Feature Trace Recording

Paul Maximilian Bittner[1]

Alexander Schultheiß[2]

Thomas Thüm[1]

Timo Kehrer[3]

Jeffrey M. Young[4]

Lukas Linsbauer[5]

1 universität ulm

2 Humboldt-Universität zu Berlin

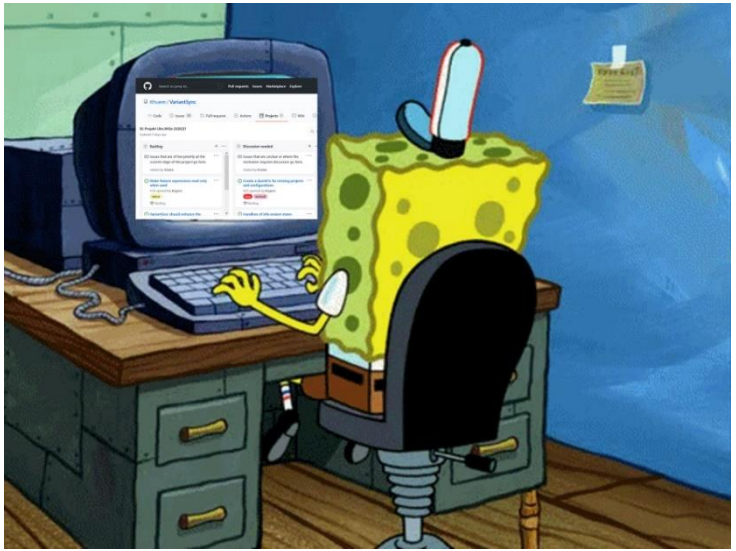
3 Universität Bern

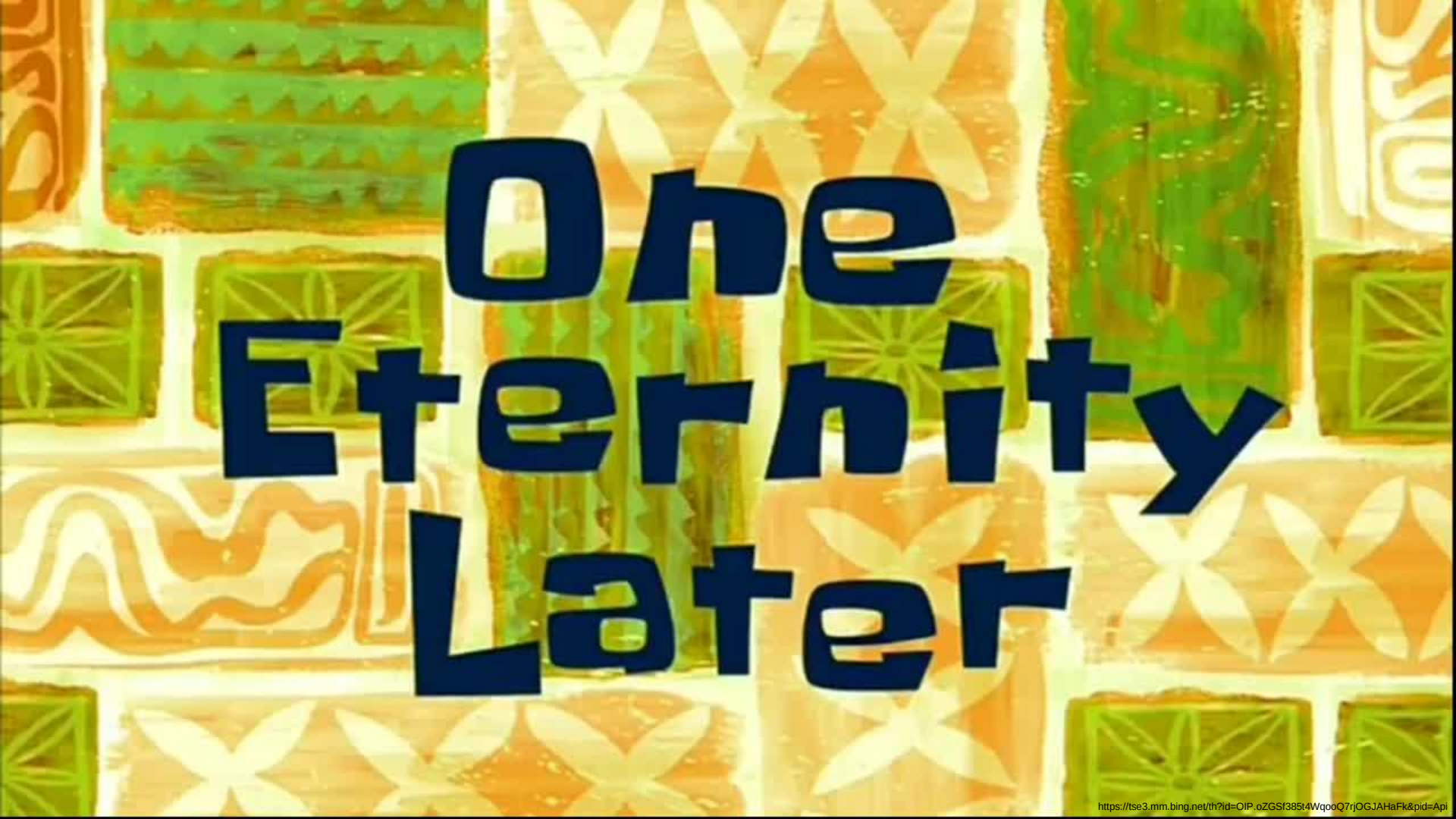
4 INPUT | OUTPUT

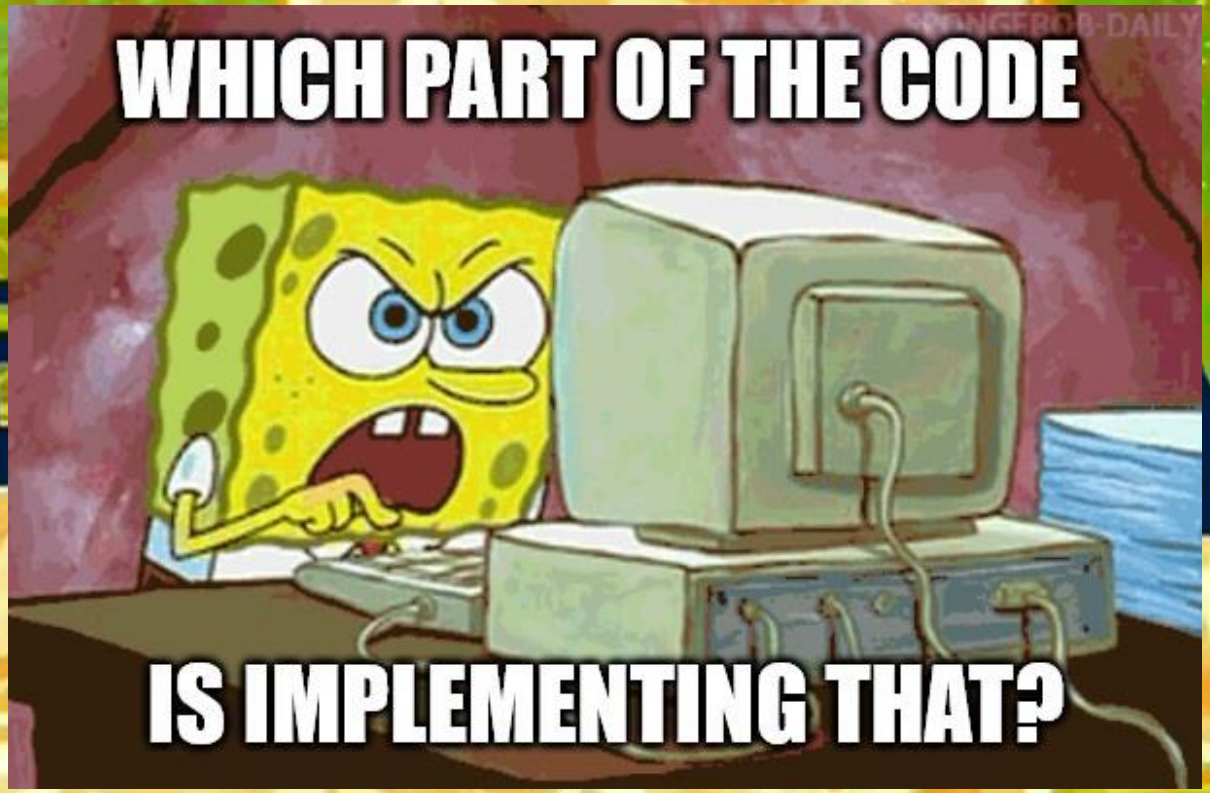
5 Technische Universität Braunschweig


YOUR NEW EMPLOYEE


YOUR NEW EMPLOYEE





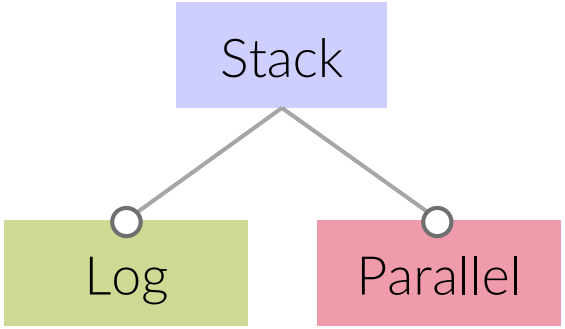


https://tse3.mm.bing.net/th?id=OIP.oZGSf385t4WqooQ7rjOGJAHaFk&pid=Api

Feature Traceability Problem

Feature Traceability is the knowledge where each feature is implemented.

