

# Feature Trace Recording



Paul Maximilian  
Bittner<sup>1</sup>



Alexander  
Schultheiß<sup>2</sup>



Thomas  
Thüm<sup>1</sup>



Timo  
Kehrner<sup>2</sup>



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



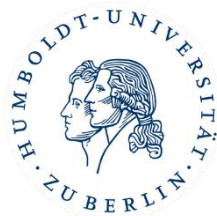
Lukas  
Linsbauer<sup>4</sup>

1



ulm university universität  
uulm

2



3



Oregon State  
University

4







**One  
Eternity  
Later**



**WHICH PART OF THE CODE**

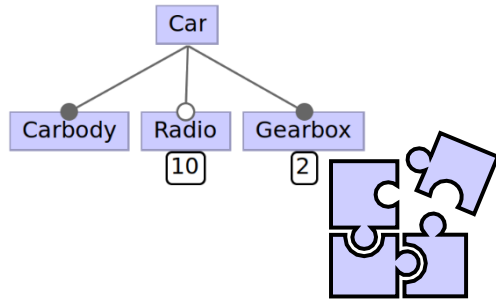


**IS IMPLEMENTING THAT?**

# Feature Traceability Problem

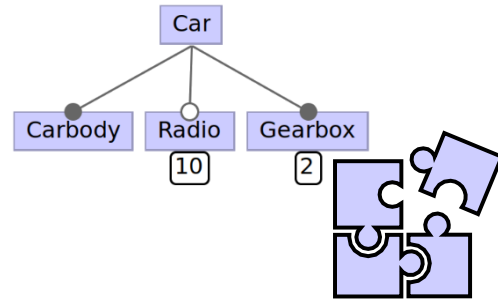
Feature Traceability is the knowledge  
where each feature is implemented.

Traceability is given in  
*software product lines ...*



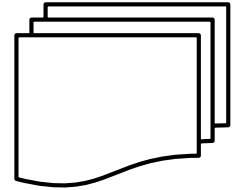
complete  
feature traces

Traceability is given in  
*software product lines ...*



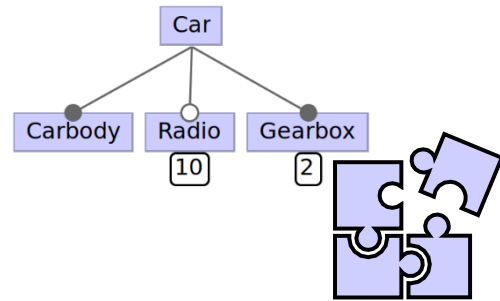
complete  
feature traces

... but in practice variability  
is often implemented with  
*clone-and-own*.



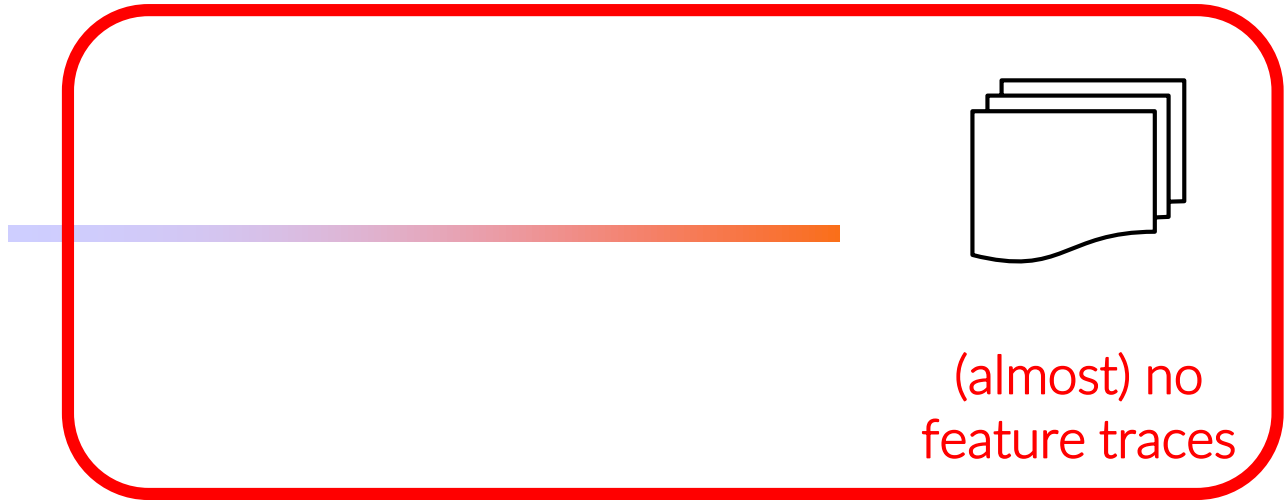
(almost) no  
feature traces

Traceability is given in  
*software product lines ...*



complete  
feature traces

... but in practice variability  
is often implemented with  
*clone-and-own*.



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



# Feature traces can be documented ...

**Retroactively:** after development (Variability Mining [Kästner et al.]

Requires to halt development

Not always possible because knowledge is lost

**Proactively:** during development (Embedded Annotations [Ji et al.]

No automation yet

→ Feature Trace Recording

# Example of Feature Trace Recording

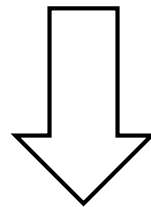
This crashes  
when the stack  
is empty!



