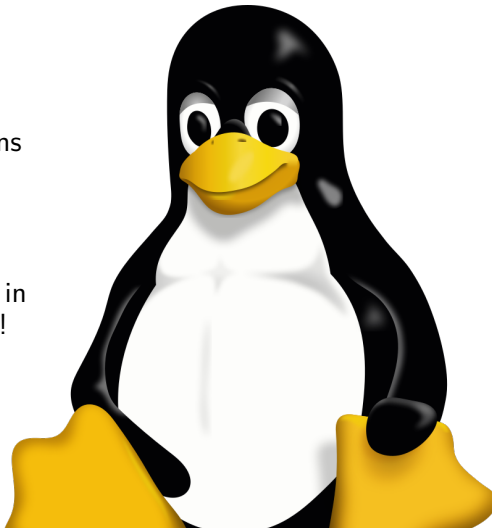


Software Comprises Massive Evolving Variability

$\geq 4,000$
configuration options

$\geq 10^{725}$
different variants
Only $\sim 10^{80}$ atoms in
observable universe!

[2007]

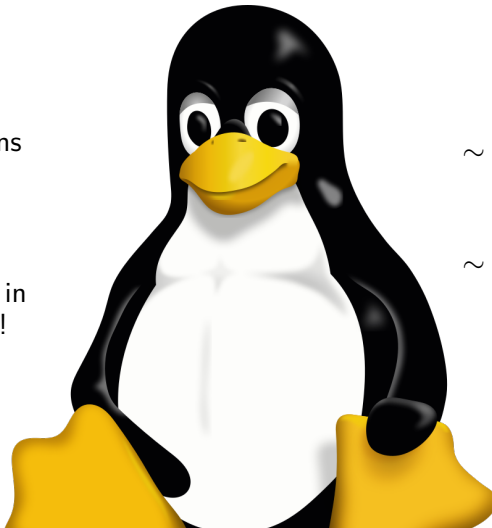


Software Comprises Massive Evolving Variability

$\geq 4,000$
configuration options

$\geq 10^{725}$
different variants
Only $\sim 10^{80}$ atoms in
observable universe!

[2007]



$\sim 21,000$
new LOC / week

~ 15
new configuration
options / week

Variability via C Preprocessor

```
static void
f_foreground(/* params */)
{
#ifdef FEAT_GUI
    if (gui.in_use)
        gui_mch_set_foreground();
#else
# ifdef MSWIN
    win32_set_foreground();
# endif
#endif
}
```

Variability via C Preprocessor

```
static void
f_foreground(/* params */)
{
#ifdef FEAT_GUI
    if (gui.in_use)
        gui_mch_set_foreground();
#else
# ifdef MSWIN
    win32_set_foreground();
# endif
#endif
}
```

FEAT_GUI



```
static void
f_foreground(/* params */)
{
    if (gui.in_use)
        gui_mch_set_foreground();
}
```

Variability via C Preprocessor

```
static void
f_foreground(/* params */)
{
#ifdef FEAT_GUI
    if (gui.in_use)
        gui_mch_set_foreground();
#else
# ifdef MSWIN
    win32_set_foreground();
# endif
#endif
}
```

FEAT_GUI

¬FEAT_GUI, MSWIN

```
static void
f_foreground(/* params */)
{
    if (gui.in_use)
        gui_mch_set_foreground();
}
```

```
static void
f_foreground(/* params */)
{
    win32_set_foreground();
}
```

Variability via C Preprocessor

```
static void
f_foreground(/* params */)
{
#ifdef FEAT_GUI
    if (gui.in_use)
        gui_mch_set_foreground();
#else
# ifdef MSWIN
    win32_set_foreground();
# endif
#endif
}
```

FEAT_GUI

\neg FEAT_GUI, MSWIN

\neg FEAT_GUI, \neg MSWIN

```
static void
f_foreground(/* params */)
{
    if (gui.in_use)
        gui_mch_set_foreground();
}
```

```
static void
f_foreground(/* params */)
{
    win32_set_foreground();
}
```

```
static void
f_foreground(/* params */)
{
}
```


Edits to Variability via C Preprocessor

```
#ifdef A
    foo();
#else
    #ifdef B
        baz();
    #endif
#endif
```

Example simplified from Vim commit [afde13b](#).

Edits to Variability via C Preprocessor

```
#ifdef A
    foo();
#else
    #ifdef B
        baz();
    #endif
#endif
```

Commit [afde13b](#)

```
#ifdef A
    foo();
    bar();
#endif
#if B && (!A || C)
    baz();
#endif
```

Example simplified from Vim commit [afde13b](#).

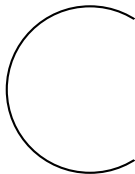
Edits to Variability via C Preprocessor

<pre>#ifdef A foo(); #else #ifdef B baz(); #endif #endif</pre>	→	<pre>#ifdef A foo(); -#else - #ifdef B + bar(); +#endif +#if B && (!A C) baz(); - #endif #endif</pre>	→	<pre>#ifdef A foo(); bar(); #endif #if B && (!A C) baz(); #endif</pre>
--	---	--	---	---

Example simplified from Vim commit [afde13b](#).

Related Work on Edit Classification is . . .

incomplete



[Stănciulescu et al., 2016]

[Borba et al., 2012]

[Al-Hajjaji et al., 2016]

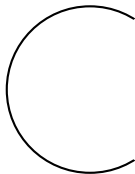
[Passos et al., 2016]

Related Work on Edit Classification is . . .

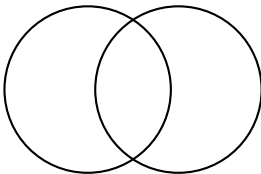
incomplete

or

ambiguous



[Stănciulescu et al., 2016]
[Borba et al., 2012]
[Al-Hajjaji et al., 2016]
[Passos et al., 2016]



[Ji et al., 2015]
[Stănciulescu et al., 2016]

Related Work on Edit Classification is . . .

