

# Feature Trace Recording



Paul Maximilian  
Bittner<sup>1</sup>



Alexander  
Schultheiß<sup>2</sup>



Thomas  
Thüm<sup>1</sup>



Timo  
Kehrér<sup>2</sup>



Jeffrey M.  
Young<sup>3</sup>



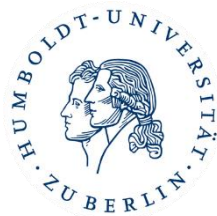
Lukas  
Linsbauer<sup>4</sup>

1



universität  
**uulm**

2



3



**Oregon State**  
University

4







**One  
Eternity  
Later**



**WHICH PART OF THE CODE**

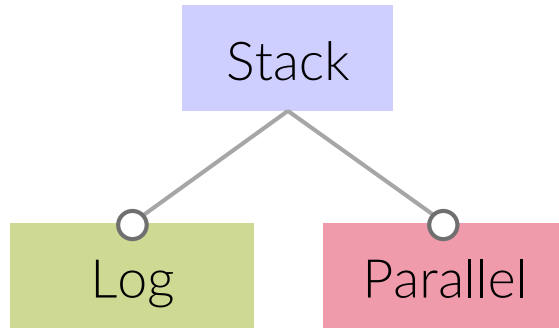


**IS IMPLEMENTING THAT?**

# Feature Traceability Problem

Feature Traceability is the knowledge  
where each feature is implemented.

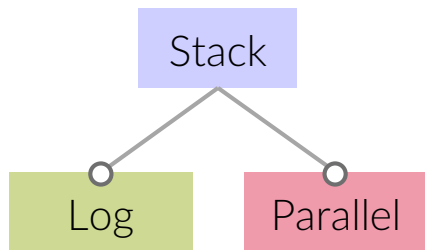
# Software Product Lines – Problem Solved?



```
class Stack {  
    void push(int x) {  
        log("Push");  
        lock();  
        storage[head++] = x;  
        unlock();  
    }  
}
```

Not yet: *Software product lines*

- require education and tools,
- are a long-term investment with high initial costs.

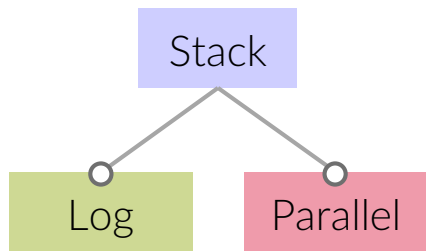


complete  
feature traces

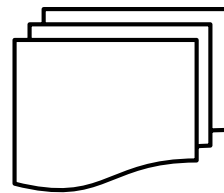
Not yet: *Software product lines*

- require education and tools,
- are a long-term investment with high initial costs.

In practice variability is often implemented via *clone-and-own*.



complete  
feature traces



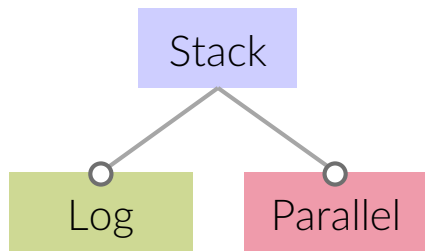
(almost) no  
feature traces



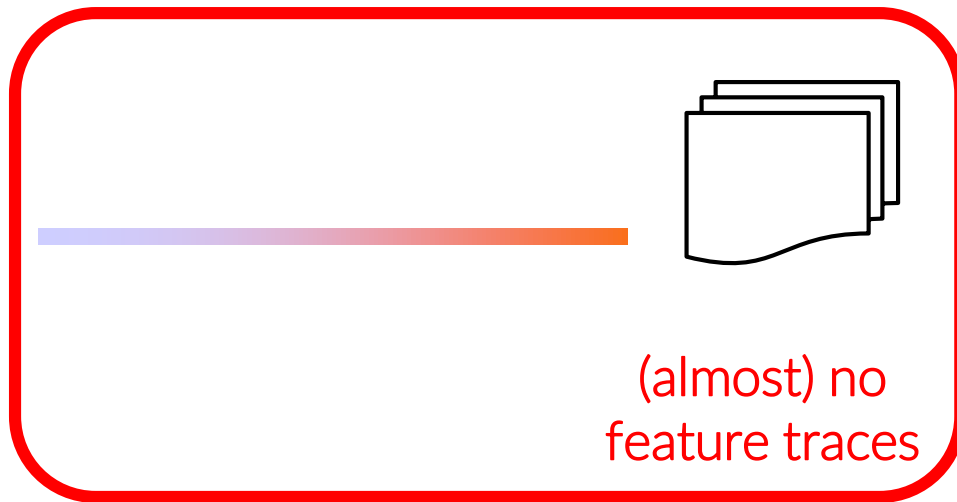
Not yet: *Software product lines*

- require education and tools,
- are a long-term investment with high initial costs.

In practice variability is often implemented via *clone-and-own*.



complete  
feature traces



So how can we help developers to *document and maintain* feature traces here?