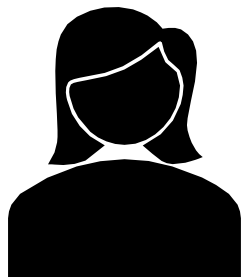
```
class Stack {  
  
    /* ... */  
  
    void pop() {  
        storage[head--] = null;  
    }  
  
}
```

# Example of Feature Trace Recording

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

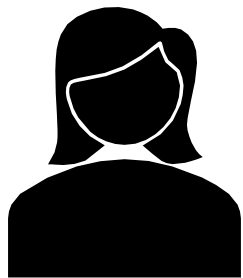


```
void pop() {  
    if (!empty()) {}  
    storage[head--] = null;  
}
```

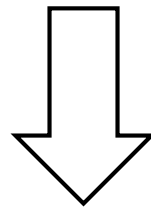


# Example of Feature Trace Recording

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

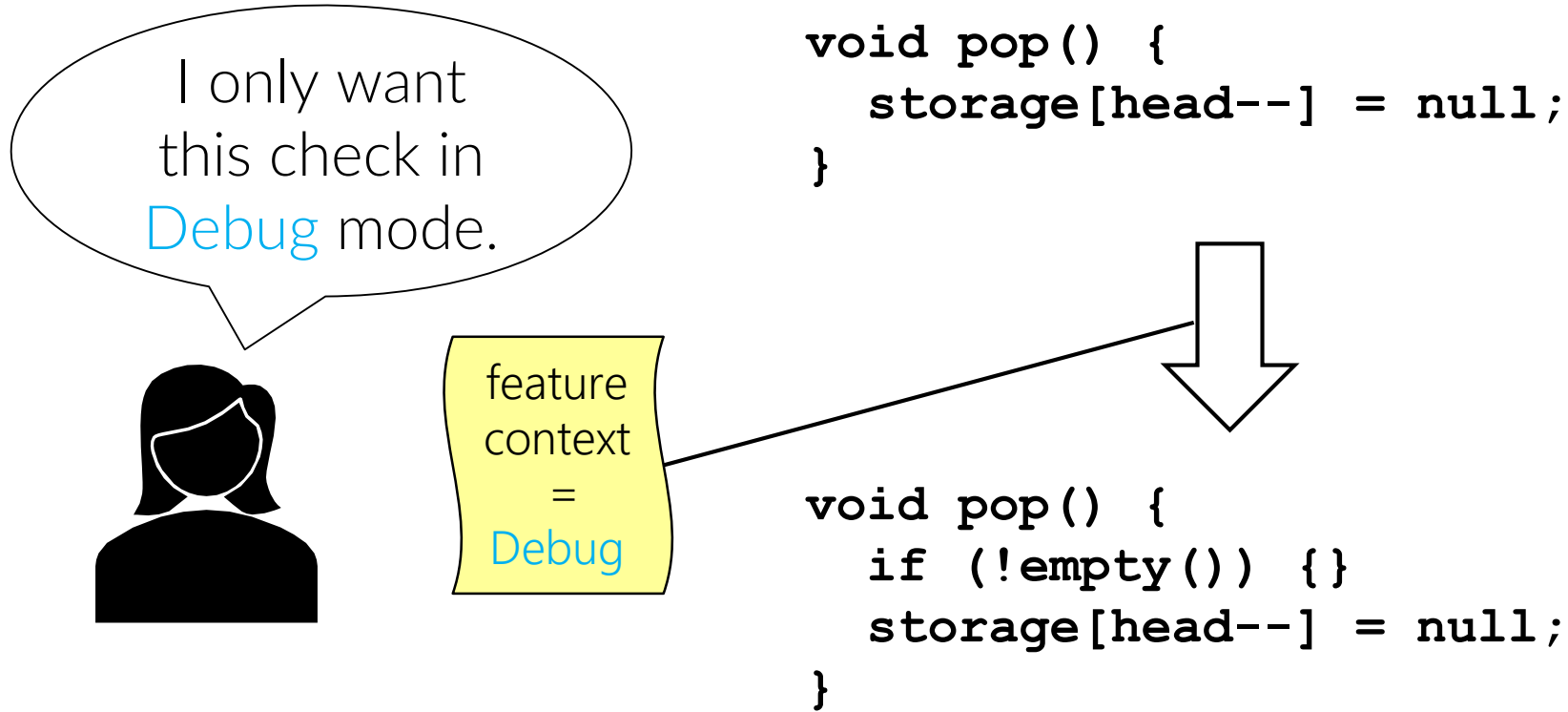


```
void pop() {  
    storage[head--] = null;  
}
```



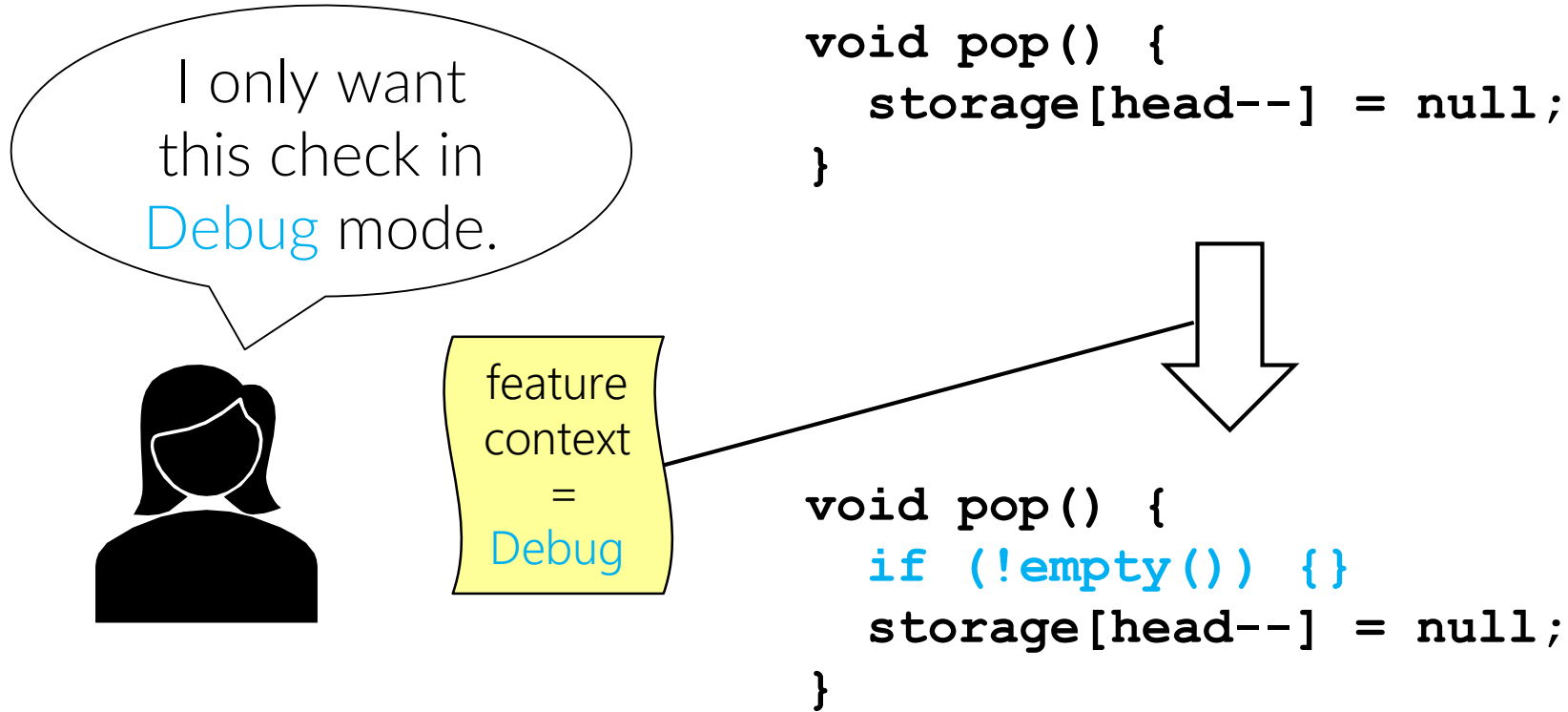
```
void pop() {  
    if (!empty()) {}  
    storage[head--] = null;  
}
```