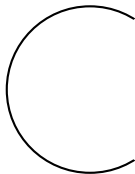
incomplete

or

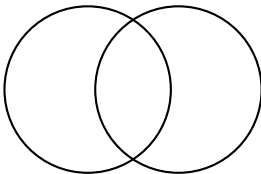
ambiguous

or

not automatable



[Stănciulescu et al., 2016]
[Borba et al., 2012]
[Al-Hajjaji et al., 2016]
[Passos et al., 2016]



[Ji et al., 2015]
[Stănciulescu et al., 2016]



[Ji et al., 2015]
[Borba et al., 2012]

```
#ifdef A
    foo();
#else
    #ifdef B
        baz();
    #endif
#endif
```

diff

```
#ifdef A
    foo();
-#else
-    #ifdef B
+    bar();
+#endif
+#if B && (!A || C)
    baz();
-    #endif
#endif
```

diff

```
#ifdef A
    foo();
    bar();
#endif
#if B && (!A || C)
    baz();
#endif
```

Research Goal:
complete, unambiguous, and automatable
model and classification

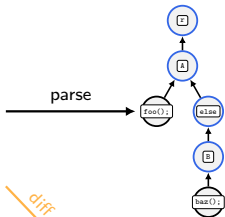
?

Classification

```

#ifdef A
    foo();
#else
    #ifdef B
        baz();
    #endif
#endif

```



First, we introduce *Variation Trees* as a model for variability in source code.

diff

```

#ifdef A
    foo();
-#else
- #ifdef B
+ bar();
+#endif
+#if B && (!A || C)
    baz();
- #endif
#endif

```

diff

Research Goal:
complete, unambiguous, and automatable
model and classification

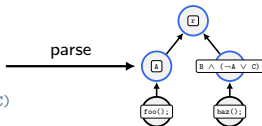
?

Classification

```

#ifdef A
    foo();
    bar();
#endif
#if B && (!A || C)
    baz();
#endif

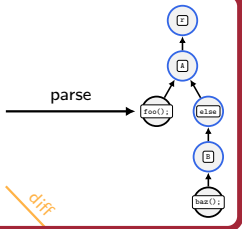
```




```

#ifdef A
    foo();
#else
    #ifdef B
        baz();
    #endif
#endif

```

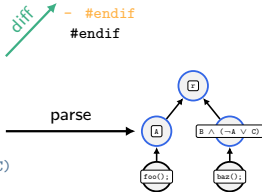


First, we introduce *Variation Trees* as a model for variability in source code.

```

#ifdef A
    foo();
-#else
- #ifdef B
+ bar();
+#endif
+if B && (!A || C)
    baz();
- #endif
#endif

```



```

#ifdef A
    foo();
    bar();
#endif
#if B && (!A || C)
    baz();
#endif

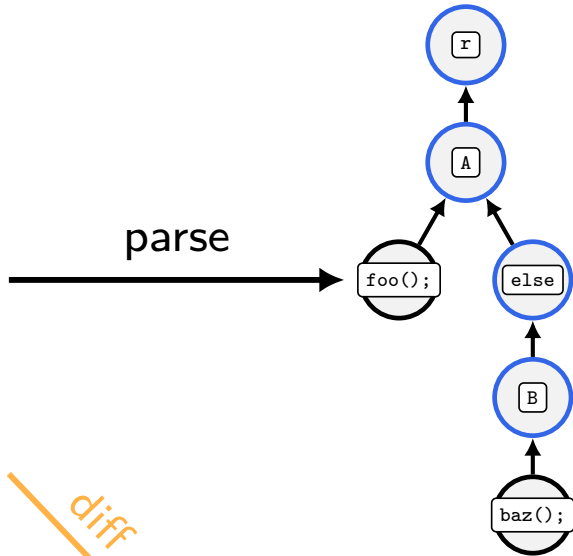
```

Research Goal:
complete, unambiguous, and automatable
model and classification

?

Classification

```
#ifdef A
    foo();
#else
    #ifdef B
        baz();
    #endif
#endif
```



diff

#ifdef A

```

#ifdef A
    foo();
#else
    #ifdef B
        baz();
    #endif
#endif

```

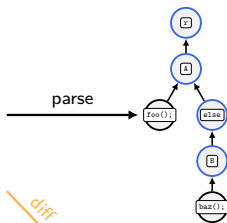


Diagram illustrating the diff between the initial code and the modified code. The diff shows the following changes:

```

#ifdef A
    foo();
-#else
- #ifdef B
+ bar();
+#endif
+#if B && (!A || C)
    baz();
- #endif
#endif

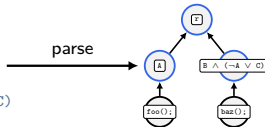
```

An orange arrow labeled "diff" points from the initial code to the modified code. A green arrow labeled "diff" points from the modified code to the final parse tree.

```

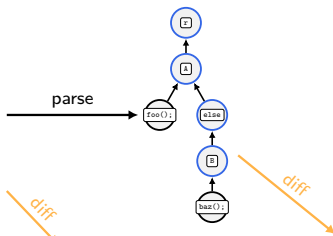
#ifdef A
    foo();
    bar();
#endif
#if B && (!A || C)
    baz();
#endif

```

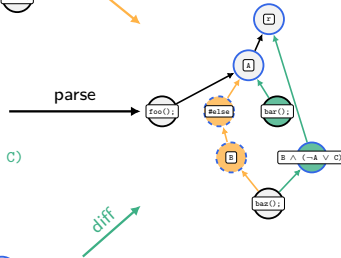


Edits to variability become edits to Variation Trees

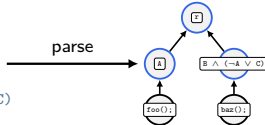
```
#ifdef A
foo();
#else
#ifdef B
baz();
#endif
#endif
```



```
#ifdef A
foo();
-#else
- #ifdef B
+ bar();
+#endif
+#if B && (!A || C)
baz();
- #endif
#endif
```



```
#ifdef A
foo();
bar();
#endif
#if B && (!A || C)
baz();
#endif
```