# Feature traces can be documented ...

**Retroactively:** after development (Feature Location, Variability Mining)  
separate step in workflow  
not always possible because knowledge is lost

**Proactively:** during development (Embedded Annotations [Ji et al.])  
manual  
→ our contribution: semi-automation

# Example of Feature Trace Recording



```
class Stack {  
  
    /* ... */  
  
    void pop() {  
        storage[head--] = null;  
    }  
}
```

# Example of Feature Trace Recording

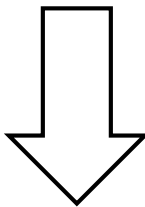
**pop** crashes  
when the stack  
is empty!



```
class Stack {  
    /* ... */  
  
    void pop() {  
        storage[head--] = null;  
    }  
}
```

# Example of Feature Trace Recording

```
void pop() {  
    storage[head--] = null;  
}
```



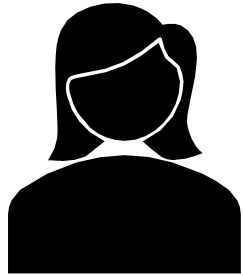
```
void pop() {  
    if (!empty()) {}  
    storage[head--] = null;  
}
```



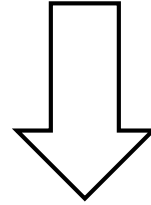


# Example of Feature Trace Recording

I only want  
this check in  
**Debug** mode.



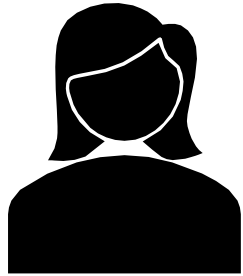
```
void pop() {  
    storage[head--] = null;  
}
```



```
void pop() {  
    if (!empty()) {}  
    storage[head--] = null;  
}
```

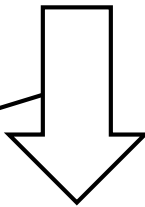
# Example of Feature Trace Recording

I only want  
this check in  
Debug mode.



feature  
context  
=  
Debug

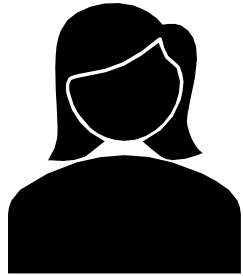
```
void pop() {  
    storage[head--] = null;  
}
```



```
void pop() {  
    if (!empty()) {}  
    storage[head--] = null;  
}
```

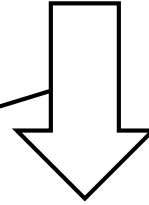
# Example of Feature Trace Recording

I only want  
this check in  
Debug mode.



feature  
context  
=  
Debug

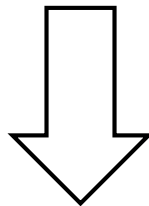
```
void pop() {  
    storage[head--] = null;  
}
```



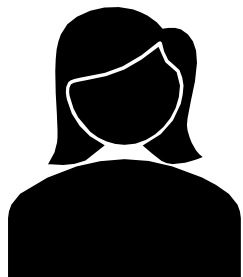
```
void pop() {  
    if (!empty()) {}  
    storage[head--] = null;  
}
```

# Example of Feature Trace Recording

```
void pop() {  
    if (!empty()) {}  
    storage[head--] = null;  
}
```

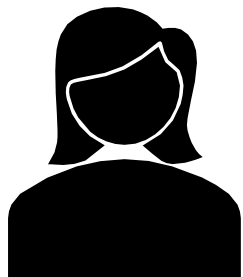


```
void pop() {  
    if (!empty()) {  
        storage[head--] = null;  
    }  
}
```

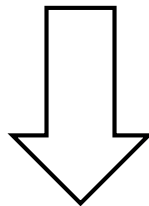


# Example of Feature Trace Recording

I don't know  
the feature.



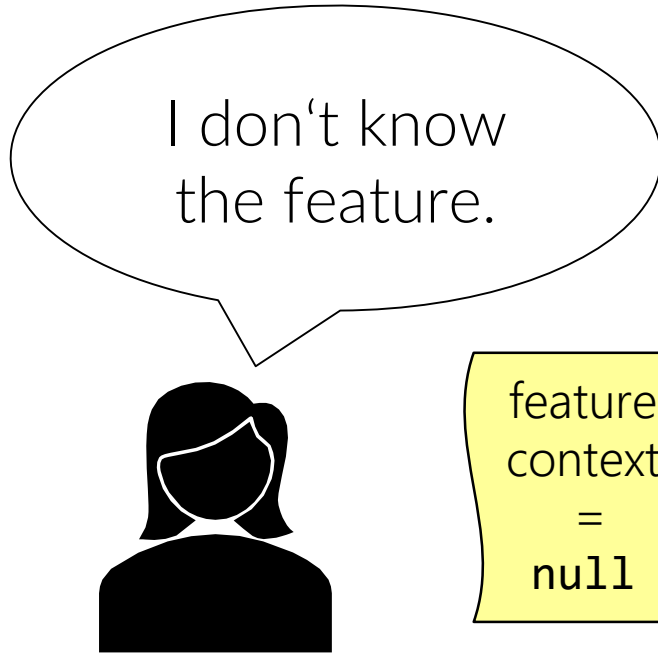
```
void pop() {  
    if (!empty()) {}  
    storage[head--] = null;  
}
```



