



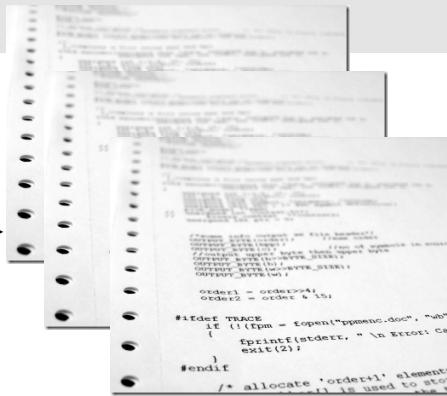
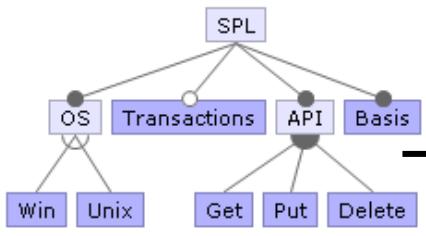
Technische  
Universität  
Braunschweig



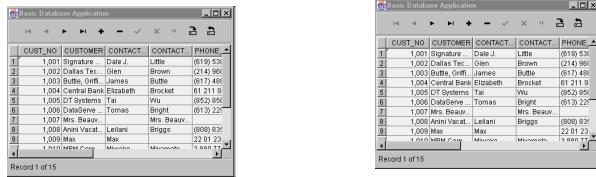
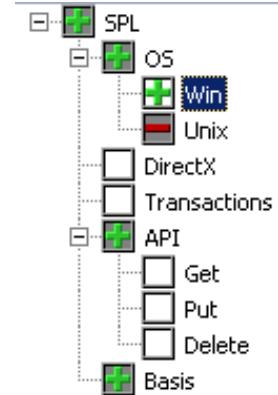
# Program Slicing in the Presence of Variability

Sandro Schulze, Frederik Kanning, FOSD meeting, 4.-7.05.2014

# Problem



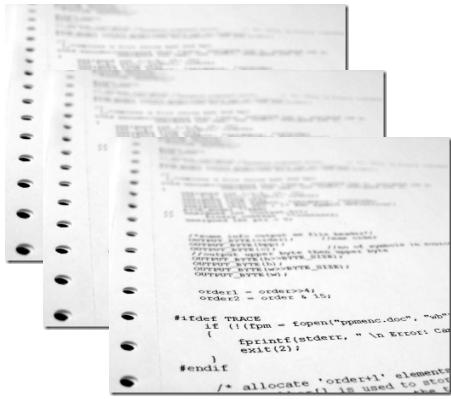
# Testing: incremental pairwise CIT



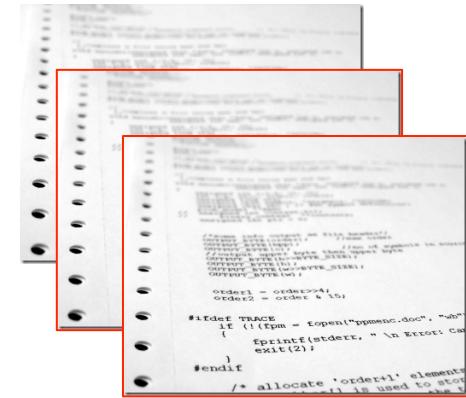
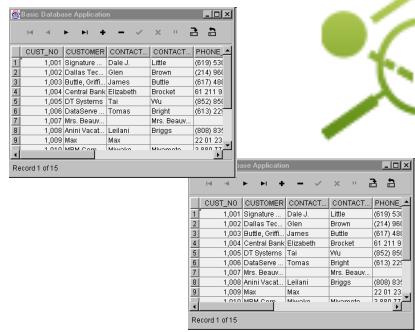
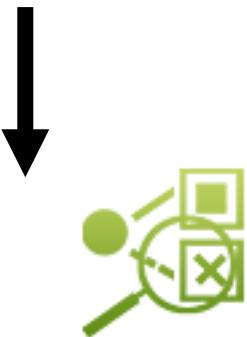
# Motivation