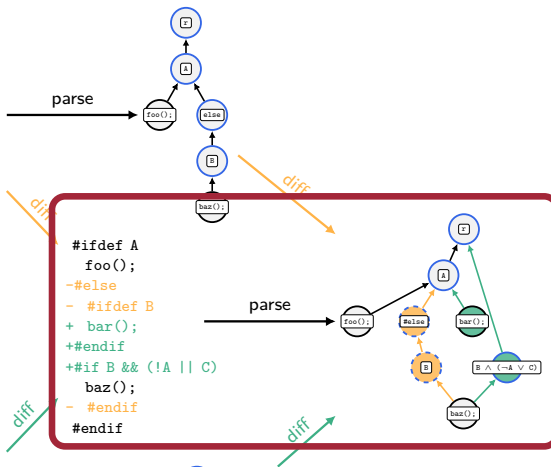


Edits to variability become
edits to Variation Trees, for
which we introduce
Variation Diffs.

```

#ifdef A
    foo();
#else
    #ifdef B
        baz();
    #endif
#endif

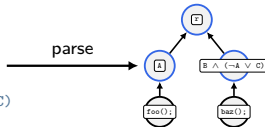
```



```

#ifdef A
    foo();
    bar();
#endif
#if B && (!A || C)
    baz();
#endif

```

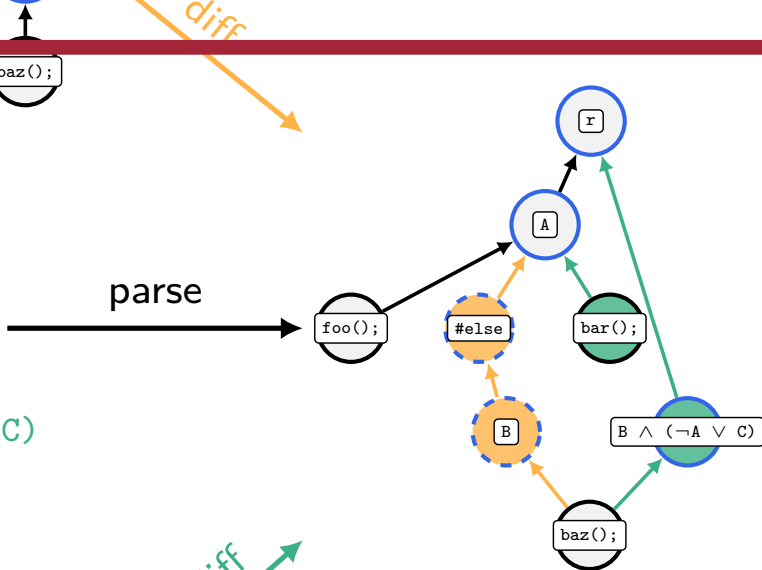


Edits to variability become edits to Variation Trees, for which we introduce *Variation Diffs*.

```

#ifdef A
    foo();
-#else
-  #ifdef B
+  bar();
+#endif
+#if B && (!A || C)
    baz();
-  #endif
#endif

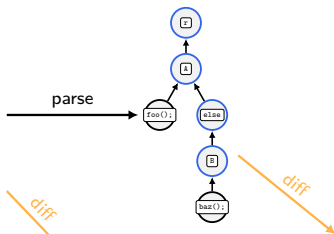
```



```

#ifdef A
  foo();
#else
  #ifdef B
    baz();
  #endif
#endif

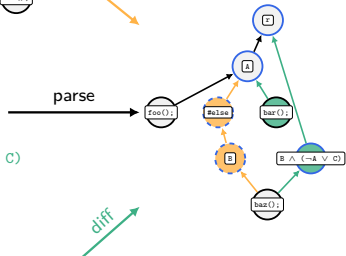
```



```

#ifdef A
  foo();
-#else
- #ifdef B
+ bar();
+#endif
+#if B && (!A || C)
  baz();
- #endif
#endif

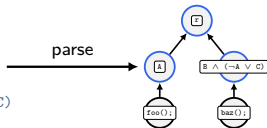
```



```

#ifdef A
  foo();
  bar();
#endif
#if B && (!A || C)
  baz();
#endif

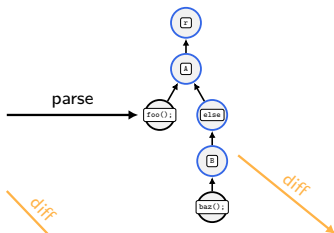
```



```

#ifdef A
    foo();
#else
    #ifdef B
        baz();
    #endif
#endif

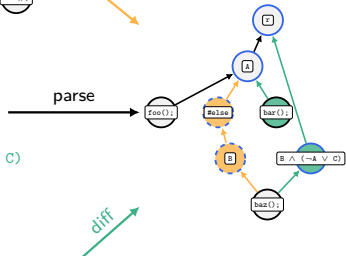
```



```

#ifdef A
    foo();
-#else
-    #ifdef B
+    bar();
+#endif
+#if B && (!A || C)
    baz();
-    #endif
#endif