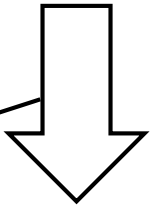
```
void pop() {  
    if (!empty()) {  
        storage[head--] = null;  
    }  
}
```



# Example of Feature Trace Recording



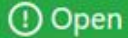
```
void pop() {  
    if (!empty()) {}  
    storage[head--] = null;  
}
```



```
void pop() {  
    if (!empty()) {  
        storage[head--] = null;  
    }  
}
```

# Example of Feature Trace Recording – The Next Week

## Stacks should be immutable #1

[Edit](#)[New issue](#)

pmbittner opened this issue now · 0 comments



pmbittner commented now



We like functional programming now!

Assignees

 Alice



# Example of Feature Trace Recording – The Next Week

Stacks should be immutable #1

New issue

Open

pmbittner opened this issue now · 0 comment



pmbittner commented now

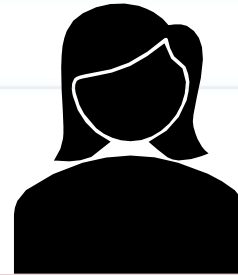
We like functional programming now!

Ok, I am working  
on

feature  
context

=

Functional



```
void pop() {  
    if (!empty()) {  
        storage[head--] = null;  
    }  
}
```

delete

feature  
context  
=  
Functional

```
void pop() {  
    if (!empty()) {  
    }  
}
```

```
void pop() {
    if (!empty()) {
        storage[head--] = null;
    }
}
```

delete

feature  
context

=

Functional

```
void pop() {
    if (!empty()) {
    }
}
```

insert

```
void pop() {
    Stack<T> c = clone();
    if (!empty()) {
        c.storage[c.head--] = null;
    }
    return c;
}
```



```
void pop() {
    if (!empty()) {
        storage[head--] = null;
    }
}
```

delete

```
void pop() {
    if (!empty()) {
    }
}
```

feature  
context  
=  
Functional

insert

```
Stack<T> pop() {
    Stack<T> c = clone();
    if (!empty()) {
        c.storage[c.head--] = null;
    }
    return c;
}
```

update

```
void pop() {
    Stack<T> c = clone();
    if (!empty()) {
        c.storage[c.head--] = null;
    }
    return c;
}
```

```
void pop() {
    if (!empty()) {
        storage[head--] = null;
    }
}
```

delete

```
void pop() {
    if (!empty()) {
    }
}
```

insert

done with single

feature  
context  
=  
Functional

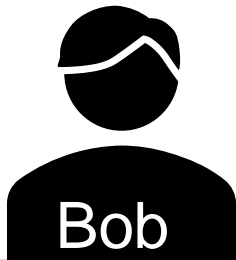
update

```
Stack<T> pop() {
    Stack<T> c = clone();
    if (!empty()) {
        c.storage[c.head--] = null;
    }
    return c;
}
```

```
void pop() {
    Stack<T> c = clone();
    if (!empty()) {
        c.storage[c.head--] = null;
    }
    return c;
}
```

Hey Alice, can I merge your changes?

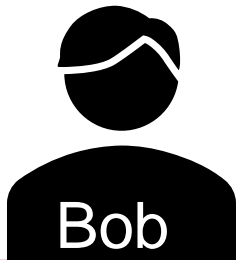
Sure! 😊



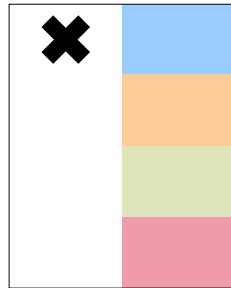
Hey Alice, can I merge your changes?

Sure! 😊

But I have another variant!

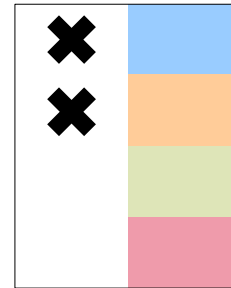


Bob



Debug  
Functional

⋮



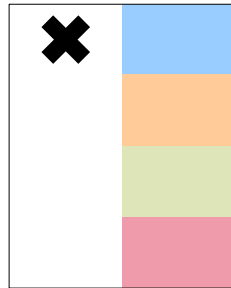
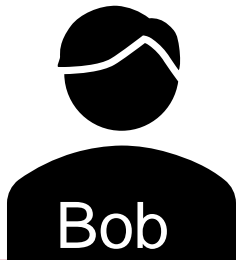
Alice

Hey Alice, can I merge your changes?

Sure! 😊

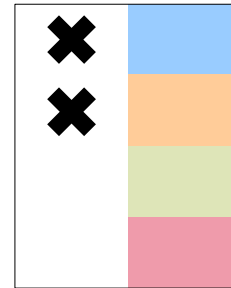
But I have another variant!

That's fine, I recorded all  
feature traces!