## Change

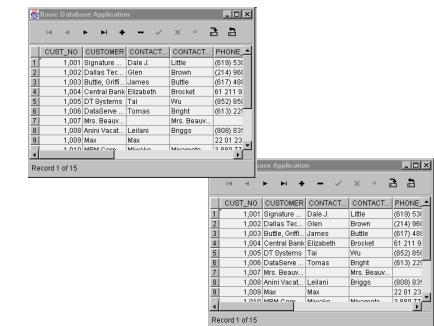
e.g., bug fix, feature request



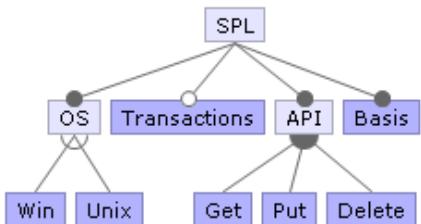
V 1.0



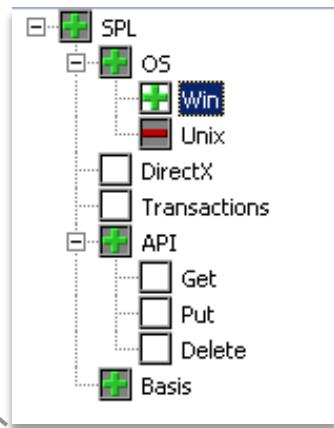
V2.0



# Determining the Impact of Change



Features affected by a certain change?



Variants affected by a certain change?



Which tests have to be re-run?  
→ **Regression Testing**

# Program Slicing (Mark Weiser, 1984)

```
1  read(text);
2  read(n);
3  lines = 1;
4  chars = 1;
5  subtext = "";
6  c = getChar(text);
7  while (c != \eof)
8      if (c == \n)
9          then lines = lines + 1;
10         chars = chars + 1;
11     else chars = chars + 1;
12     if (n != 0)
13         then subtext = subtext ++ c;
14         n = n - 1;
15     c = getChar(text);
16 write(lines);
17 write(chars);
18 write(subtext);
```

What program statements can influence specified variables at a certain statement?

Slicing criterion:

<16, {lines}>



# Static Forward Slicing

```
1 void main()
2 {
3     int a, b, c;          Slicing criterion
4
5     a = 32;
6
7     b = 23;
8     c = 2 * b; ←
9
10    print(a);
11    print(c); ←
12
13    a += 1;
14    c = b; ←
15
16    print(a);
17    print(c); ←
18 }
```

....determining how a modification affects (part of) a program.

# Variability-Aware Forward Slicing

```
1 void main()
2 {
3     int a, b, c;
4
5     a = 32;
6
7     b = 23;
8     c = 2 * b;
9
10    print(a);
11    print(c);
12
13    a += 1;
14    c = b;
15
16    print(a);
17    print(c);
18 }
```

Slicing with  
preprocessors

```
4 // lines 1-3 omitted
5
6 a = 32;
7 b = 23;
8 c = 2 * b;
9
10 print(a);
11 print(c);
12
13 #ifdef X
14     a = b;
15     #ifdef Z
16         c = b;
17     #endif
18 #else
19     a = 2*b;
20 #endif
21
22 print(a);
23 print(c);
24 }
```



# General Approach

- Data flow dependence
  - Compute Def-Use-Chains via reachable uses analysis
- Control dependence analysis
  - How does dependent predicates affect definitions in subordinate branches?
- Taking interdependence into account
- Slice → transitive closure of all dependencies
- Based on TypeChef and Monotone framework

# Data Flow Dependence

```
// lines 1-3 omitted
4   a = 32;
5   b = 23;
6   c = 2 * b;           ← Slicing criterion
7
8
9   print(a);
10  print(c);           ←
11
12 #ifdef X
13   a = b;             ← {X}
14   #ifdef Z
15     c = b;           ← {X && Z}
16   #endif
17 #else
18   a = 2*b;           ← {!X}
19 #endif
20   print(a);          ← {!X || X}
21   print(c);           ←
22 }
```

Which features are affected if there is a change in the respective statement?

Under which configurations particular statements belong to the slice?



# Control Flow Dependence

```
1 void control()    Slicing criterion  
2 {                  <5, {x}>  
3     int x, y, a = 0;  
4  
5     x = 0;  
6  
7     if(y < 5) {  
8         y += 1;  
9     #ifdef X  
10        a = 2*x;   ← {X}  
11    #endif  
12    }  
13  
14    print(x);    ←  
15    print(y);  
16 }
```

```
1 void control()    Slicing criterion  
2 {  
3     int x, y, a = 0;  
4  
5     x = 0; // ←  
6  
7     if(x < 5) { ←  
8 #ifdef Y  
9         y += a;   ← {Y}  
10 #endif  
11 #ifdef X  
12         a += 1;   ← {X}  
13 #endif  
14     }  
15  
16     print(x); ←  
17     print(y); ← {Y}  
18 }
```