```



classify

Classifying Edits to
Variability in Source
Code

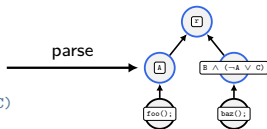
reduces to

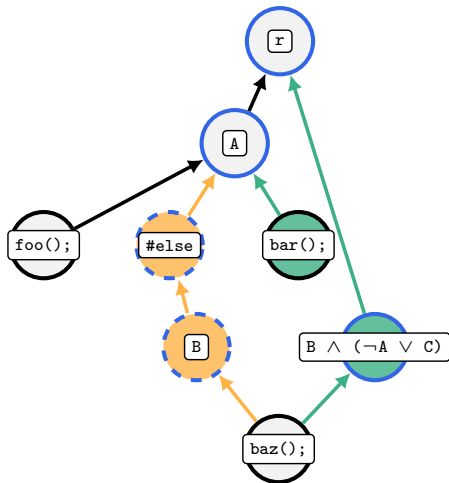
Classifying Structures in
Variation Diffs


```

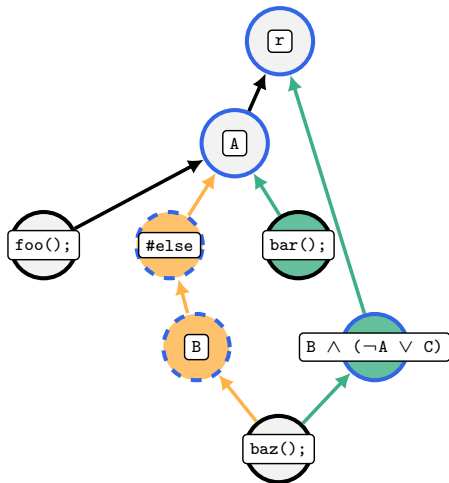
#ifdef A
    foo();
    bar();
#endif
#if B && (!A || C)
    baz();
#endif

```



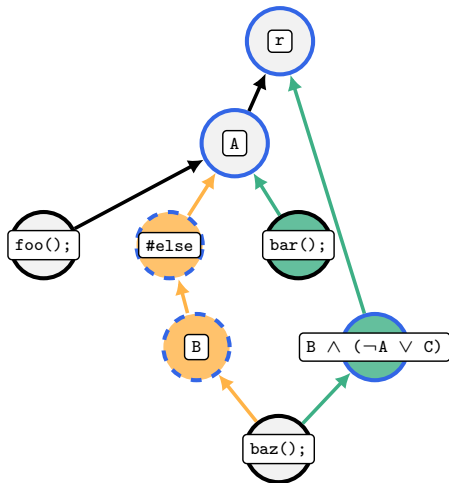


 is unchanged



`foo();` is unchanged

`bar();` is **added** to feature A



`foo();` is unchanged

`bar();` is **added** to feature A

`baz();` is moved
 from $B \wedge \neg A$
 to $B \wedge (\neg A \vee C)$

Classification $:=$ Set of Classes

Classification \coloneqq Set of Classes

$class : \textcircled{\text{code}} \rightarrow \{true, false\}$

Classification \coloneqq Set of Classes







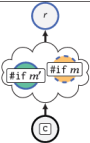
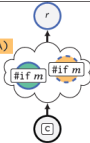
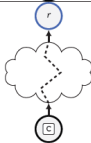
$class : \textcircled{\text{code}} \rightarrow \{true, false\}$

Example:








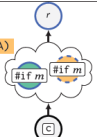

$AddToPC(\textcircled{\text{code}}) := \text{added}(\textcircled{\text{code}}) \wedge \neg \text{added}(p_a(\textcircled{\text{code}}))$



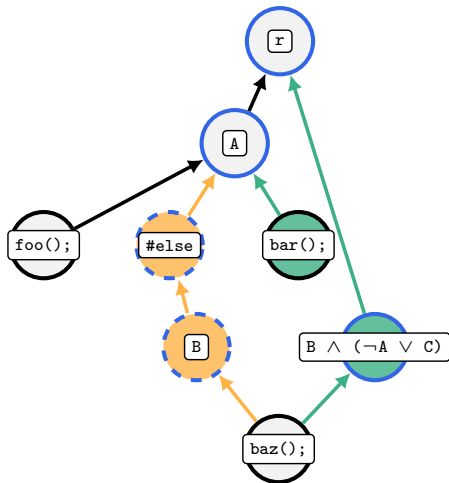
We define 9 classes.

<p><i>AddWithMapping</i>(c) := $\text{added}(c) \wedge \text{added}(M_a(c))$</p> <p>+ #if m c #endif</p> 	<p><i>AddToPC</i>(c) := $\text{added}(c) \wedge \neg \text{added}(M_a(c))$</p> <p>+ #if m c #endif</p> 	<p><i>Specialization</i>(c) := $\text{unchanged}(c)$ $\wedge \neg(PC_b(c) \models PC_a(c))$ $\wedge (PC_a(c) \models PC_b(c))$</p> <p>+ #if m c #endif</p> 
<p><i>RemWithMapping</i>(c) := $\text{removed}(c) \wedge \text{removed}(M_b(c))$</p> <p>- #if m c #endif</p> 	<p><i>RemFromPC</i>(c) := $\text{removed}(c) \wedge \neg \text{removed}(M_b(c))$</p> <p>- #if m c #endif</p> 	<p><i>Generalization</i>(c) := $\text{unchanged}(c)$ $\wedge (PC_b(c) \models PC_a(c))$ $\wedge \neg(PC_a(c) \models PC_b(c))$</p> <p>- #if m c #endif</p> 
<p><i>Reconfiguration</i>(c) := $\text{unchanged}(c)$ $\wedge \neg(PC_b(c) \models PC_a(c))$ $\wedge \neg(PC_a(c) \models PC_b(c))$</p> <p>- #if m c #endif</p> 	<p><i>Refactoring</i>(c) := $\text{unchanged}(c)$ $\wedge (PC_b(c) \models PC_a(c))$ $\wedge (PC_a(c) \models PC_b(c))$ $\wedge (path_b(c) \neq path_a(c))$</p> <p>- #if $A \parallel (B \ \&\& \ !A)$ c #endif</p> 	<p><i>Untouched</i>(c) := $\text{unchanged}(c)$ $\wedge (PC_b(c) \models PC_a(c))$ $\wedge (PC_a(c) \models PC_b(c))$ $\wedge (path_b(c) = path_a(c))$</p> 

We define 9 classes.