Software Product Lines – Problem Solved?

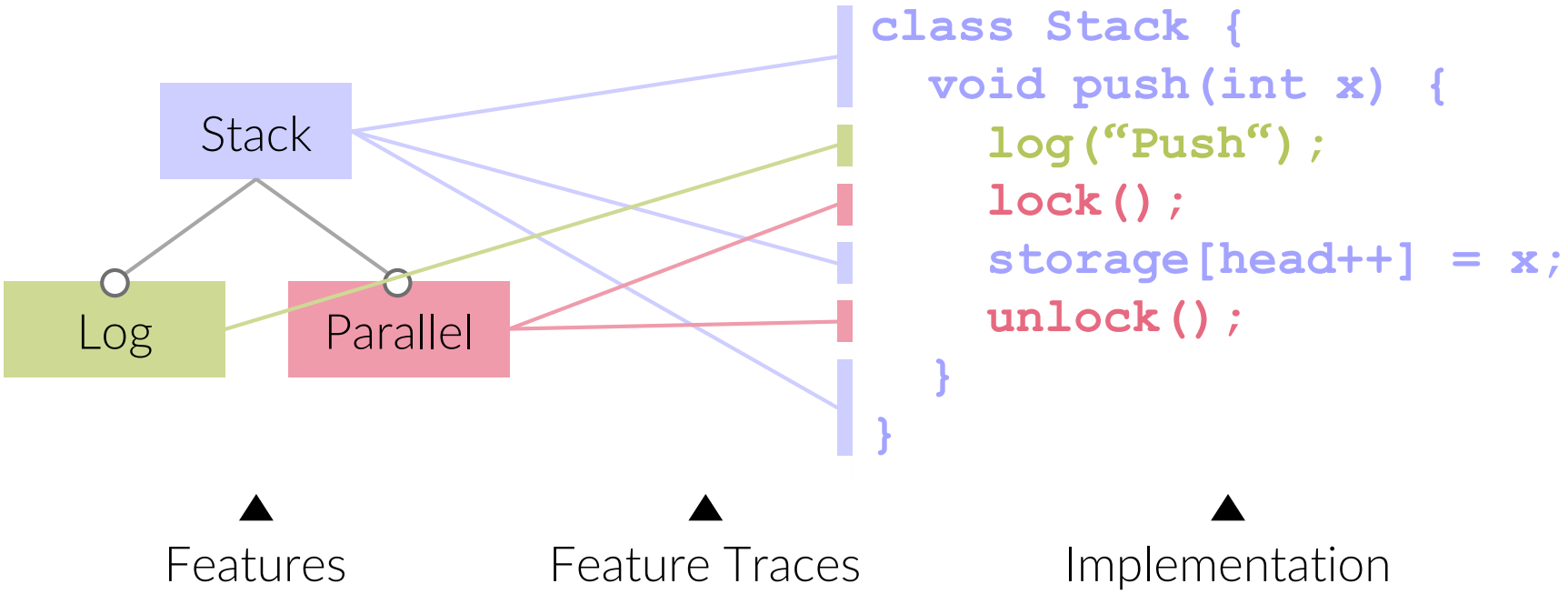


Features

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

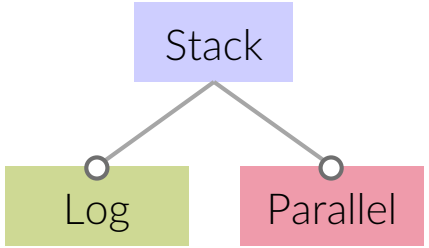
Implementation

Software Product Lines – Problem Solved?



Not yet: *Software product lines*

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

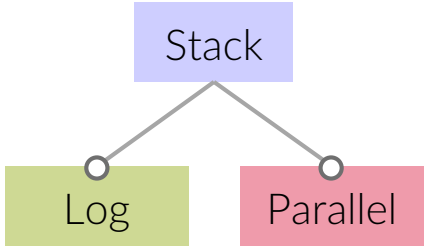


complete feature traces

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*.



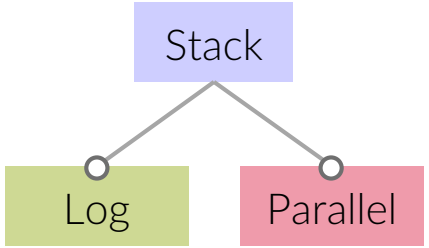
complete feature traces

(almost) no feature traces

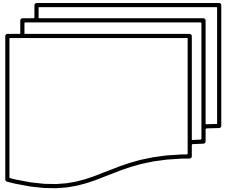
Not yet: *Software product lines*

- 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



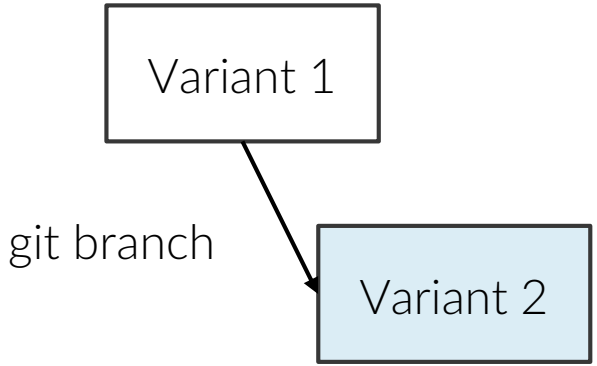
(almost) no feature traces

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

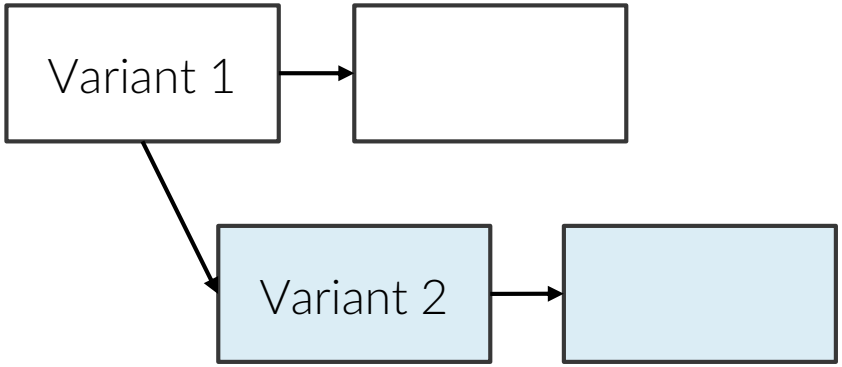
Clone-and-Own as Prominent Variability Approach

Software

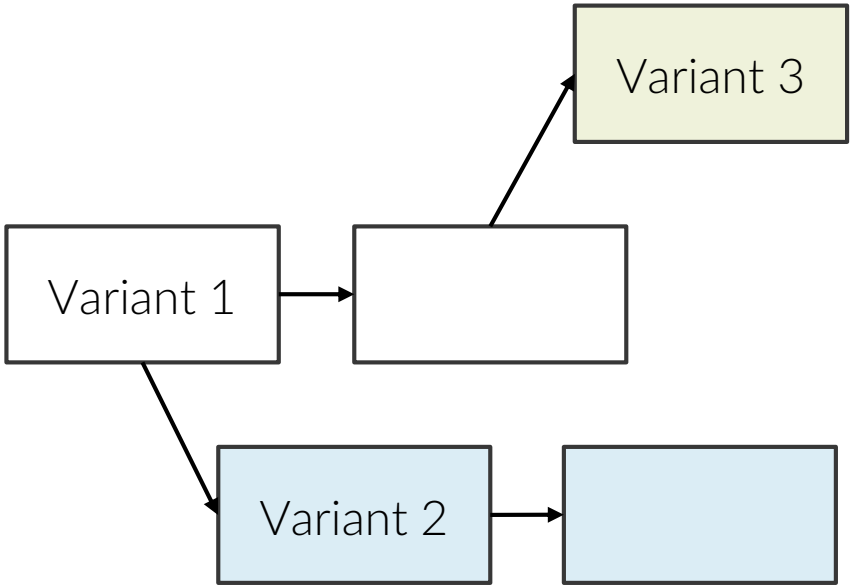
Clone-and-Own as Prominent Variability Approach



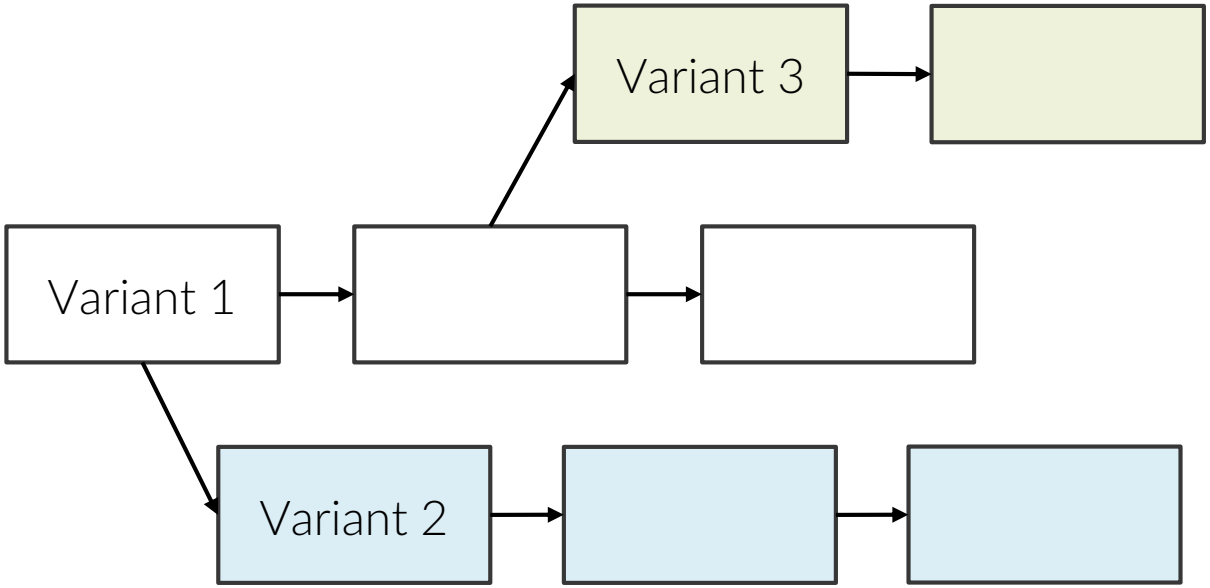
Clone-and-Own as Prominent Variability Approach