Debug  
Functional

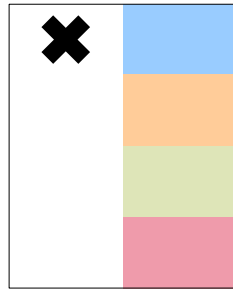
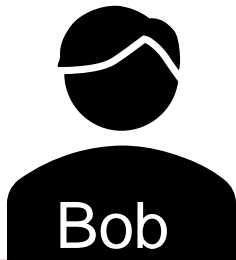
⋮





```
void pop() { /*...*/ }
```

```
void pop() { /*...*/ }
```



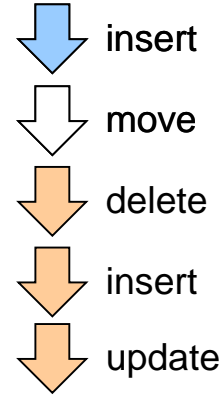
Debug  
Functional

⋮

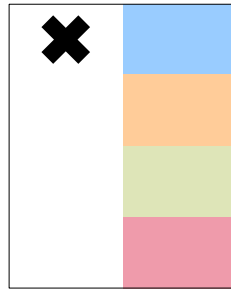
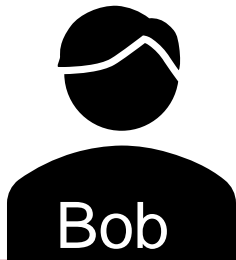


```
void pop() { /* ... */ }
```

```
void pop() { /* ... */ }
```



```
Stack<T> pop() { /* ... */ }
```



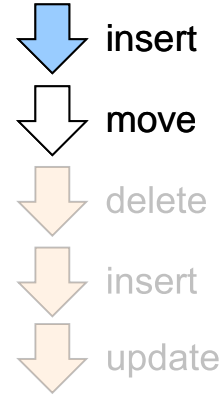
Debug  
Functional

⋮

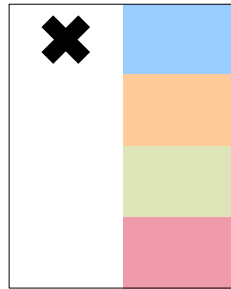
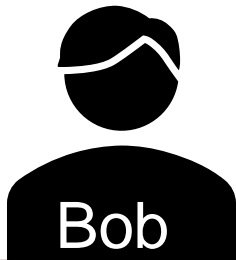


```
void pop() { /* ... */ }
```

```
void pop() { /* ... */ }
```



```
Stack<T> pop() { /* ... */ }
```

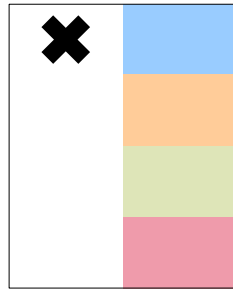
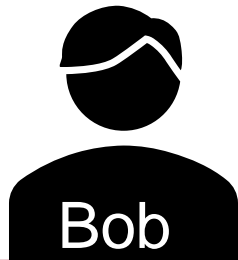
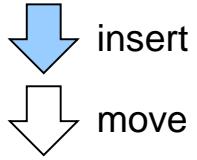


Debug  
Functional

⋮



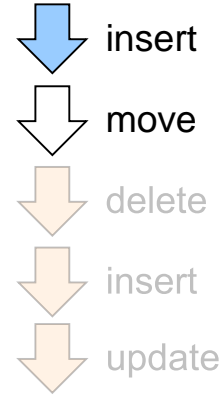
```
void pop() { /* ... */ }
```



Debug  
Functional

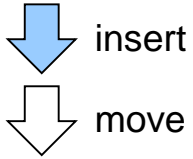
⋮

```
void pop() { /* ... */ }
```

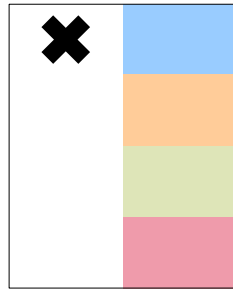
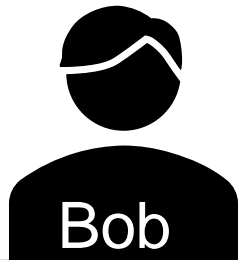


```
Stack<T> pop() { /* ... */ }
```

```
void pop() { /*...*/ }
```



```
void pop() {  
    if (!empty()) {  
        storage[head--] = null;  
    }  
}
```

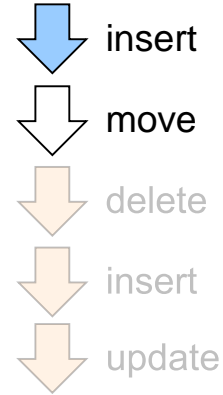


Debug  
Functional

⋮

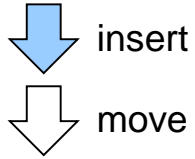


```
void pop() { /*...*/ }
```



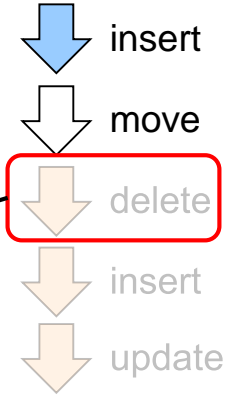
```
Stack<T> pop() { /*...*/ }
```

```
void pop() { /*...*/ }
```