$AddWithMapping(c) :=$ $added(c) \wedge added(M_a(c))$	$+ \begin{array}{l} \text{\#if } m \\ c \\ \text{\#endif} \end{array}$ 	$AddToPC(c) :=$ $added(c) \wedge \neg added(M_a(c))$	$+ \begin{array}{l} \text{\#if } m \\ c \\ \text{\#endif} \end{array}$ 	$Specialization(c) := unchanged(c)$ $\wedge \neg(PC_b(c) \models PC_a(c))$ $\wedge (PC_a(c) \models PC_b(c))$	$+ \begin{array}{l} \text{\#if } m \\ c \\ \text{\#endif} \end{array}$ 
$RemWithMapping(c) :=$ $removed(c) \wedge removed(M_b(c))$	$- \begin{array}{l} \text{\#if } m \\ c \\ \text{\#endif} \end{array}$ 	$RemFromPC(c) :=$ $removed(c) \wedge \neg removed(M_b(c))$	$- \begin{array}{l} \text{\#if } m \\ c \\ \text{\#endif} \end{array}$ 	$Generalization(c) := unchanged(c)$ $\wedge (PC_b(c) \models PC_a(c))$ $\wedge \neg(PC_a(c) \models PC_b(c))$	$- \begin{array}{l} \text{\#if } m \\ c \\ \text{\#endif} \end{array}$ 
$Reconfiguration(c) := unchanged(c)$ $\wedge \neg(PC_b(c) \models PC_a(c))$ $\wedge \neg(PC_a(c) \models PC_b(c))$	$- \begin{array}{l} \text{\#if } m \\ \text{\#if } m' \\ c \\ \text{\#endif} \end{array}$ 	$Refactoring(c) := unchanged(c)$ $\wedge (PC_b(c) \models PC_a(c))$ $\wedge (PC_a(c) \models PC_b(c))$ $\wedge (path_b(c) \neq path_a(c))$	$- \begin{array}{l} \text{\#if } A \parallel (B \ \&\& \ !A) \\ \text{\#if } A \parallel B \\ c \\ \text{\#endif} \end{array}$ 	$Untouched(c) := unchanged(c)$ $\wedge (PC_b(c) \models PC_a(c))$ $\wedge (PC_a(c) \models PC_b(c))$ $\wedge (path_b(c) = path_a(c))$	

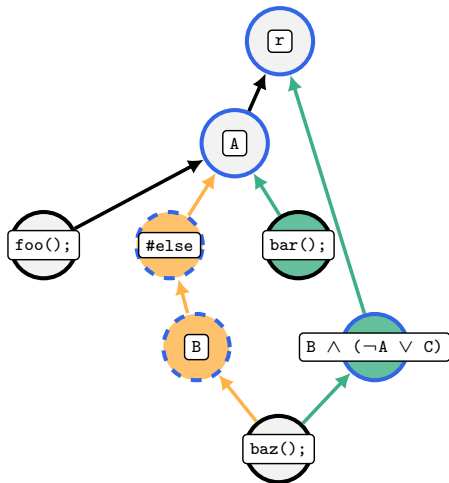
Custom classifications possible.



`foo();` is unchanged

`bar();` is **added** to feature A

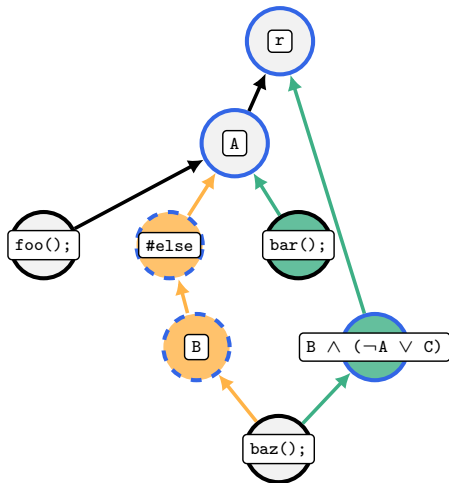
`baz();` is moved
 from $B \wedge \neg A$
 to $B \wedge (\neg A \vee C)$



`foo();` is unchanged

$AddToPC(\text{bar}();) = true$

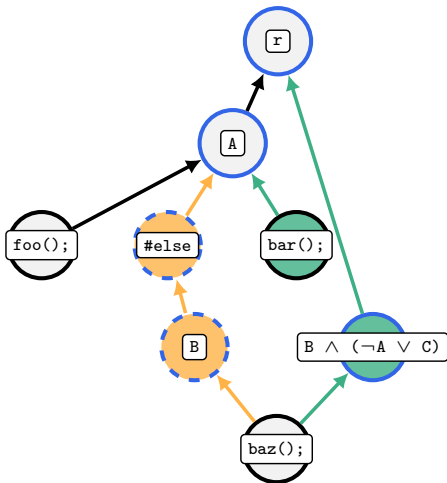
`baz();` is moved
 from $B \wedge \neg A$
 to $B \wedge (\neg A \vee C)$



$Untouched(\text{foo();}) = true$

$AddToPC(\text{bar();}) = true$

baz(); is moved
 from $B \wedge \neg A$
 to $B \wedge (\neg A \vee C)$



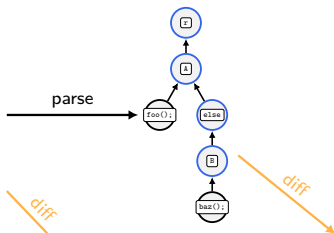
$$\text{Untouched}(\text{foo();}) = \text{true}$$

$$\text{AddToPC}(\text{bar();}) = \text{true}$$

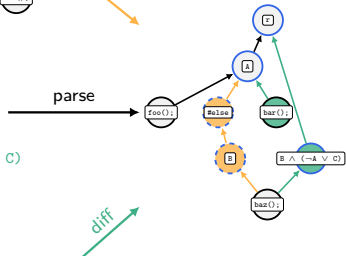
$$\text{Generalization}(\text{baz();}) = \text{true}$$

because $B \wedge \neg A \models B \wedge (\neg A \vee C)$

```
#ifdef A
foo();
#else
#ifdef B
baz();
#endif
#endif
```



```
#ifdef A
foo();
-#else
- #ifdef B
+ bar();
+#endif
+#if B && (!A || C)
baz();
- #endif
#endif
```