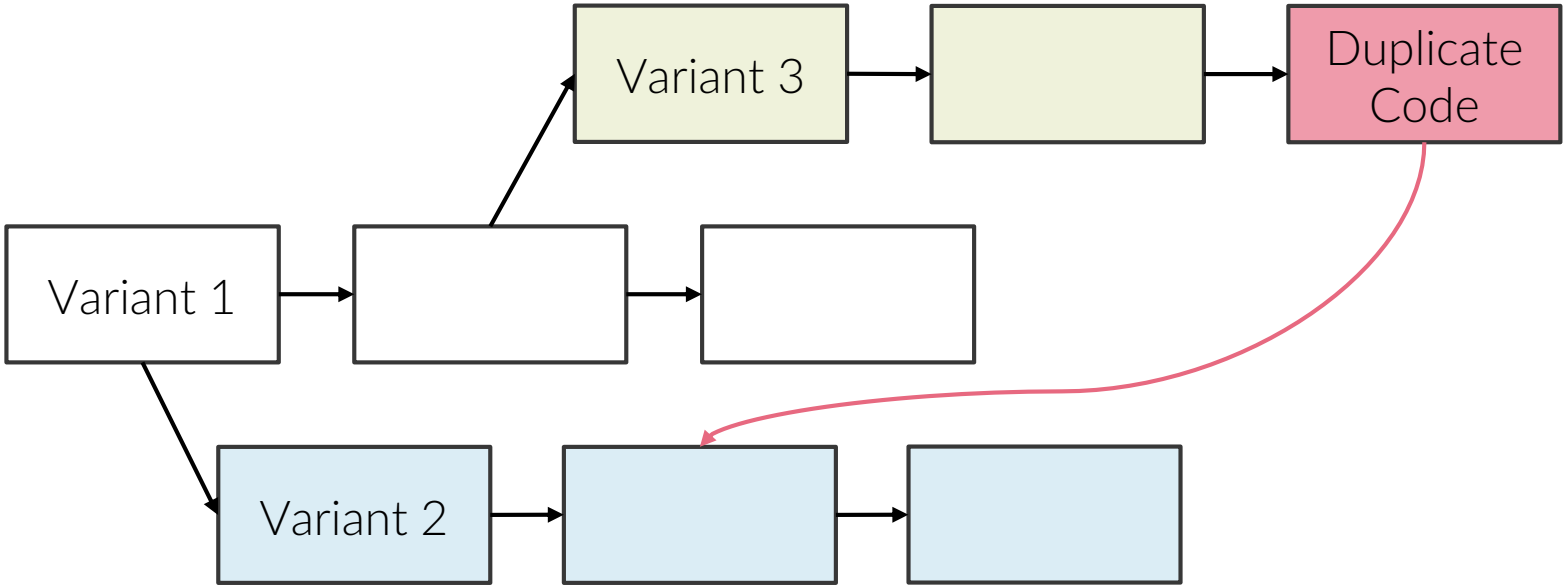
Clone-and-Own as Prominent Variability Approach



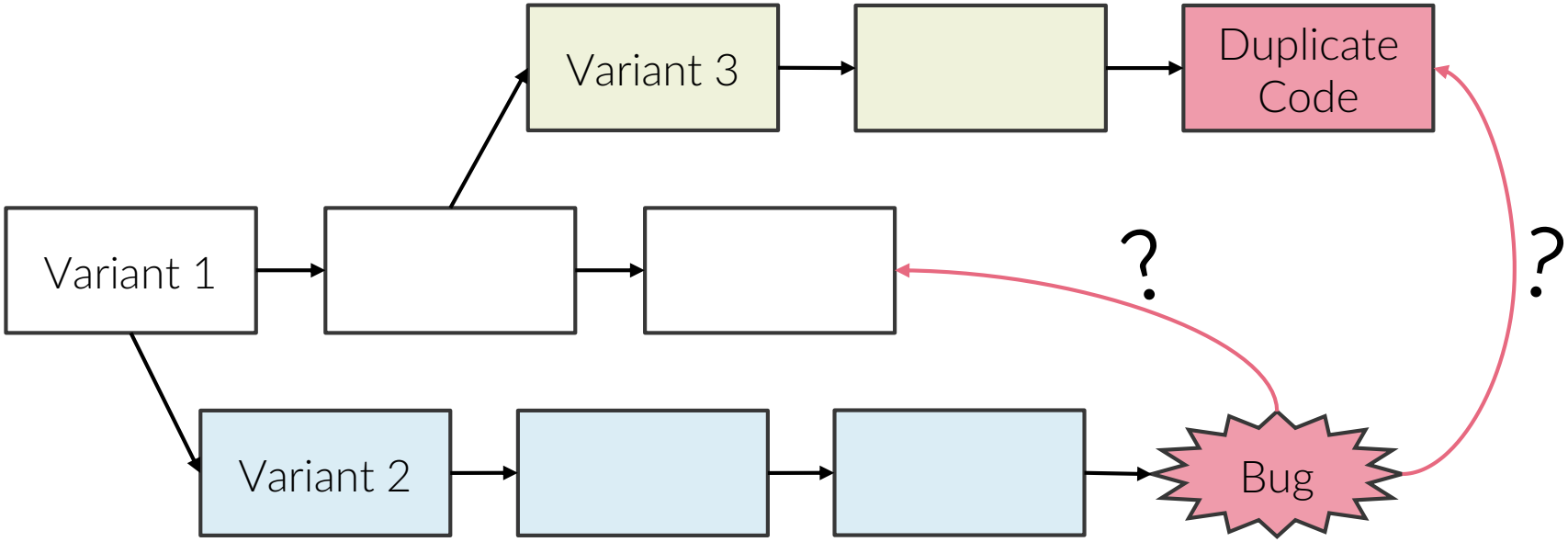
Clone-and-Own as Prominent Variability Approach



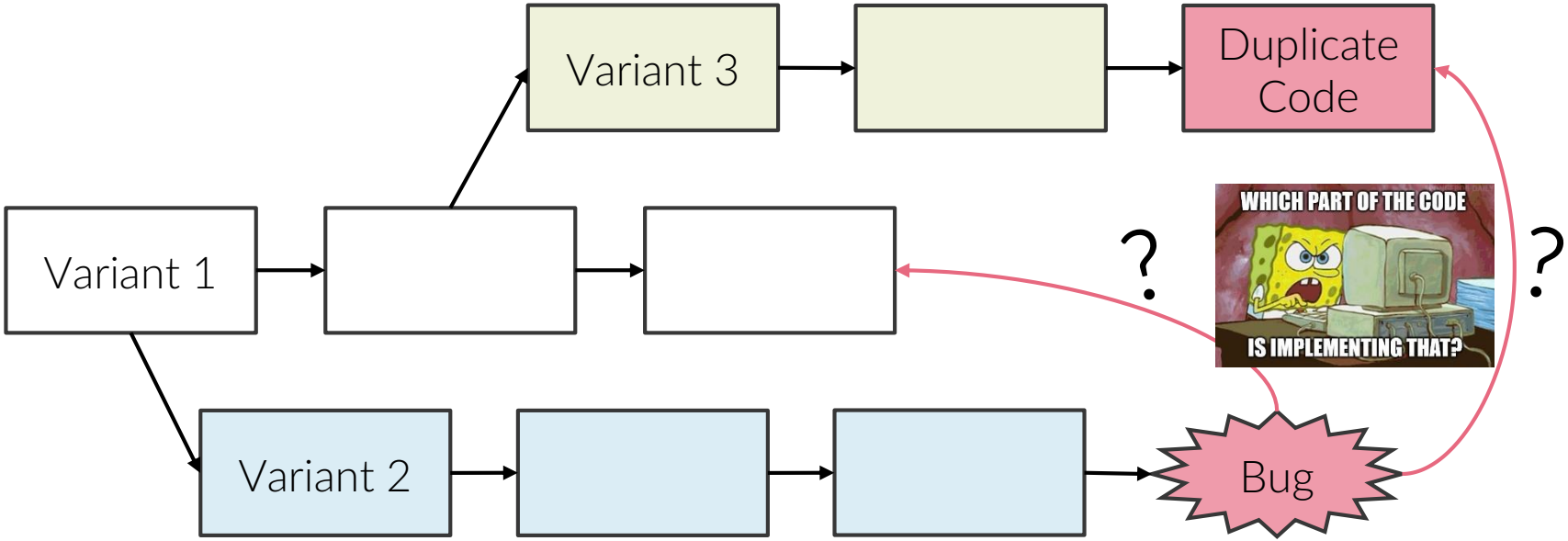
Clone-and-Own as Prominent Variability Approach



Clone-and-Own as Prominent Variability Approach



Clone-and-Own as Prominent Variability Approach



Feature traces can be documented …

Retroactively: after development (Feature Location, Variability Mining)

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

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.])