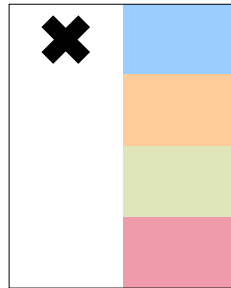
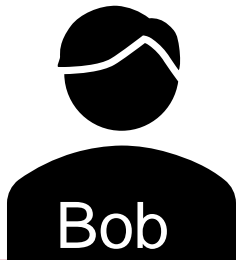


```
void pop() {  
  if (!empty()) {  
    storage[head--] = null;  
  }  
}
```

```
void pop() { /*...*/ }
```



```
Stack<T> pop() { /*...*/ }
```

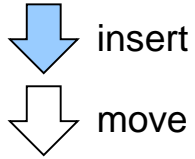


Debug  
Functional

⋮



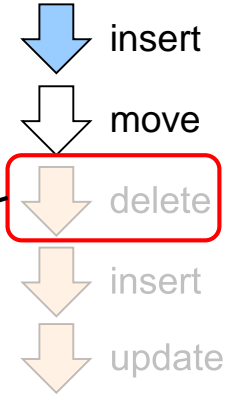
```
void pop() { /*...*/ }
```



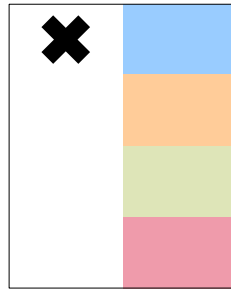
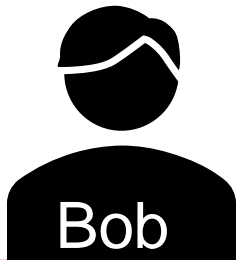
```
void pop() {  
    if (!empty()) {  
        storage[head--] = null;  
    }  
}
```

¬Functional

```
void pop() { /*...*/ }
```



```
Stack<T> pop() { /*...*/ }
```

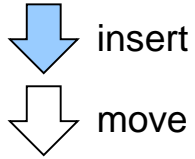


Debug  
Functional

⋮



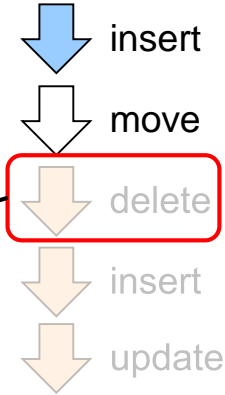
```
void pop() { /*...*/ }
```



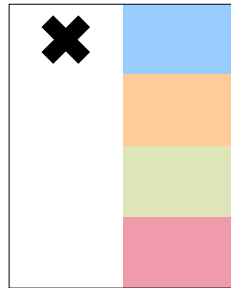
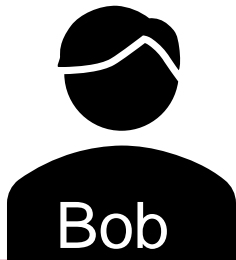
```
void pop() {  
    if (!empty()) {  
        storage[head--] = null;  
    }  
}
```

¬Functional

```
void pop() { /*...*/ }
```



```
Stack<T> pop() { /*...*/ }
```



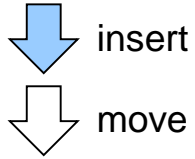
Debug  
Functional

⋮





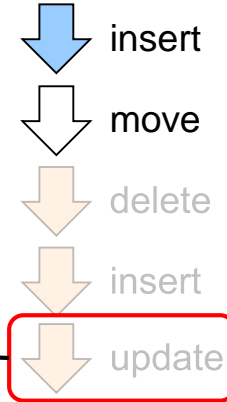
```
void pop() { /*...*/ }
```



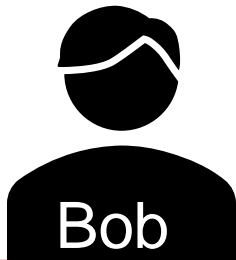
```
void pop() {  
  if (!empty()) {  
    storage[head--] = null;  
  }  
}
```

→Functional

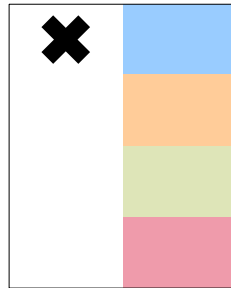
```
void pop() { /*...*/ }
```



```
Stack<T> pop() { /*...*/ }
```



Bob



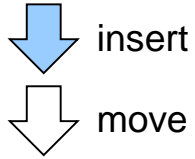
Debug  
Functional

⋮



Alice

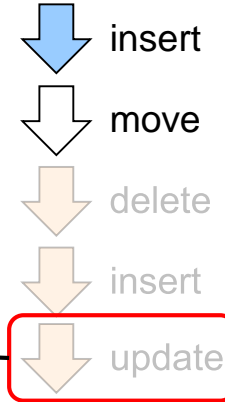
```
void pop() { /*...*/ }
```



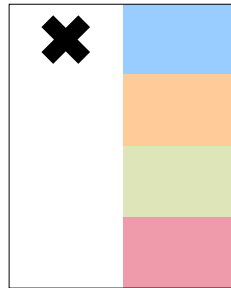
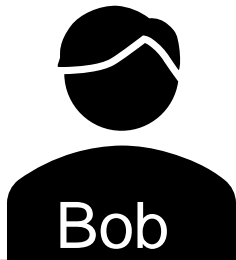
```
void pop() {  
  if (!empty()) {  
    storage[head--] = null;  
  }  
}
```