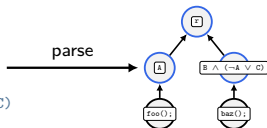
classify

$Untouched(\text{foo}()) = true$

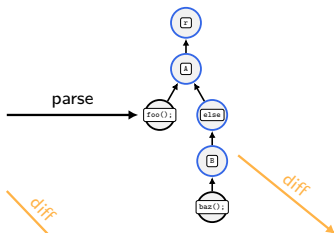
$AddToPC(\text{bar}()) = true$

$Generalization(\text{baz}()) = true$

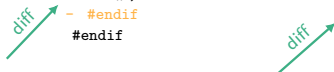
```
#ifdef A
foo();
bar();
#endif
#if B && (!A || C)
baz();
#endif
```



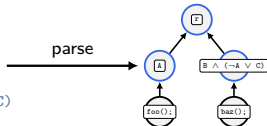
```
#ifdef A
foo();
#else
#ifdef B
baz();
#endif
#endif
```



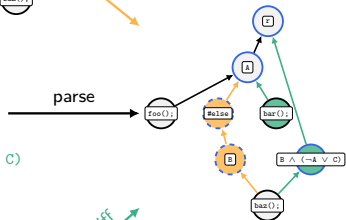
```
#ifdef A
foo();
-#else
- #ifdef B
+ bar();
+#endif
+#if B && (!A || C)
baz();
- #endif
#endif
```



```
#ifdef A
foo();
bar();
#endif
#if B && (!A || C)
baz();
#endif
```



Analytical Evaluation



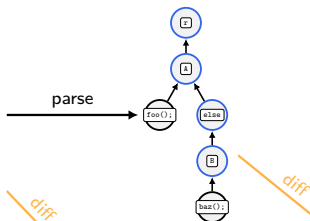
$Untouched(\text{foo}()) = true$

classify

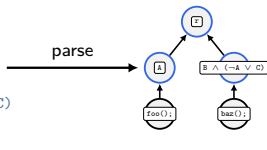
$AddToPC(\text{bar}()) = true$

$Generalization(\text{baz}()) = true$

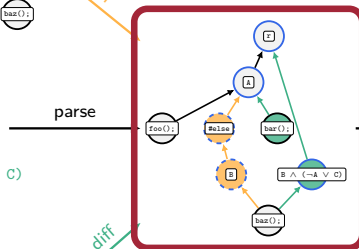
```
#ifdef A
foo();
#else
#ifdef B
baz();
#endif
#endif
```



```
#ifdef A
foo();
-#else
- #ifdef B
+ bar();
+#endif
+#if B && (!A || C)
baz();
- #endif
#endif
```



Analytical Evaluation



We prove
completeness ○
and soundness. □

$Untouched(\text{foo}()) = \text{true}$

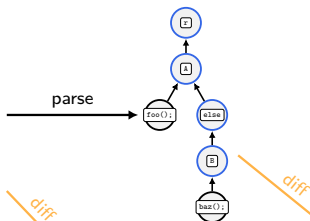
$AddToPC(\text{bar}()) = \text{true}$

$Generalization(\text{baz}()) = \text{true}$

```

#ifdef A
  foo();
#else
  #ifdef B
    baz();
  #endif
#endif

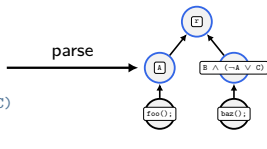
```



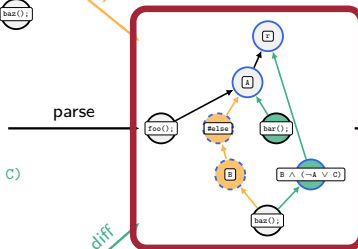
```

#ifdef A
  foo();
-#else
- #ifdef B
+ bar();
+ #endif
+ #if B && (!A || C)
  baz();
- #endif
#endif

```



Analytical Evaluation



We prove
completeness and
soundness.

classify

$Untouched(\text{foo}()) = true$

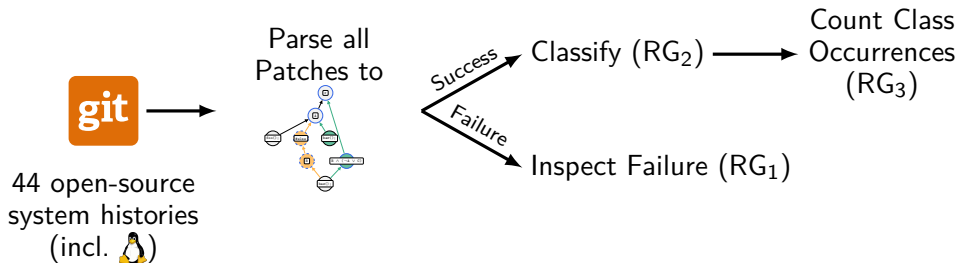
$AddToPC(\text{bar}()) = true$

$Generalization(\text{baz}()) = true$

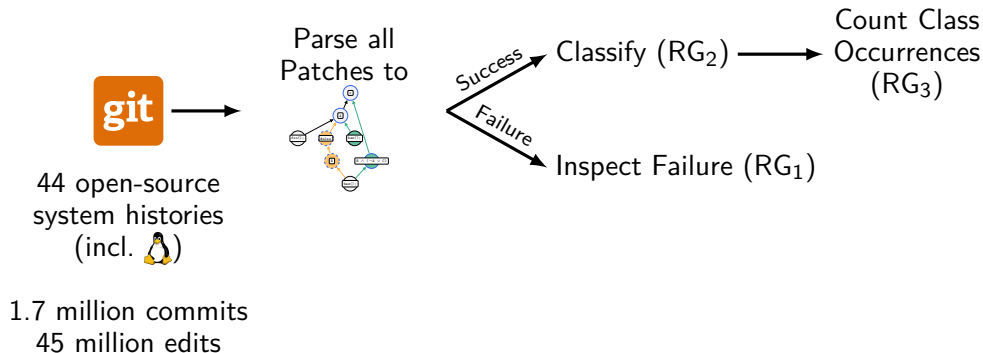
We prove
completeness and
unambiguity .