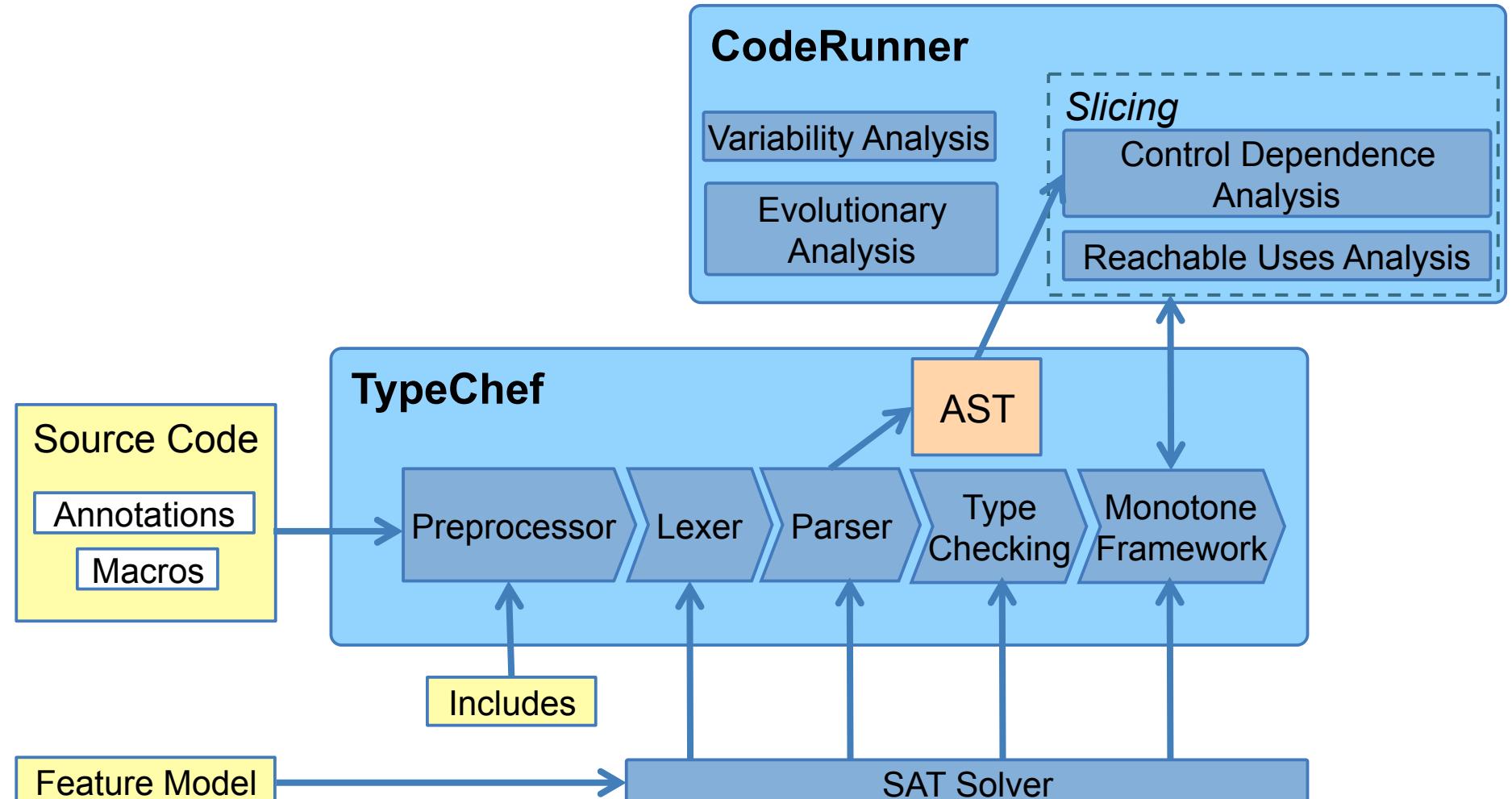
# Collecting all dependencies

```
1 void loop() {  
2     int a = 0; int b = 0; int c = 0;  
3     int x;          Slicing criterion  
4  
5     x = 5; // <-      <5, {x}>  
6  
7     while(foo() == true)  
8     {  
9 #ifdef X  
10        c = b;           ← {X}  
11 #endif  
12        b = a;           ← {X || Z}  
13 #if defined(X) || defined(Z)  
14        a = x;           ← {X || Z}  
15 #endif  
16    }  
17  
18    print(c);          ← {X}  
19    print(x);          ← {X}  
20 }
```

Iterating through all indirect dependencies.



# Infrastructure



# A Long Road To Go...



We are  
probably here



Technische  
Universität  
Braunschweig

Sandro Schulze | Program Slicing & Variability | FOSD 2014| Slide 13



# Current State & Ongoing Work

Currently implemented/Ongoing:

- Data flow dependency analysis (on statement level)
- Control dependence (work in progress)
- Changes to function parameters (work in progress)

Future work

- Support more tricky constructs → pointers, go-to
- *Interprocedural* forward slicing
- Optimizations (performance, resolving variability)
- Evaluation wrt regression testing

# Ultimate Goal

## Interprocedural, variability-aware slicing