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

Feature traces can be documented …

Retroactively: after development (Feature Location, Variability Mining)

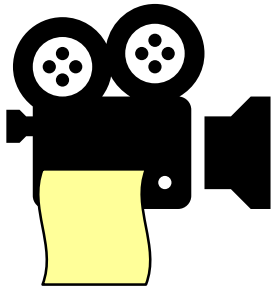
separate step in workflow

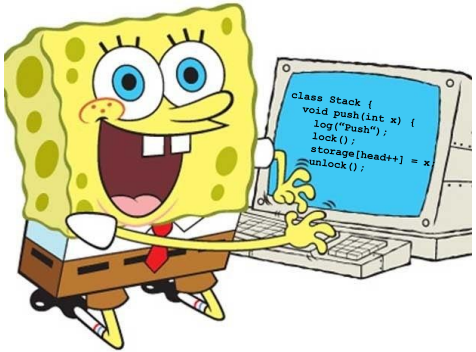
not always possible because knowledge is lost

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

manual

→ our contribution: semi-automation





https://images.techhive.com/images/article/2015/02/spongebob_computer-620x465-100566767-gallery.idge.jpg

Feature Trace Recording

Semi-Automation of Proactive Feature-Trace Specification

Software Engineering
Programming Languages

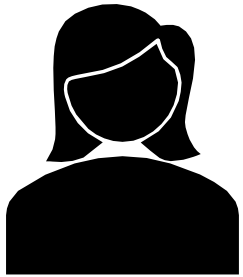
Example of Feature Trace Recording


```
class Stack {

  /* ... */

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

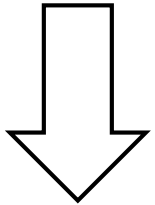
Example of Feature Trace Recording


```
class Stack {

  /* ... */

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

Example of Feature Trace Recording

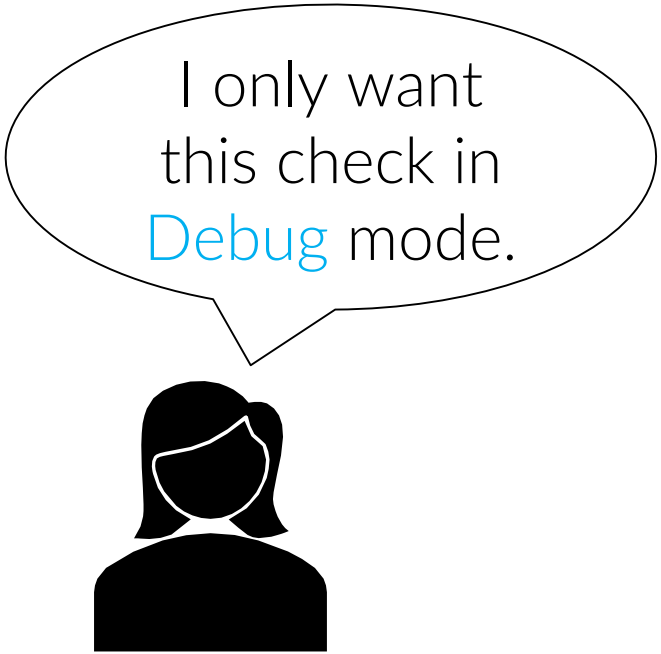


```
void pop() {
  storage[head--] = null;
}
```

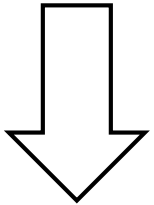


```
void pop() {
  if (!empty()) {}
  storage[head--] = null;
}
```

Example of Feature Trace Recording

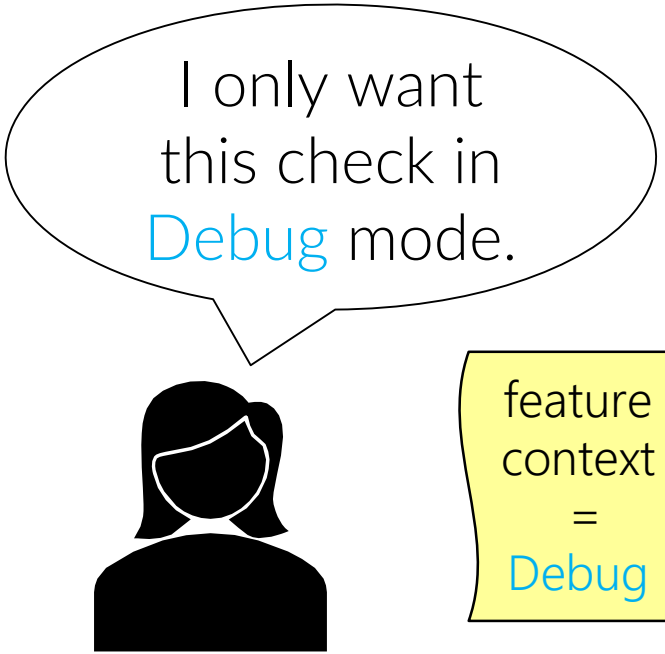



```
void pop() {
  storage[head--] = null;
}
```



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

Example of Feature Trace Recording

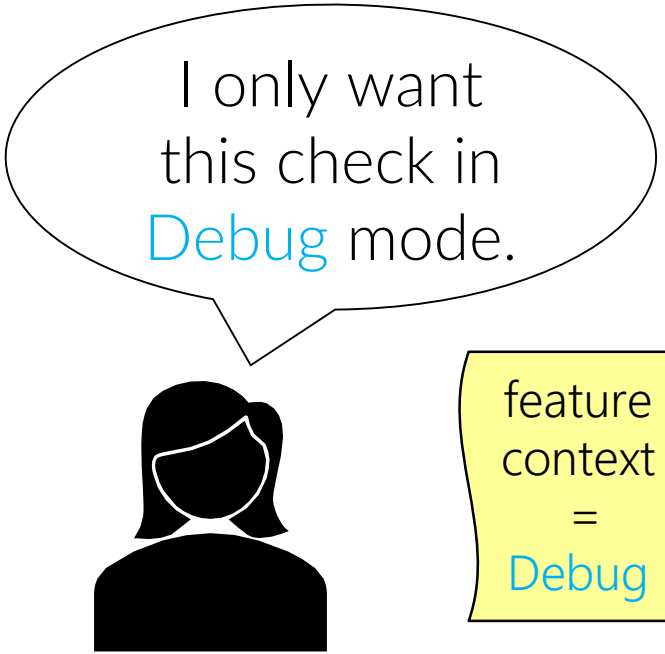



```
void pop() {
  storage[head--] = null;
}
```

feature context = Debug

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

Example of Feature Trace Recording

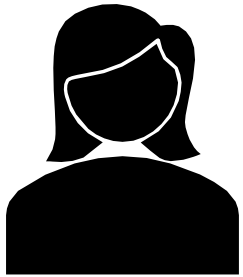



```
void pop() {
  storage[head--] = null;
}
```

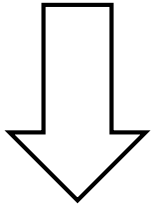
feature context = Debug

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

Example of Feature Trace Recording



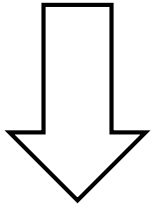
```
void pop() {
  if (!empty()) {}
  storage[head--] = null;
}
```



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

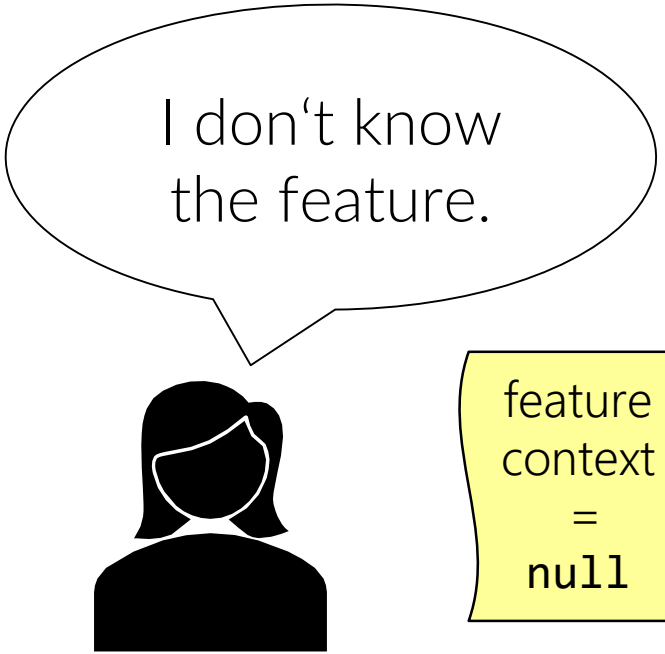
Example of Feature Trace Recording


```
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;
}
```

feature context = null

```
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

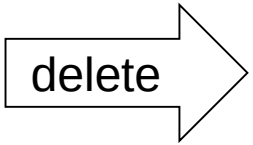
We like functional programming now!

Assignees

Alice

Example of Feature Trace Recording – The Next Week

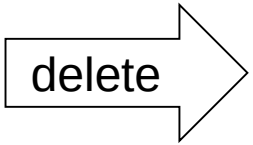
```
void pop() {
  if (!empty()) {
    storage[head--] = null;
  }
}
```



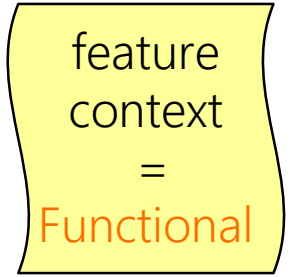

```
void pop() {
  if (!empty()) {

  }
}
```

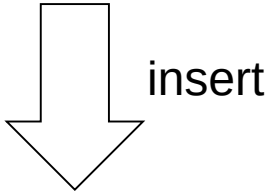
feature
context
=
Functional

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



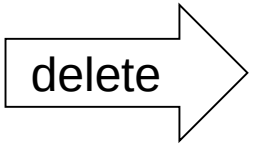