# Example of Feature Trace Recording

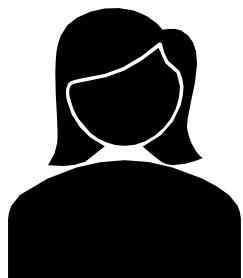




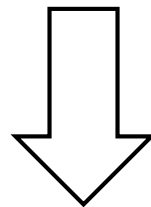
# Example of Feature Trace Recording



# Example of Feature Trace Recording



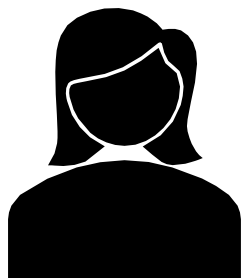
```
void pop() {  
    if (!empty()) {}  
    storage[head--] = null;  
}
```



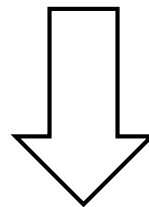
```
void pop() {  
    if (!empty()) {  
        storage[head--] = null;  
    }  
}
```

# Example of Feature Trace Recording

I don't know  
the feature.

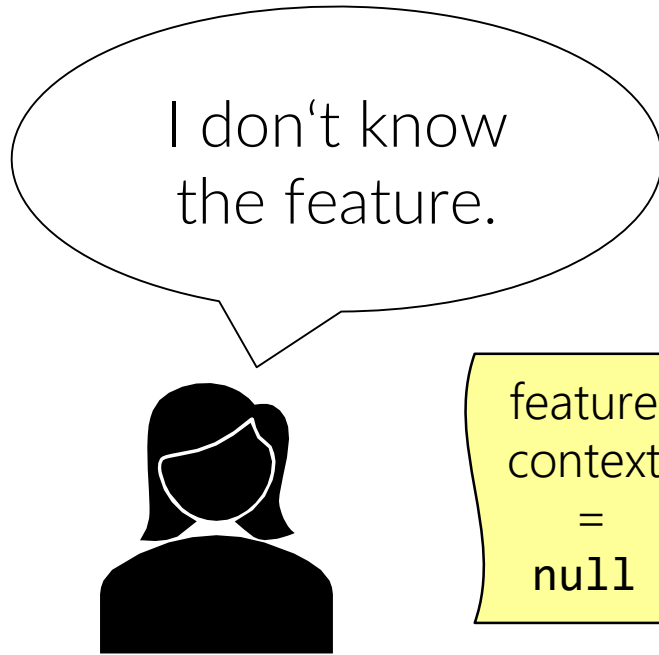


```
void pop() {  
    if (!empty()) {}  
    storage[head--] = null;  
}
```



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

# Example of Feature Trace Recording



feature  
context  
=  
null

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

A large white arrow points from the top code block to the bottom code block. A line also connects the yellow 'feature context = null' box to the start of the bottom code block.

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

# Example of Feature Trace Recording – The Next Week

## Stacks should be immutable #1

Edit

New issue

Open

pmbittner opened this issue now · 0 comments



pmbittner commented now



We like functional programming now!

Assignees



Alice



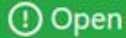


# Example of Feature Trace Recording – The Next Week

## Stacks should be immutable #1

Edit

New issue



pmbittner opened this issue now · 0 comments



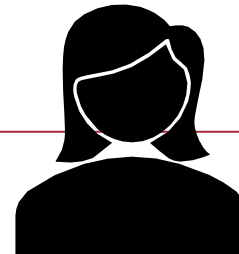
pmbittner commented now

We like functional programming now!

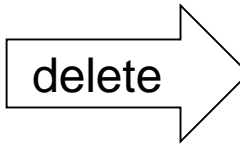
Ok, I am working  
on

feature  
context  
=

Functional



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



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

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

delete

```
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()) {
    }
}
```

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()) {
    }
}
```

done with single

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;
}
```

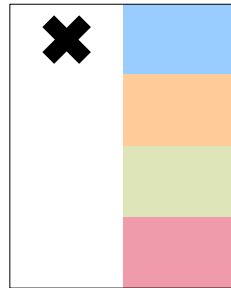
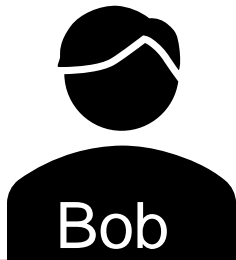


Hey Alice, can I merge your changes?

Sure! 😊

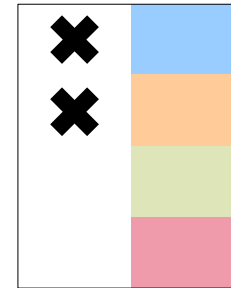
But I have another variant!

That's fine, I recorded all  
feature traces!