→Functional

```
void pop() { /*...*/ }
```



```
Stack<T> pop() { /*...*/ }
```

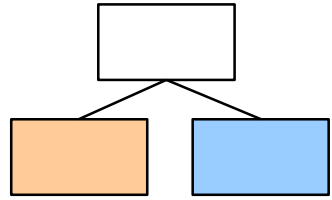


Debug  
Functional

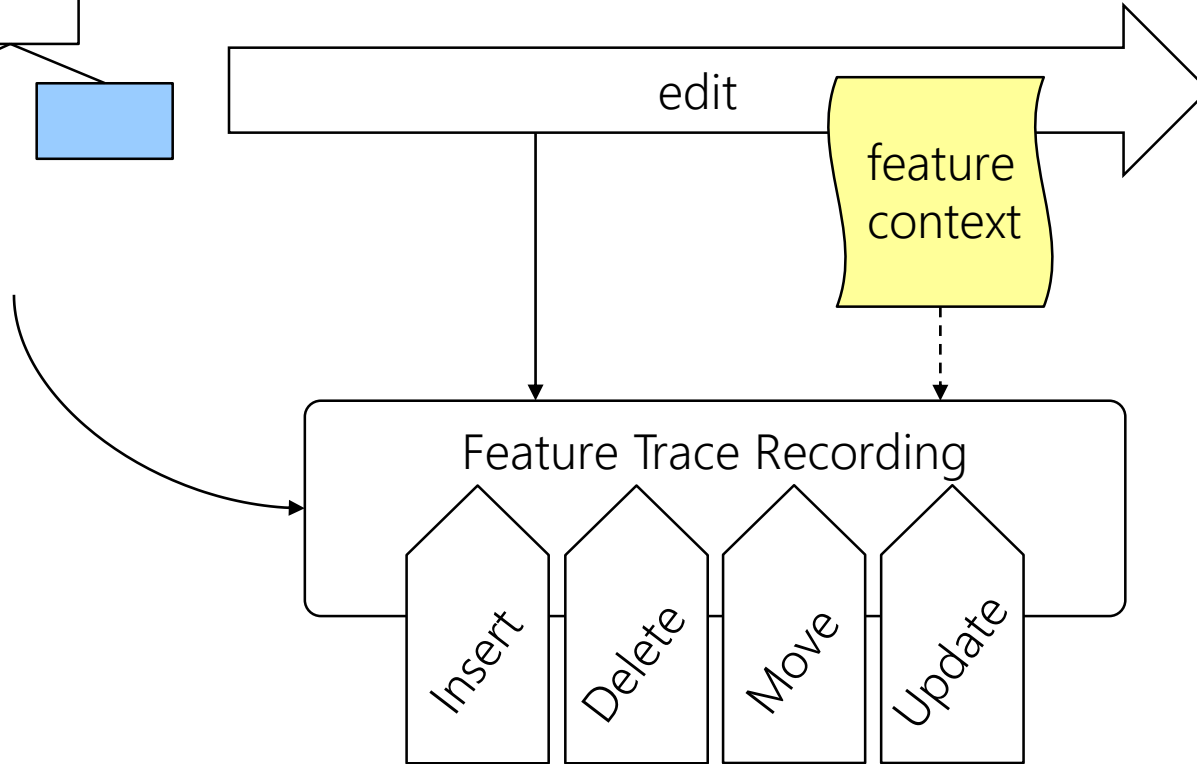
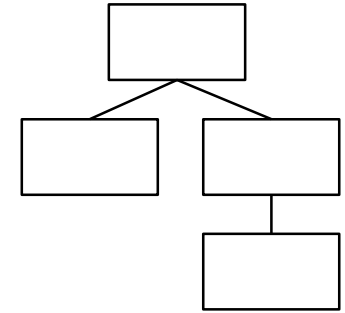
⋮