```
void pop() {
  if (!empty()) {

  }
}
```



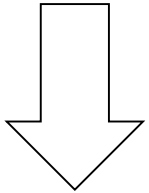



```
void pop() {
  Stack<T> c = clone();
  if (!empty()) {
    c.storage[c.head--] = null;
  }
  return c;
}
```

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




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

  }
}
```

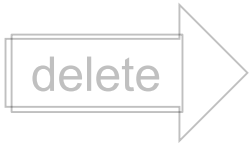


feature context = Functional

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


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

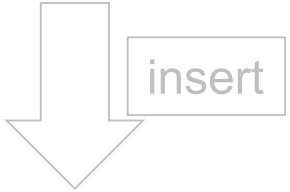
```
void pop() {
  if (!empty()) {
    storage[head--] = null;
  }
}
```




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

  }
}
```

done with single

feature context = Functional

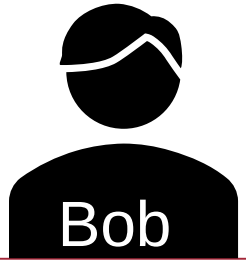



```
Stack<T> pop() {
  Stack<T> c = clone();
  if (!empty()) {
    c.storage[c.head--] = null;
  }
  return c;
}
```


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

Hey Alice, can I merge your changes?

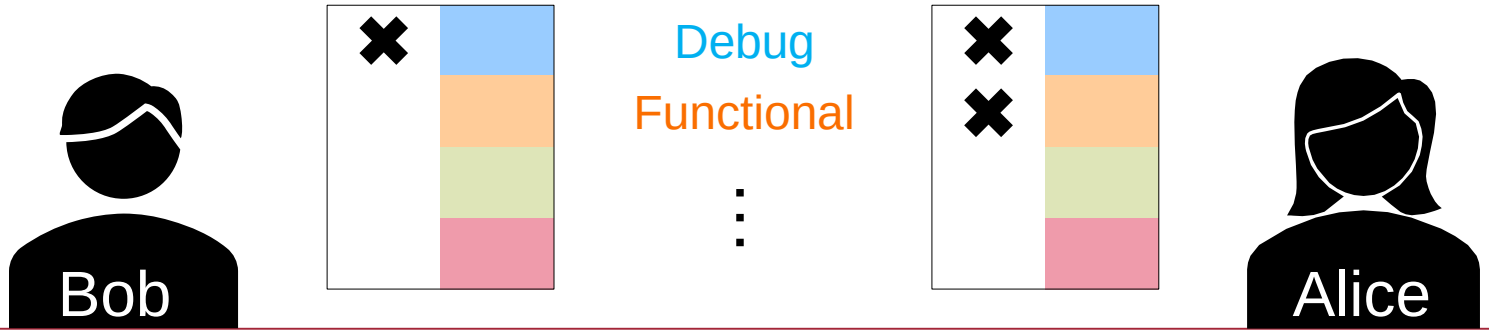
Sure! ☺



Hey Alice, can I merge your changes?

Sure! ☺

But I have another variant!

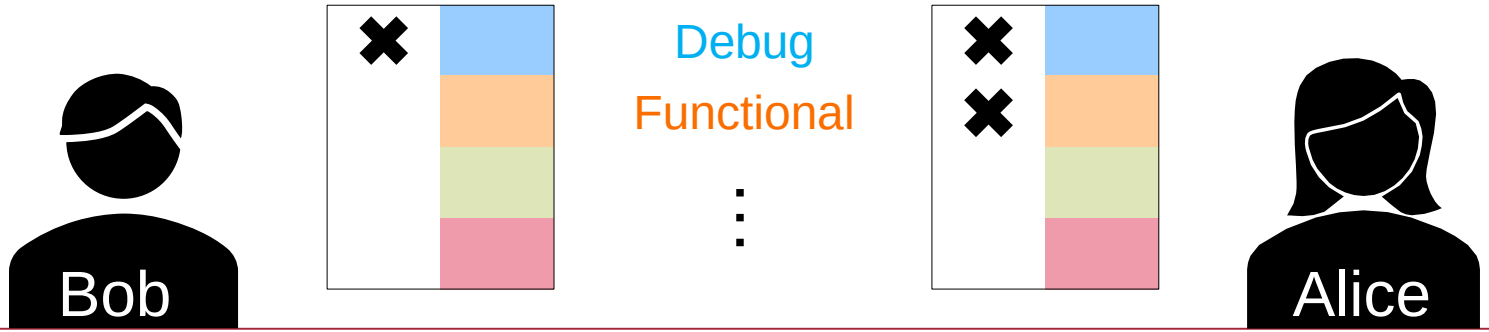


Hey Alice, can I merge your changes?

Sure! ☺

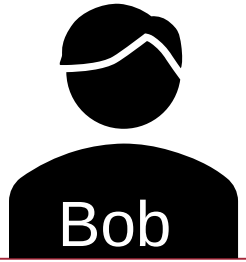
But I have another variant!

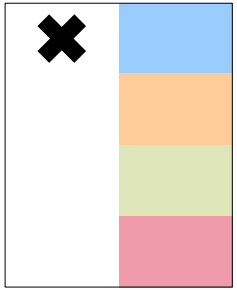
That‘s fine, I recorded all feature traces!



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

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





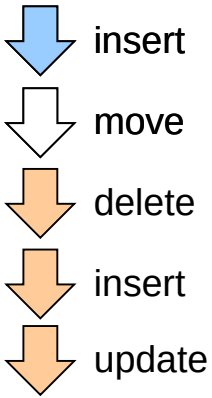
Debug

Functional

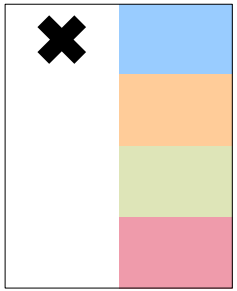
⋮

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

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



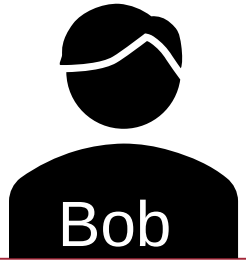

```
Stack<T> pop() {/*...*/}
```



Debug

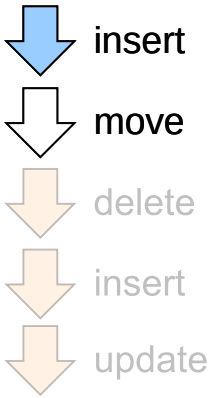
Functional

⋮

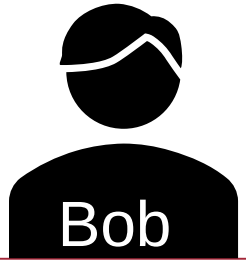


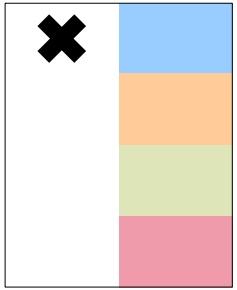
```
void pop() {/*...*/}
```

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




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



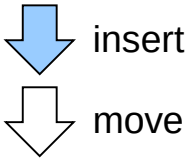


Debug

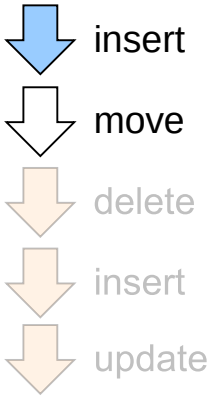
Functional

⋮

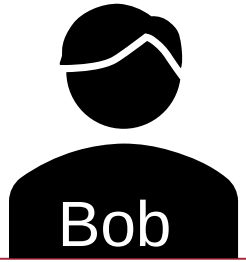
```
void pop() {/*...*/}
```

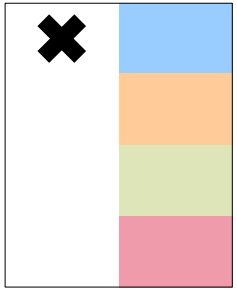



```
void pop() {/*...*/}
```




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



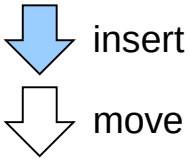


Debug

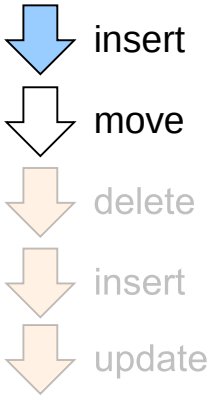
Functional

⋮

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

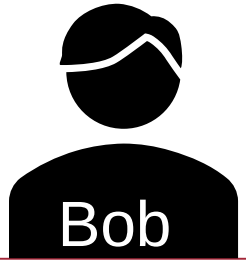


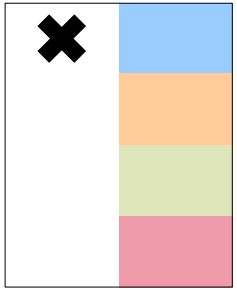

```
void pop() {/*...*/}
```




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

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



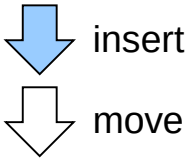


Debug

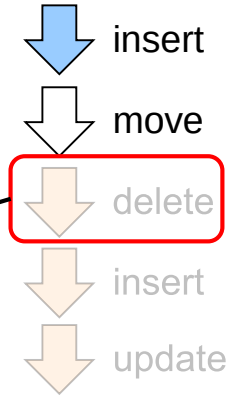
Functional

⋮

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

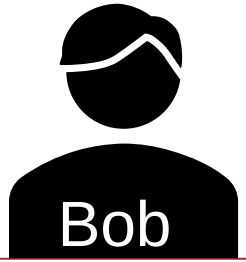


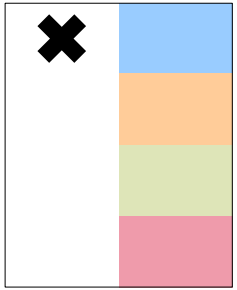