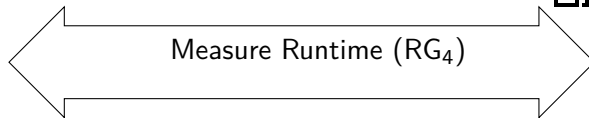
From Theory to Practice




From Theory to Practice



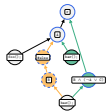
From Theory to Practice



44 open-source
system histories
(incl. )

1.7 million commits
45 million edits

Parse all
Patches to



Success
Failure

Classify (RG_2)

Inspect Failure (RG_1)

Count Class
Occurrences
(RG_3)

From Theory to Practice

RG₁ Variation Diffs Validate completeness of variation diffs.

From Theory to Practice

RG₁ Variation Diffs Result

Validate completeness of variation diffs.

All patches with syntactically correct variability annotations
can be parsed (99.82%). ✓ ⇒ ○

From Theory to Practice

RG₁ Variation Diffs **Result**

Validate completeness of variation diffs.
All patches with syntactically correct variability annotations
can be parsed (99.82%). ✓ ⇒ ○

RG₂ Classification

Validate completeness and unambiguity of classification.

From Theory to Practice

RG₁ Variation Diffs Result

Validate completeness of variation diffs.
All patches with syntactically correct variability annotations
can be parsed (99.82%). ✓ ⇒ ○

RG₂ Classification Result

Validate completeness and unambiguity of classification.
All edits were assigned exactly one class. ✓ ⇒ ○ ∧ ○○

From Theory to Practice

RG₁ Variation Diffs Result

Validate completeness of variation diffs.
All patches with syntactically correct variability annotations
can be parsed (99.82%). ✓ ⇒ ○

RG₂ Classification Result

Validate completeness and unambiguity of classification.
All edits were assigned exactly one class. ✓ ⇒ ○ ∧ ○○

RG₃ Relevancy

Validate that our edit classes are relevant (i.e., all classes
occur in practice).

From Theory to Practice

RG₁ Variation Diffs **Result**

Validate completeness of variation diffs.
All patches with syntactically correct variability annotations
can be parsed (99.82%). ✓ ⇒ ○

RG₂ Classification **Result**

Validate completeness and unambiguity of classification.
All edits were assigned exactly one class. ✓ ⇒ ○ ∧ ○○

RG₃ Relevancy **Result**

Validate that our edit classes are relevant (i.e., all classes
occur in practice).
All classes occur in practice (91,000 to 22 million
occurrences). ✓

From Theory to Practice

RG₁ Variation Diffs **Result**

Validate completeness of variation diffs.
All patches with syntactically correct variability annotations can be parsed (99.82%). ✓ ⇒ ○

RG₂ Classification **Result**

Validate completeness and unambiguity of classification.
All edits were assigned exactly one class. ✓ ⇒ ○ ∧ ○○

RG₃ Relevancy **Result**

Validate that our edit classes are relevant (i.e., all classes occur in practice).
All classes occur in practice (91,000 to 22 million occurrences). ✓

RG₄ Scalability

Validate that edit classification can be automated and scales.

From Theory to Practice

RG₁ Variation Diffs **Result**

Validate completeness of variation diffs.
All patches with syntactically correct variability annotations can be parsed (99.82%). ✓ ⇒ ○


RG₂ Classification **Result**

Validate completeness and unambiguity of classification.
All edits were assigned exactly one class. ✓ ⇒ ○ ∧ ○○

RG₃ Relevancy **Result**

Validate that our edit classes are relevant (i.e., all classes occur in practice).
All classes occur in practice (91,000 to 22 million occurrences). ✓

RG₄ Scalability **Result**

Validate that edit classification can be automated and scales.
99.89% of commits processed in < 1s with 7ms/commit as median. ✓ ⇒ 

```

#ifdef A
    foo();
#else
    #ifdef B
        baz();
    #endif
#endif

```

diff

```

#ifdef A
    foo();
-#else
-    #ifdef B
+    bar();
+#endif
+#if B && (!A || C)
    baz();
-    #endif
#endif

```

diff

```

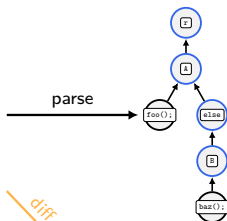
#ifdef A
    foo();
    bar();
#endif
#if B && (!A || C)
    baz();
#endif

```

```

#ifdef A
    foo();
#else
    #ifdef B
        baz();
    #endif
#endif

```



```

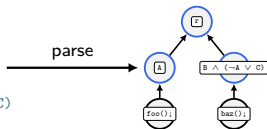
#ifdef A
    foo();
-#else
-    #ifdef B
+    bar();
+#endif
+#if B && (!A || C)
    baz();
-    #endif
#endif

```

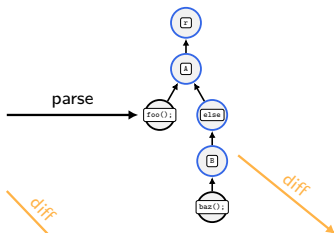
```

#ifdef A
    foo();
    bar();
#endif
#if B && (!A || C)
    baz();
#endif

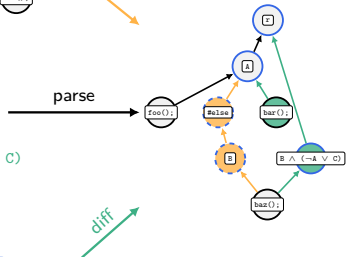
```