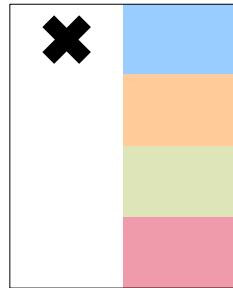
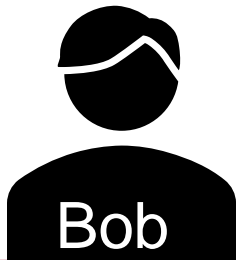
Debug  
Functional

⋮



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

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



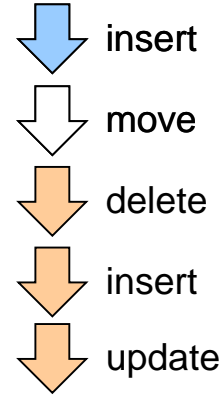
Debug  
Functional

⋮

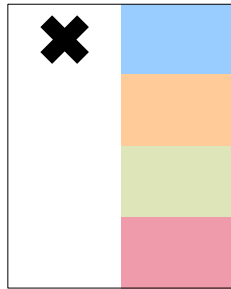
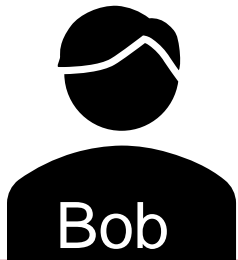


```
void pop() { /*...*/ }
```

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



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



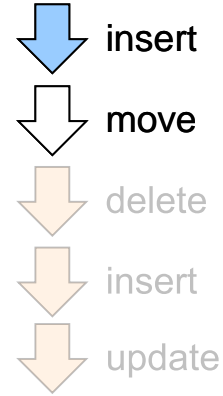
Debug  
Functional

⋮

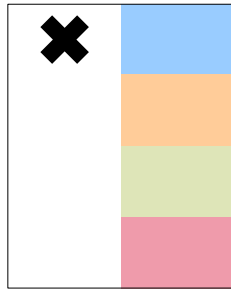
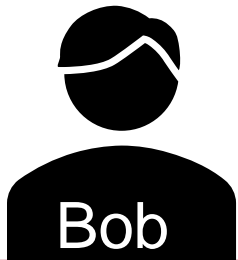


```
void pop() { /*...*/ }
```

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



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

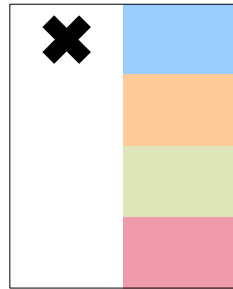
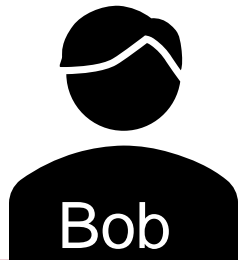
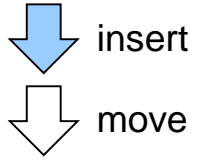


Debug  
Functional

⋮



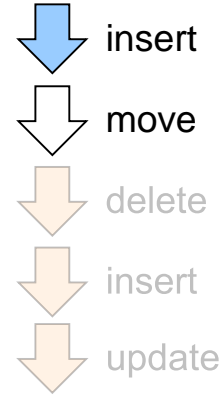
```
void pop() { /*...*/ }
```



Debug  
Functional

⋮

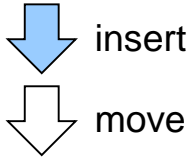
```
void pop() { /*...*/ }
```



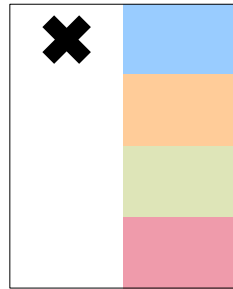
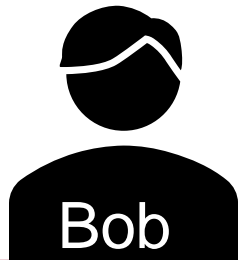
```
Stack<T> pop() { /*...*/ }
```



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



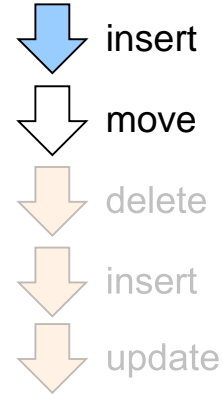
```
void pop() {  
    if (!empty()) {  
        storage[head--] = null;  
    }  
}
```



Debug  
Functional

⋮

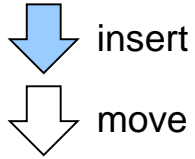
```
void pop() { /*...*/ }
```



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

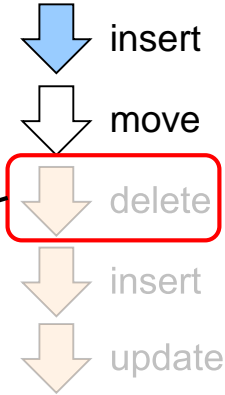


```
void pop() { /*...*/ }
```

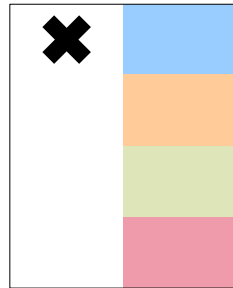
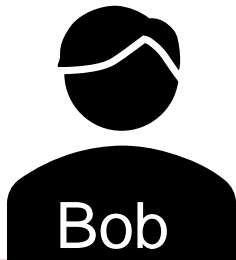


```
void pop() {  
  if (!empty()) {  
    storage[head--] = null;  
  }  
}
```

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



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

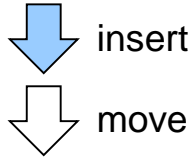


Debug  
Functional

⋮



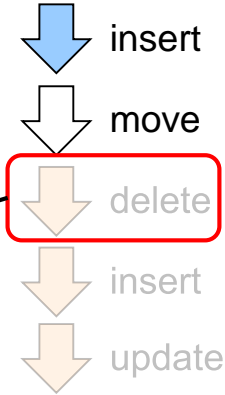
```
void pop() { /*...*/ }
```



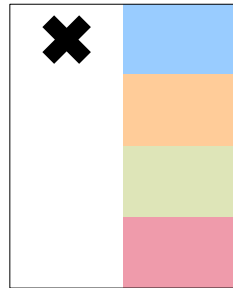
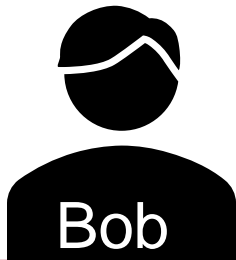
```
void pop() {  
  if (!empty()) {  
    storage[head--] = null;  
  }  
}
```