```
void pop() {/*...*/}
```




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

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



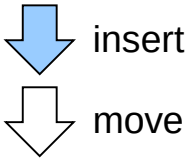


Debug

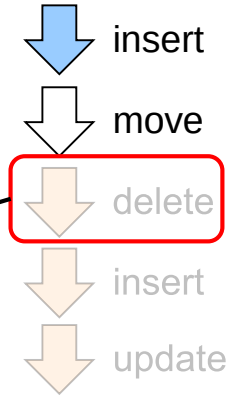
Functional

⋮

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



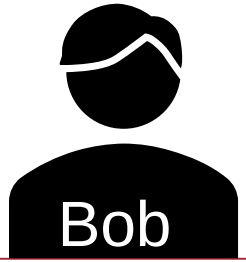

```
void pop() {/*...*/}
```

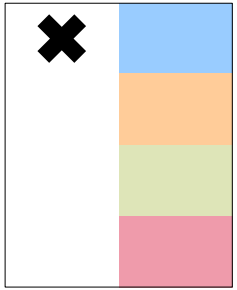



```
void pop() {
  if (!empty()) {
    storage[head--] = null;
  }
}
```

¬Functional

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



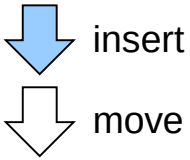


Debug

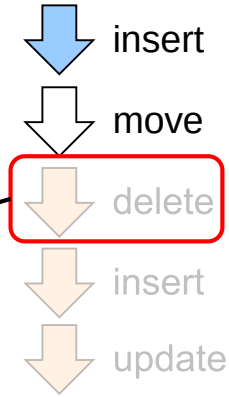
Functional

⋮

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



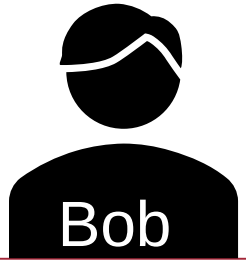

```
void pop() {/*...*/}
```

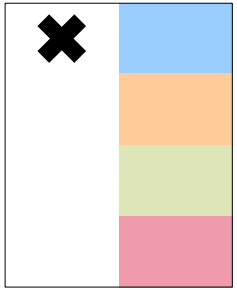



```
void pop() {
  if (!empty()) {
    storage[head--] = null;
  }
}
```

¬Functional

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



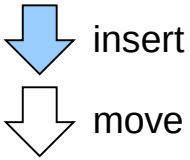


Debug

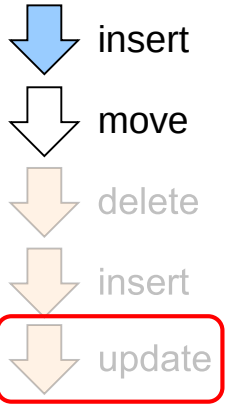
Functional

⋮

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



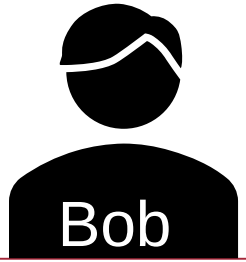

```
void pop() {/*...*/}
```

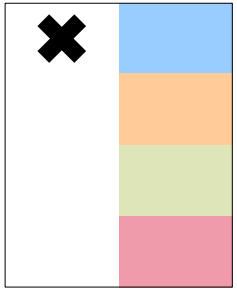



```
void pop() {
  if (!empty()) {
    storage[head--] = null;
  }
}
```

¬Functional

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



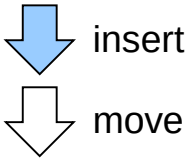


Debug

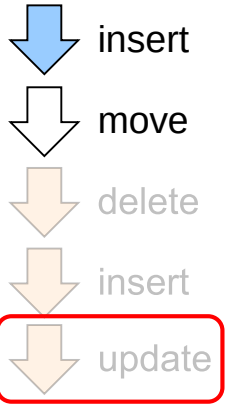
Functional

⋮

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



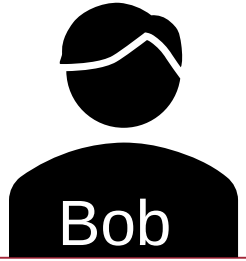