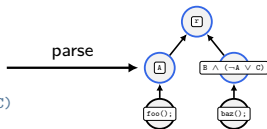
```
#ifdef A
foo();
#else
#ifdef B
baz();
#endif
#endif
```



```
#ifdef A
foo();
-#else
- #ifdef B
+ bar();
+#endif
+#if B && (!A || C)
baz();
- #endif
#endif
```



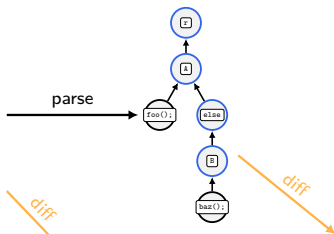
```
#ifdef A
foo();
bar();
#endif
#if B && (!A || C)
baz();
#endif
```



```

#ifdef A
  foo();
#else
  #ifdef B
    baz();
  #endif
#endif

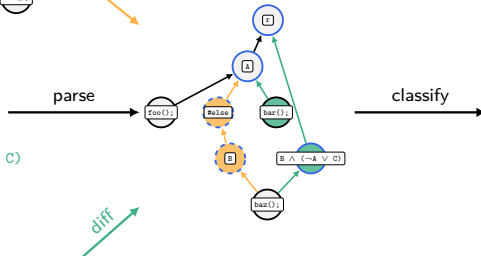
```



```

#ifdef A
  foo();
-#else
- #ifdef B
+ bar();
+#endif
+#if B && (!A || C)
  baz();
- #endif
#endif

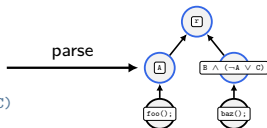
```



```

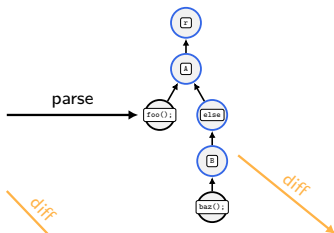
#ifdef A
  foo();
  bar();
#endif
#if B && (!A || C)
  baz();
#endif

```

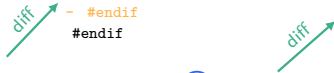


classify

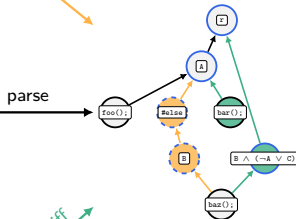
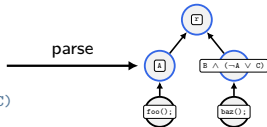
```
#ifdef A
foo();
#else
#ifdef B
baz();
#endif
#endif
```



```
#ifdef A
foo();
-#else
- #ifdef B
+ bar();
+ #endif
+ #if B && (!A || C)
baz();
- #endif
#endif
```



```
#ifdef A
foo();
bar();
#endif
#if B && (!A || C)
baz();
#endif
```



classify

Complete



Unambiguous



Automated



Backup Slides

Use Cases

Test Case Evolution

Code Understanding

Incremental Analyses

Variation Control

Managed Clone-and-Own

Detect Unintended Changes

...

Software projects are heterogeneous



Source Code



Build Files



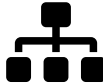
Other Resources

All identifiable *artifacts* within a software project might be subject to variability.

Variability comes in many forms

```
#ifdef A
    foo();
#else
    #ifdef B
        baz();
    #endif
#endif
```

Variability Annotations
(in different dialects)

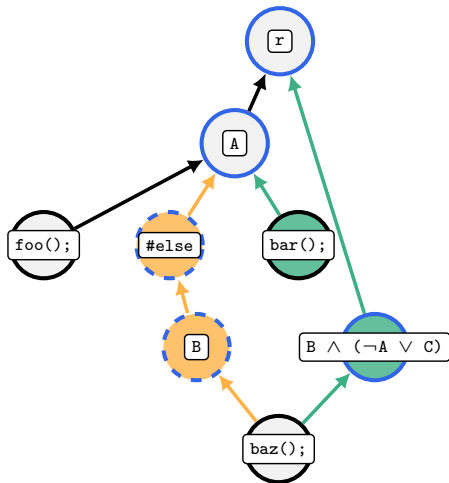


Frameworks



Language Extensions

Commonality: Each artifact is identified with its configuration options.




We prove that variation diffs are

complete (i.e., any edit to a variation tree can be described as a variation diff) and

sound (i.e., any variation diff describes an edit to a variation tree)



giving us a complete and sound model for edits to variability.

-  Al-Hajjaji, M., Benduhn, F., Thüm, T., Leich, T., and Saake, G. (2016).
Mutation Operators for Preprocessor-Based Variability.
In Proc. Int'l Workshop on Variability Modelling of Software-Intensive Systems (VaMoS), pages 81–88. ACM.
-  Borba, P., Teixeira, L., and Gheyi, R. (2012).
A Theory of Software Product Line Refinement.
Theoretical Computer Science, 455(0):2–30.
-  Ji, W., Berger, T., Antkiewicz, M., and Czarnecki, K. (2015).
Maintaining Feature Traceability with Embedded Annotations.
In Proc. Int'l Systems and Software Product Line Conf. (SPLC), pages 61–70. ACM.
-  Passos, L., Teixeira, L., Dintzner, N., Apel, S., Wąsowski, A., Czarnecki, K., Borba, P., and Guo, J. (2016).
Coevolution of Variability Models and Related Software Artifacts.
Empirical Software Engineering (EMSE), 21(4).
-  Stănciulescu, S., Berger, T., Walkingshaw, E., and Wąsowski, A. (2016).
Concepts, Operations, and Feasibility of a Projection-Based Variation Control System.
In Proc. Int'l Conf. on Software Maintenance and Evolution (ICSME), pages 323–333. IEEE.