¬Functional

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



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

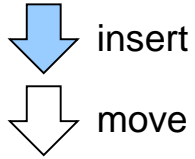


Debug  
Functional

⋮



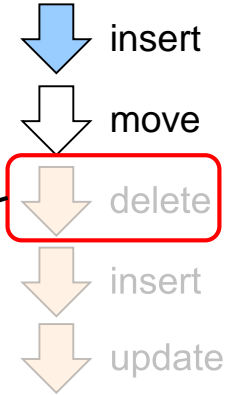
```
void pop() { /*...*/ }
```



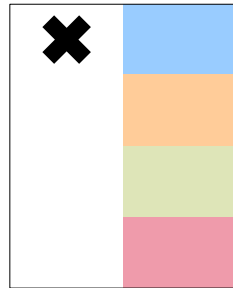
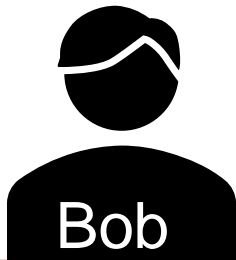
```
void pop() {  
  if (!empty()) {  
    storage[head--] = null;  
  }  
}
```

¬Functional

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

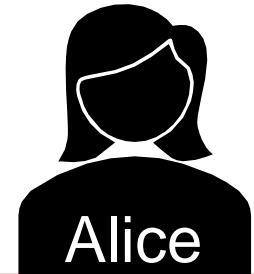


```
Stack<T> pop() { /*...*/ }
```

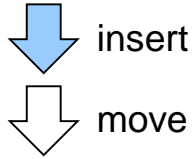


Debug  
Functional

⋮



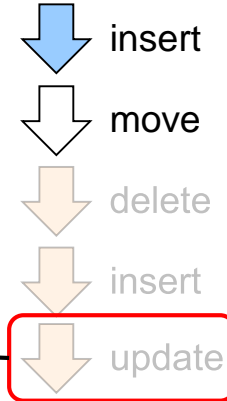
```
void pop() { /*...*/ }
```



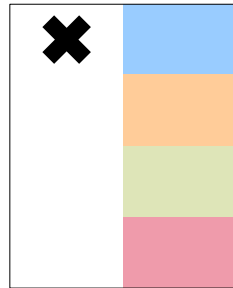
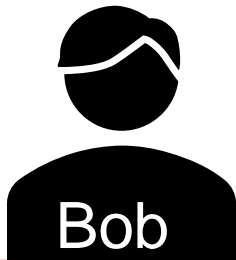
```
void pop() {  
  if (!empty()) {  
    storage[head--] = null;  
  }  
}
```

→Functional

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

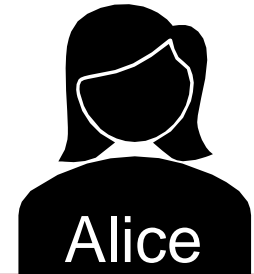


```
Stack<T> pop() { /*...*/ }
```

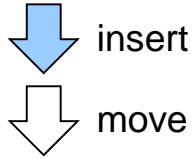


Debug  
Functional

⋮



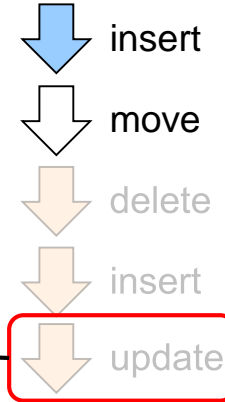
```
void pop() { /*...*/ }
```



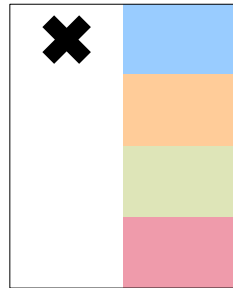
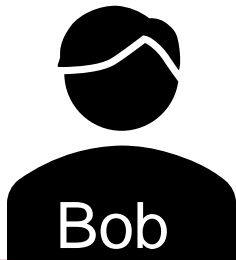
```
void pop() {  
  if (!empty()) {  
    storage[head--] = null;  
  }  
}
```

→Functional

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



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

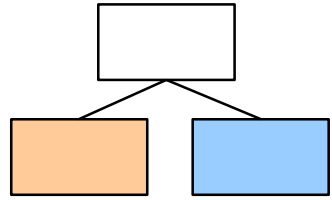


Debug  
Functional

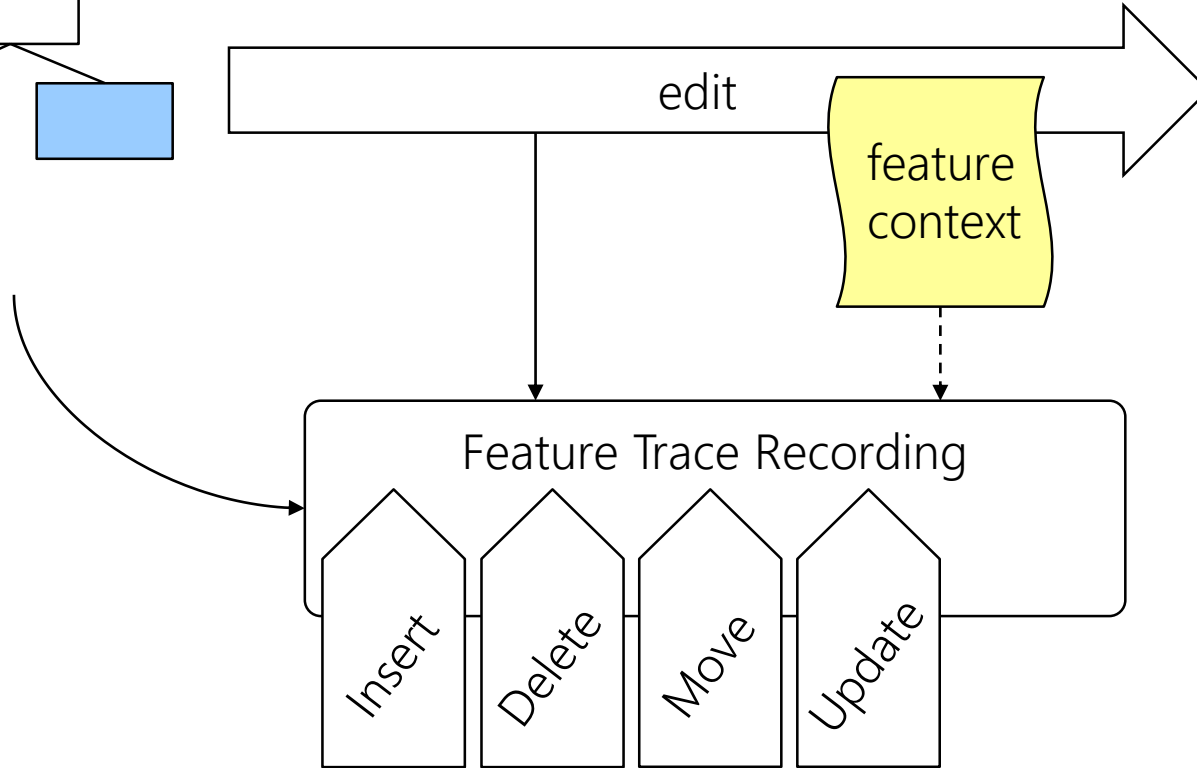
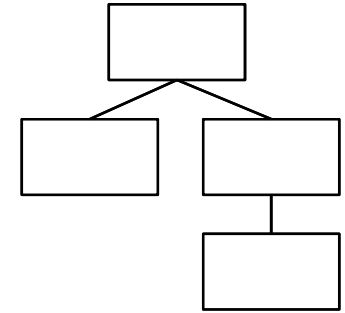
⋮