```
void pop() {/*...*/}
```

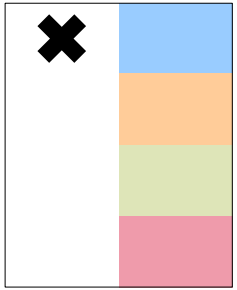



```
void pop() {
  if (!empty()) {
    storage[head--] = null;
  }
}
```

¬Functional

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



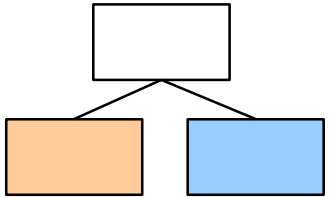


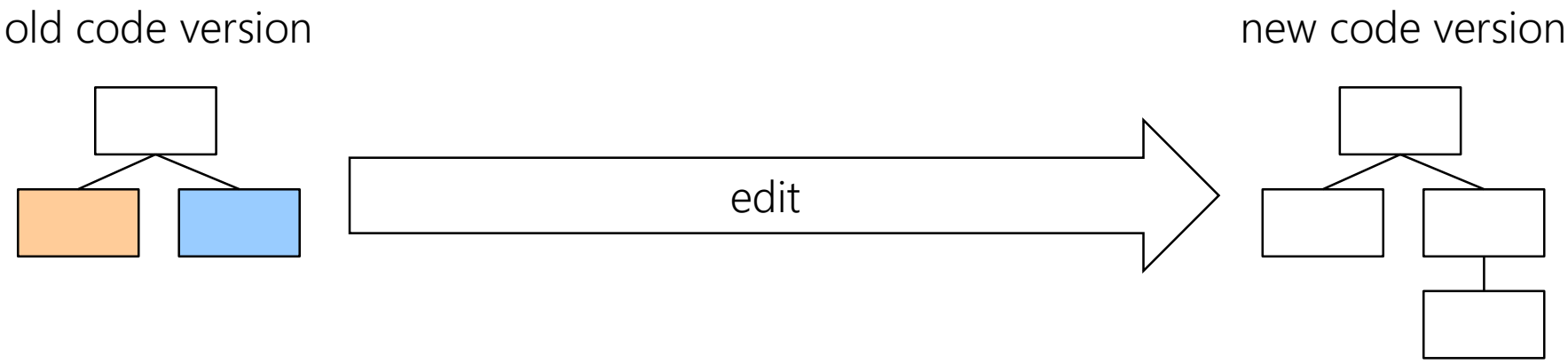
Debug

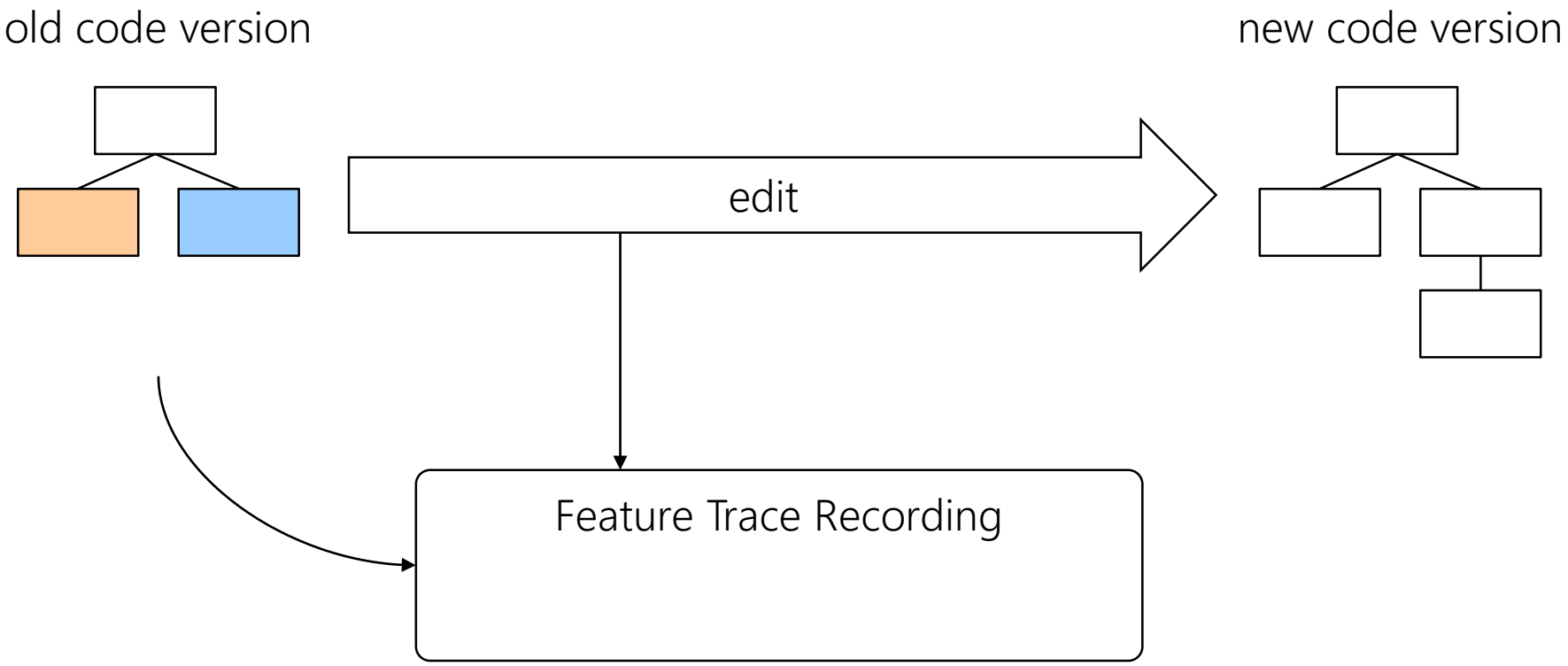
Functional

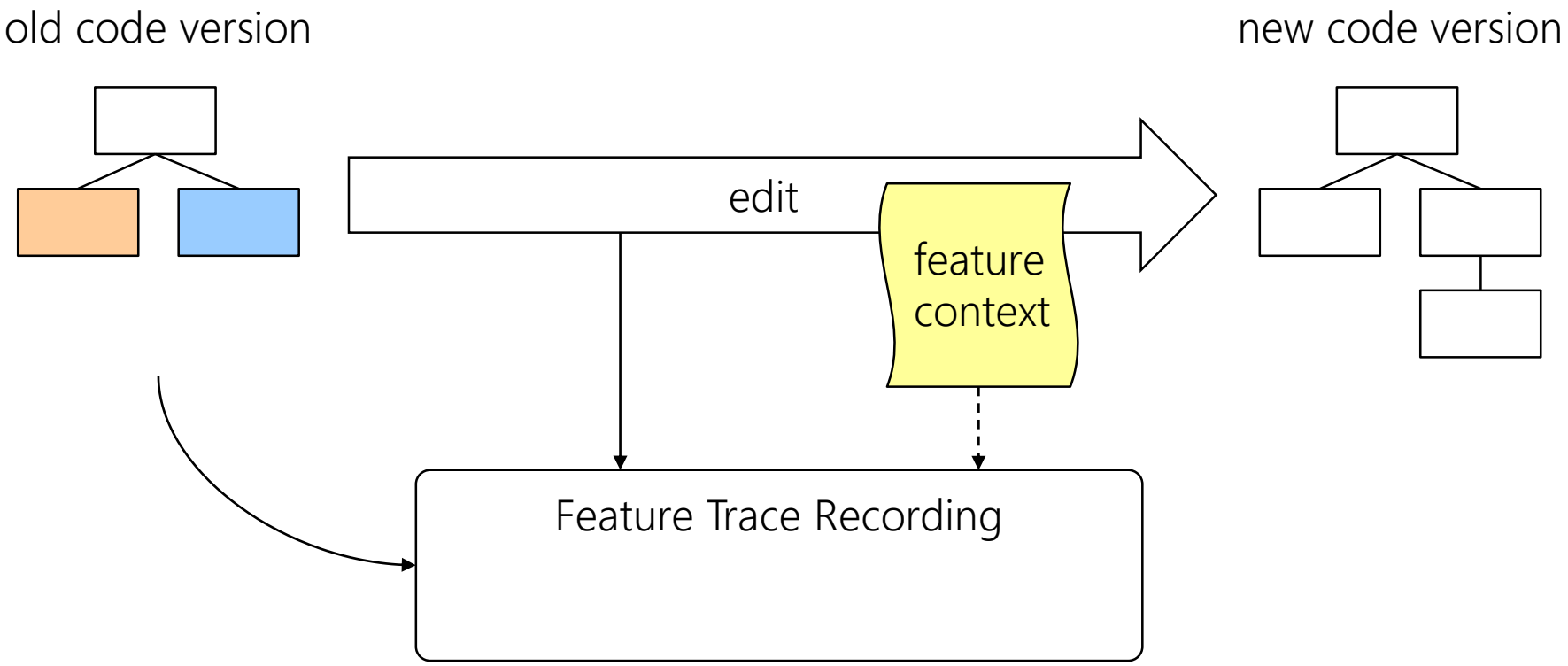
⋮

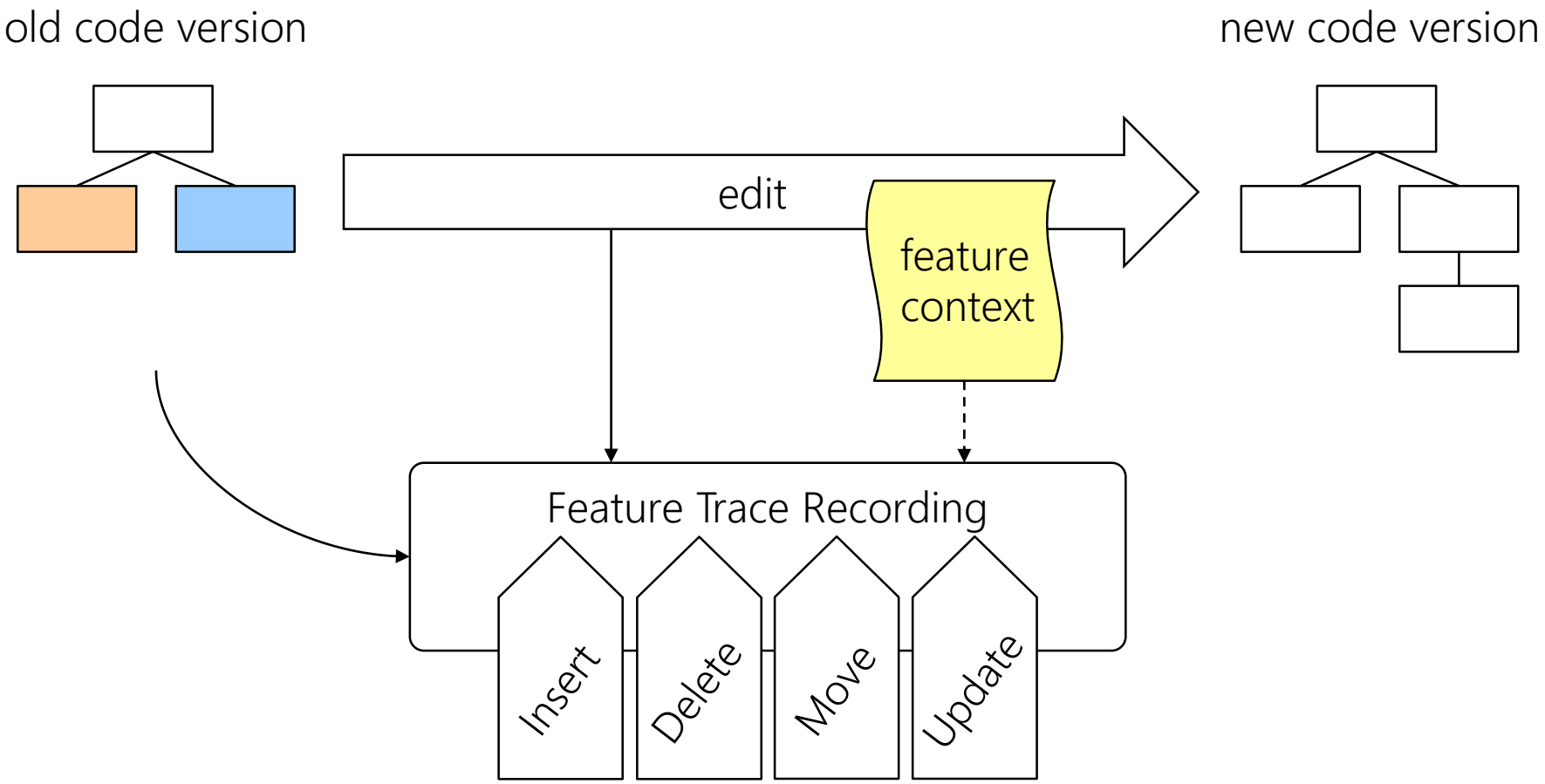
old code version

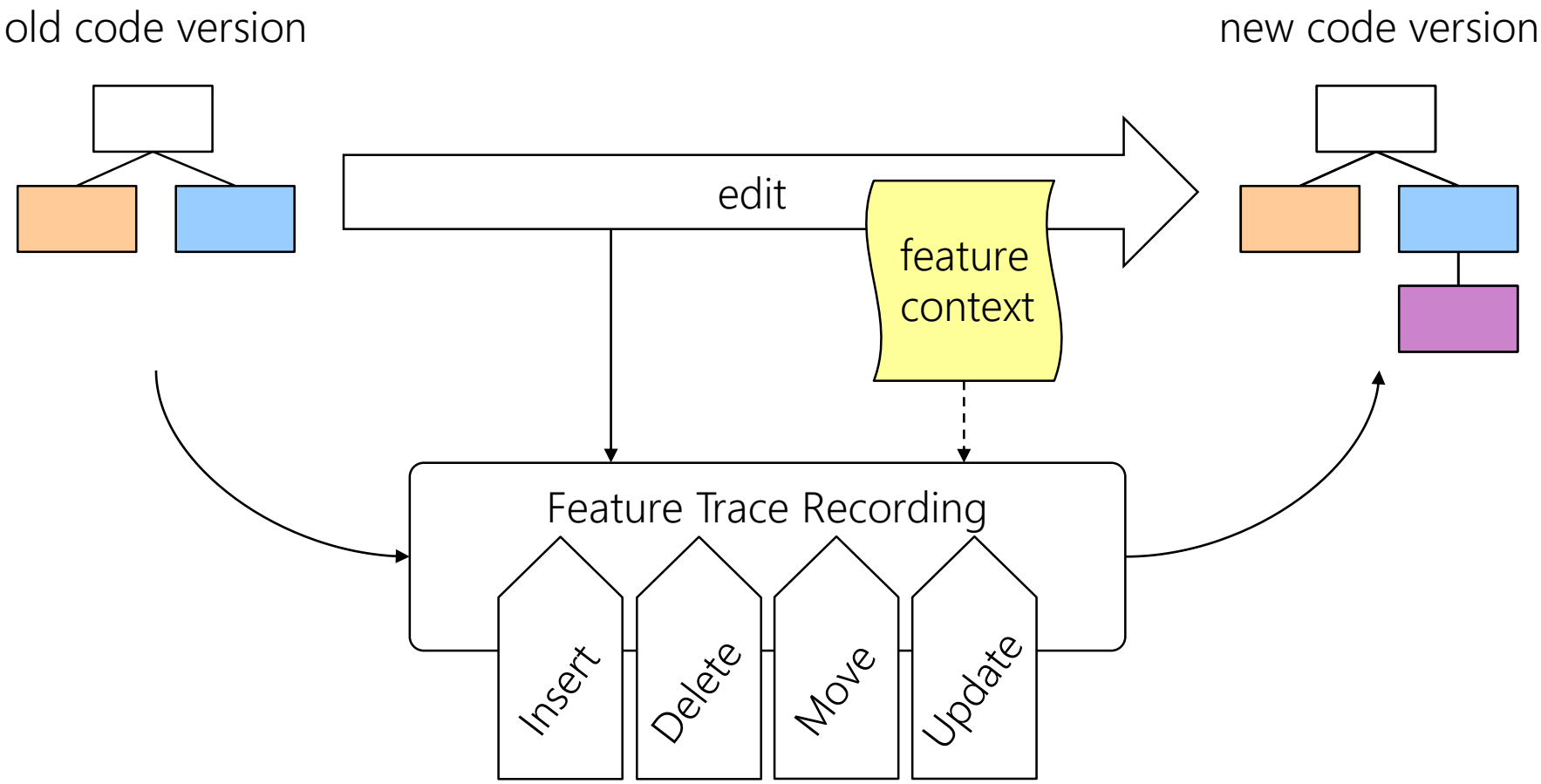



old code version
new code version
edit


old code version
new code version
edit
Feature Trace Recording

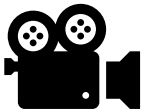

old code version
new code version
edit
feature context
Feature Trace Recording


old code version
new code version
edit
feature context
Feature Trace Recording
Insert
Delete
Move
Update

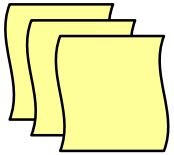