old code version

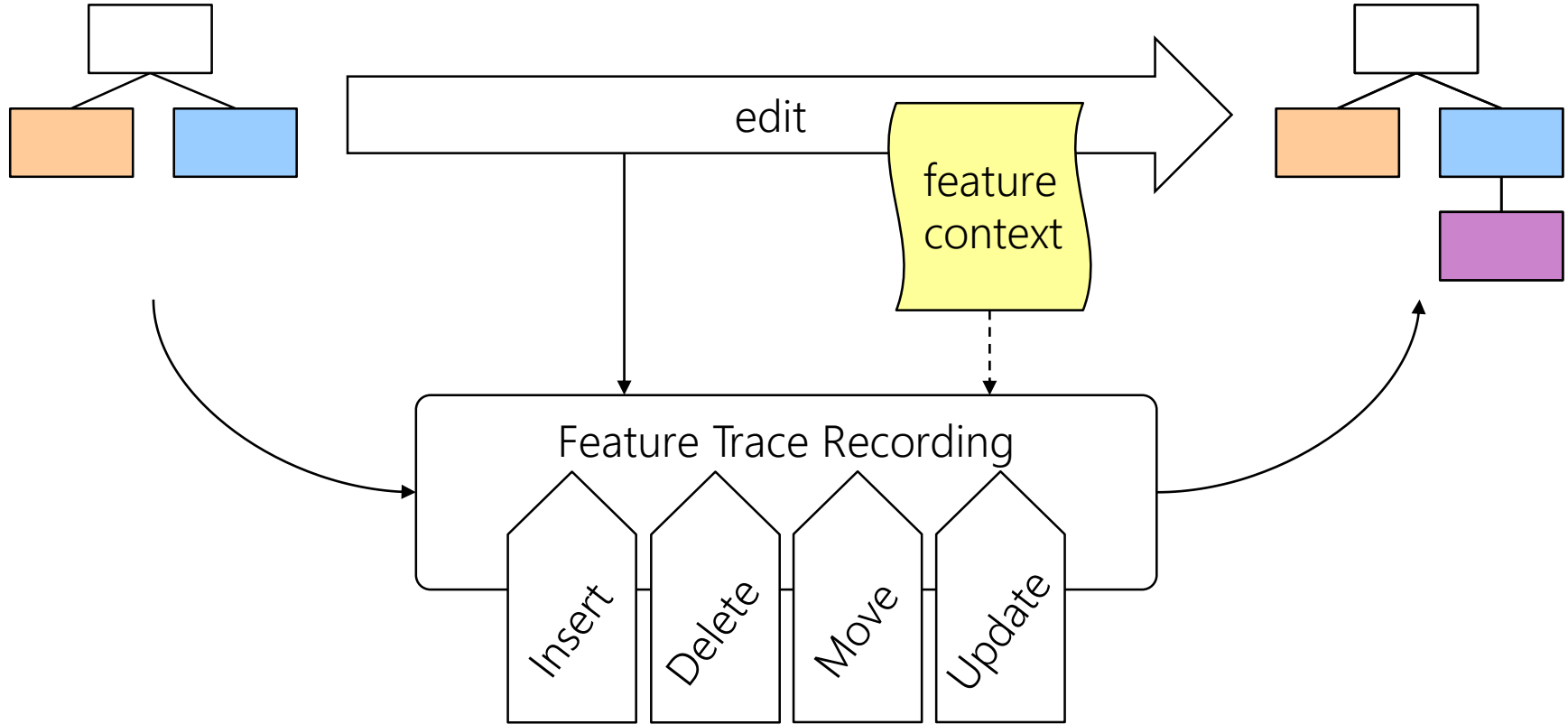


new code version

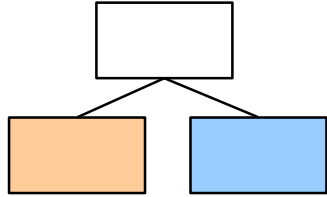


old code version

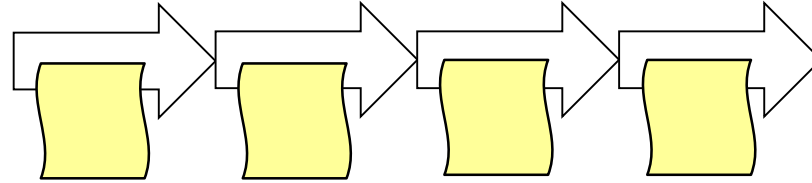
new code version



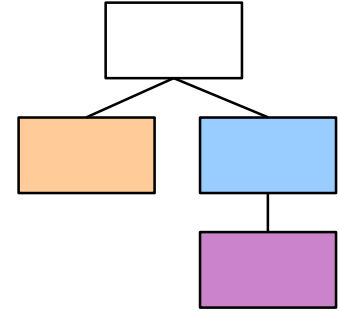
# To evaluate feature trace recording we need ...



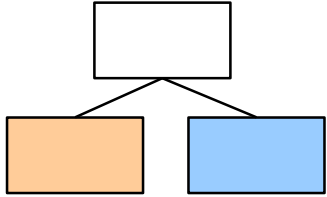
edits (e.g., derived from commit history)



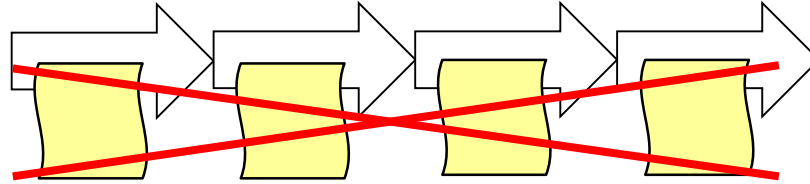
feature contexts



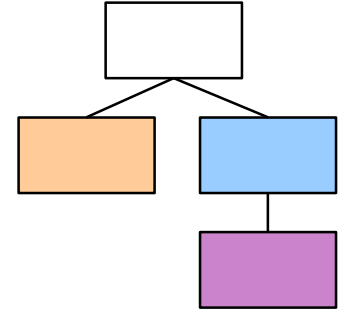
# To evaluate feature trace recording we need ...