old code version

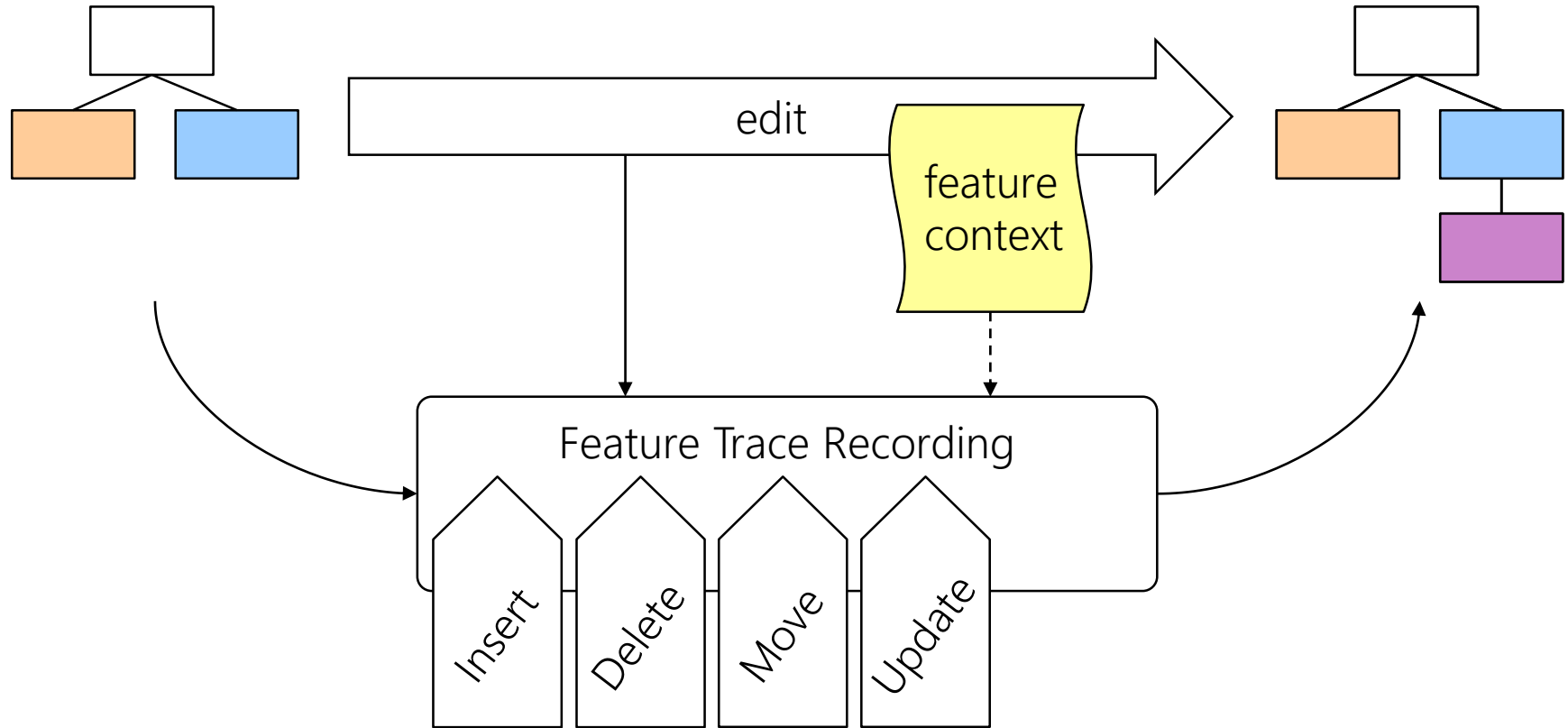


new code version



old code version

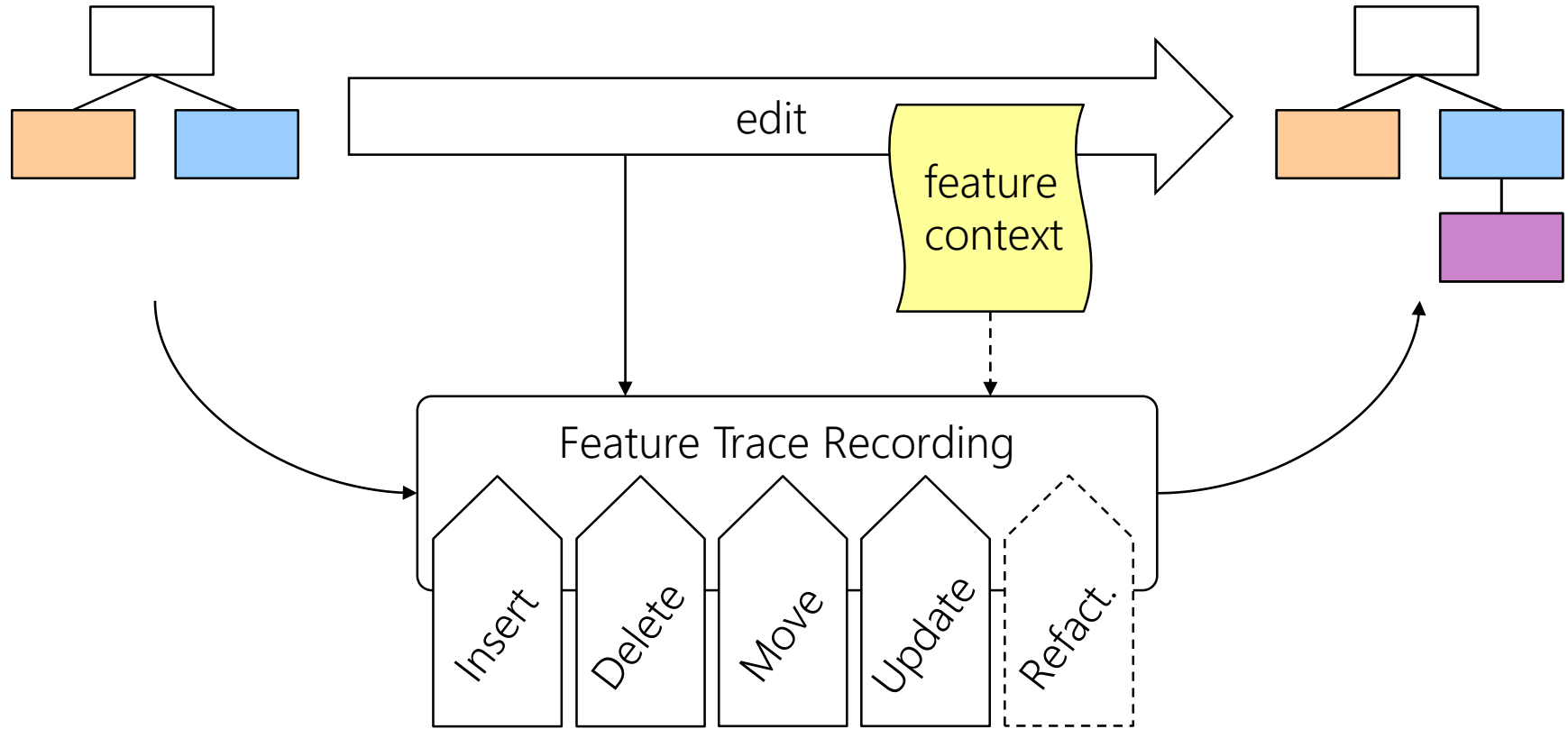
new code version



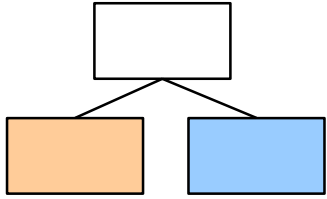


old code version

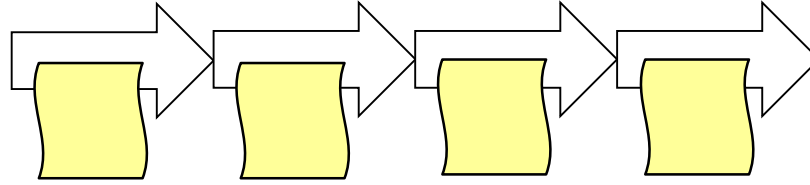
new code version



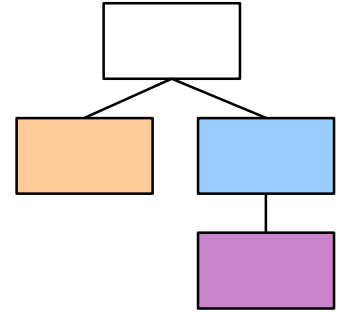
To evaluate feature trace recording we need



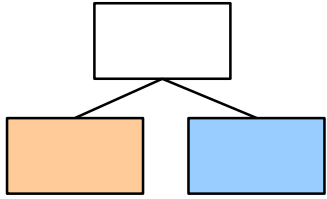
edits (e.g., derived from commit history)



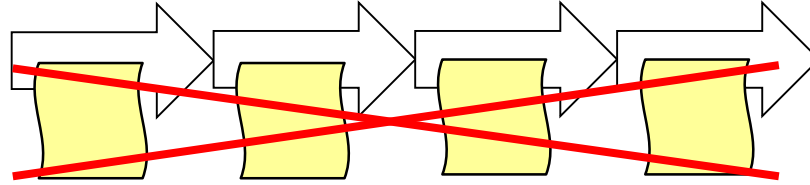
feature contexts



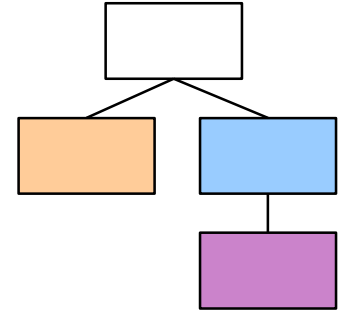
To evaluate feature trace recording we need