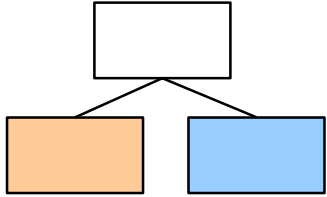
edits (e.g., derived from commit history)



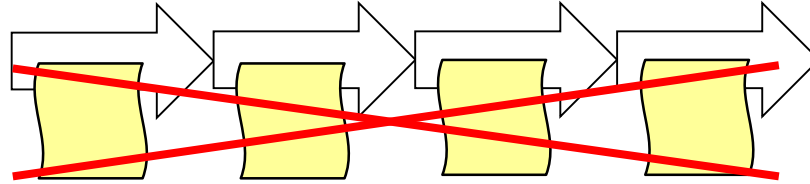
feature contexts  
not available



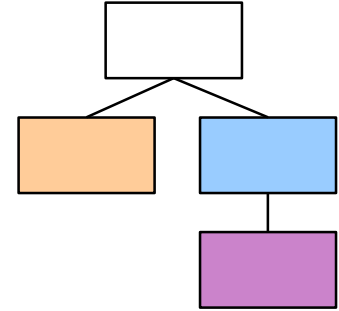
# To evaluate feature trace recording we need ...



edits (e.g., derived from commit history)



feature contexts  
not available



use software product line  
to derive feature contexts

# Can we reproduce typical edits to variability?

## Concepts, Operations, and Feasibility of a Projection-Based Variation Control System

Stefan Stănciulescu	Thorsten Berger	Eric Walkingshaw	Andrzej Wasowski
IT University of Copenhagen	Chalmers University of Gothenburg	Oregon State University	IT University of Copenhagen
Denmark	Sweden	USA	Denmark
scas@itu.dk	thorsten.berger@chalmers.se	walkiner@oregonstate.edu	wasowski@itu.dk

```
+ #if m
+ /* inserted code */
+ #endif
```

decompose

insert code into a  
variant implementing *m*  
(then merge)

*m*

type of edit to software product line

type of edit to variants



# Results

RQ1 – Can we reproduce all considered kinds of edits?

RQ2 – How many feature contexts are necessary?

RQ3 – How complex are the feature contexts?

# Results

RQ1 – Can we reproduce all considered kinds of edits?

Yes

RQ2 – How many feature contexts are necessary?

RQ3 – How complex are the feature contexts?

# Results

RQ1 – Can we reproduce all considered kinds of edits?

Yes

RQ2 – How many feature contexts are necessary?



```
class Stack {  
    void push(int x) {  
        log("Push");  
        lock();  
        storage[head++] = x;  
        unlock();  
    }  
}
```

RQ3 – How complex are the feature contexts?

# Results

RQ1 – Can we reproduce all considered kinds of edits?

Yes

RQ2 – How many feature contexts are necessary?

less or as many as when manually specifying mappings

*m*

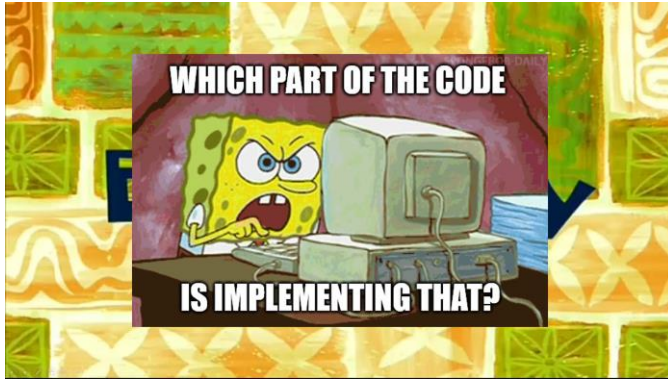


```
class Stack {  
    void push(int x) {  
        log("Push");  
        lock();  
        storage[head++] = x;  
        unlock();  
    }  
}
```

RQ3 – How complex are the feature contexts?

equal to target feature mapping

# Feature Trace Recording



Hey Alice, can I merge your changes?

Sure! 😊

But I have another variant!

That's fine, I recorded all feature traces!



Debug  
Functional  
⋮



```
void pop() {
  if (!empty()) {
    storage[head--] = null;
  }
}
```



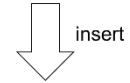
done with single

feature  
context  
=  
Functional

```
Stack<T> pop() {
  Stack<T> c = clone();
  if (!empty()) {
    c.storage[c.head--] = null;
  }
  return c;
}
```



```
void pop() {
  if (!empty()) {
  }
}
```



```
void pop() {
  Stack<T> c = clone();
  if (!empty()) {
    c.storage[c.head--] = null;
  }
  return c;
}
```

Can we reproduce edits to SPLs as edits to variants?



```
+ #if m
+ /* inserted code */
+ #endif
```

edit to SPL



insert code into a  
variant implementing *m*  
(then merge)

edit to variants

*m*