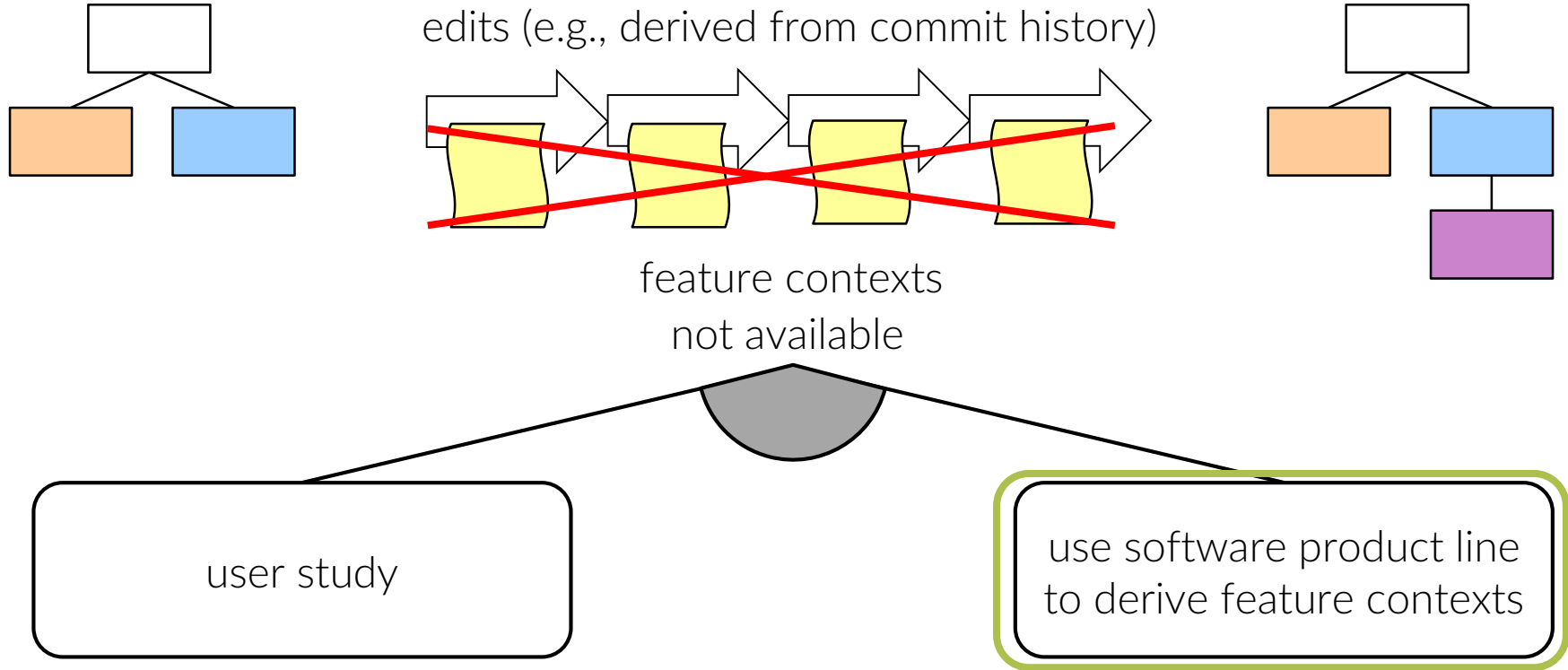
edits (e.g., derived from commit history)



feature contexts  
not available



To evaluate feature trace recording we need



# Can we reproduce edits to SPLs as edits to variants?

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

Ștefan Stănciulescu	Thorsten Berger	Eric Walkingshaw	Andrzej Wąsowski
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

## Empirical Evaluation of Feature Trace Recording on the Edit History of Marlin

Sören Viegner

# Can we reproduce edits to SPLs as edits to variants?

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

Stefan Stănculescu    Thorsten Berger    Eric Walkingshaw    Andrzej Wąsowski  
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

Empirical Evaluation of Feature Trace  
Recording on the Edit History of Marlin

Sören Viegner

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

edit to SPL

decompose

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

edit to variants

# Can we reproduce edits to SPLs as edits to variants?

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

Stefan Stănculescu    Thorsten Berger    Eric Walkingshaw    Andrzej Wąsowski  
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

Empirical Evaluation of Feature Trace  
Recording on the Edit History of Marlin

Sören Viegner

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

edit to SPL

decompose

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

*m*

edit to variants

# Results

RQ1 – Can we reproduce all considered kinds of edits?

Yes

RQ2 – How many feature contexts are necessary?

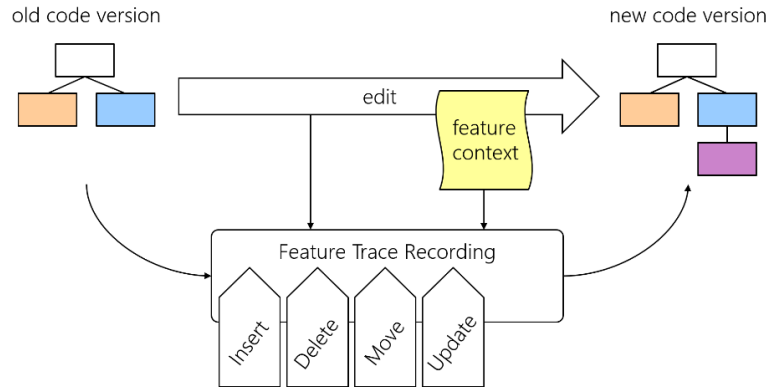
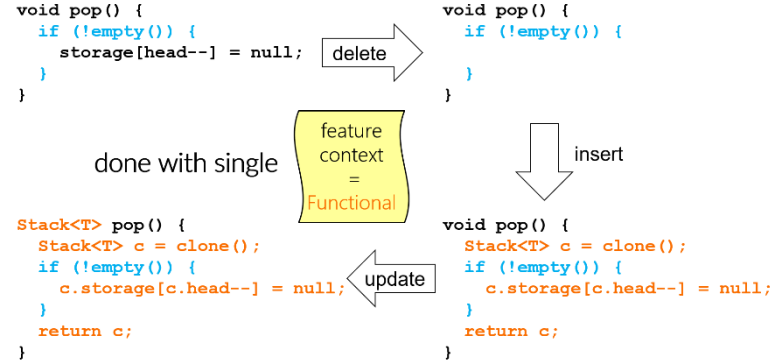
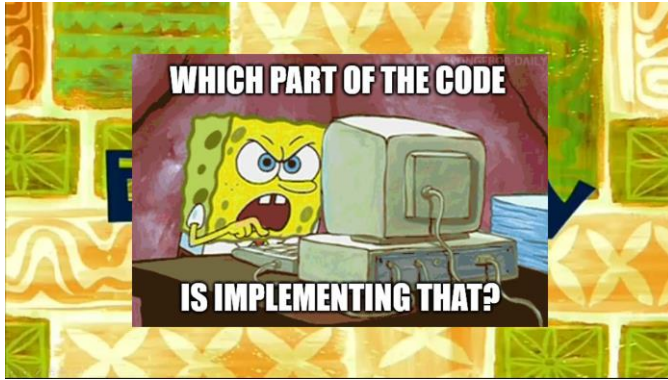
less or as many as when directly specifying mappings (worst case)

RQ3 – How complex are the feature contexts?

equal to target feature mapping (worst case)



# Feature Trace Recording



Can we reproduce edits to SPLs as edits to variants?