old code version
new code version
edit
feature context
Feature Trace Recording
Insert
Delete
Move
Update

Evaluation

RQ1 – Can we record feature traces upon common edits?

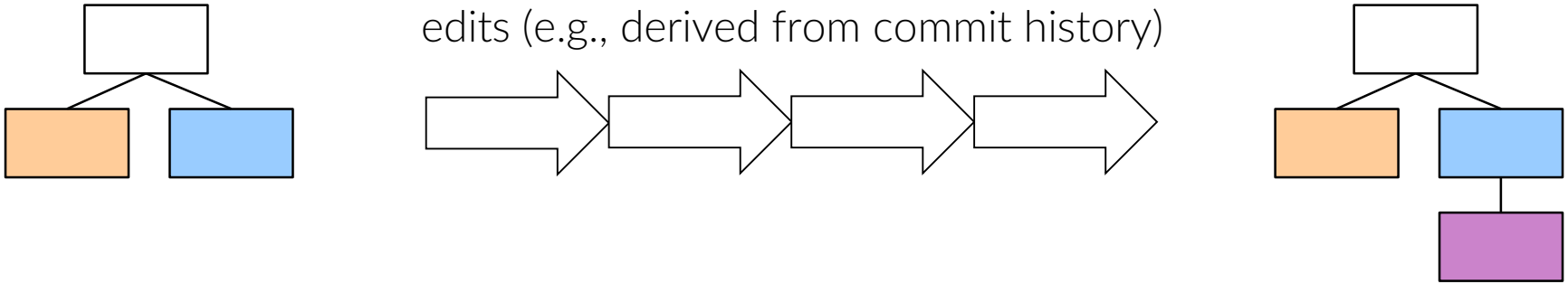


RQ2 – How many feature contexts are necessary?

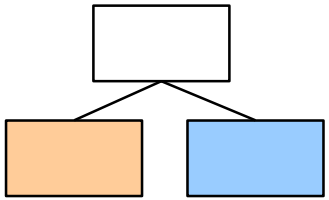


RQ3 – How complex are the feature contexts?

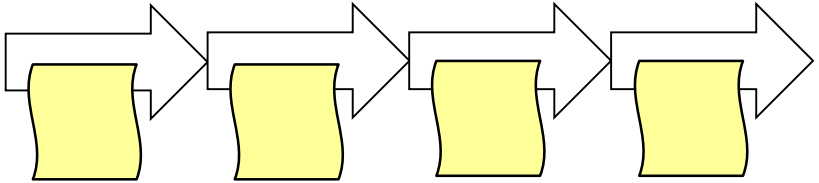
To evaluate feature trace recording we need …



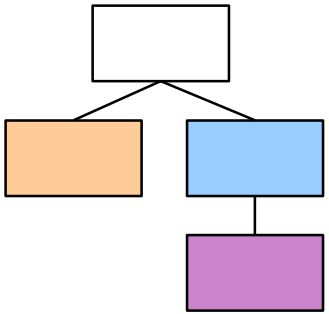
To evaluate feature trace recording we need …



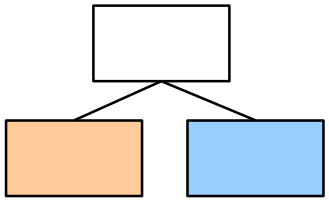
edits (e.g., derived from commit history)



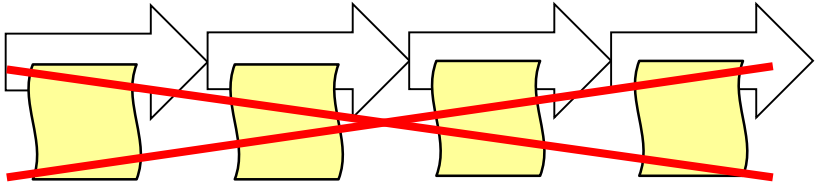
feature contexts

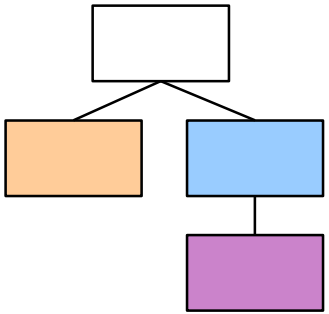


To evaluate feature trace recording we need …



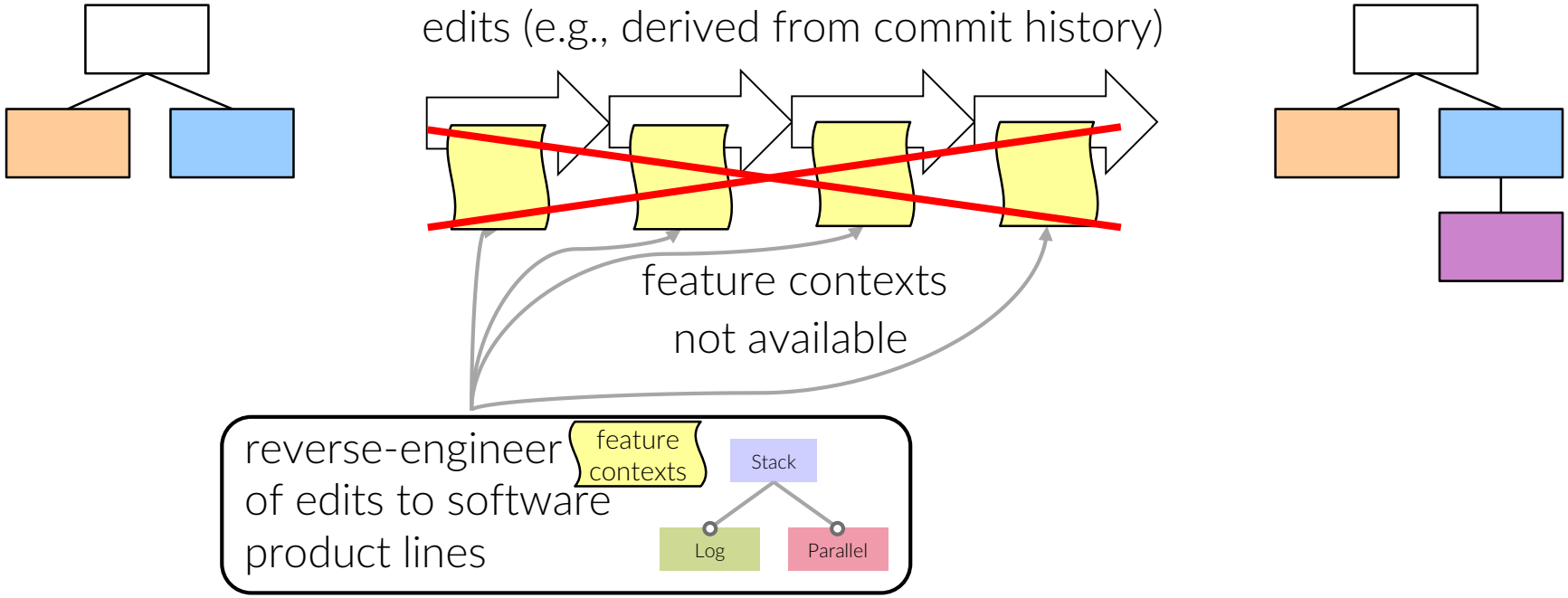
edits (e.g., derived from commit history)



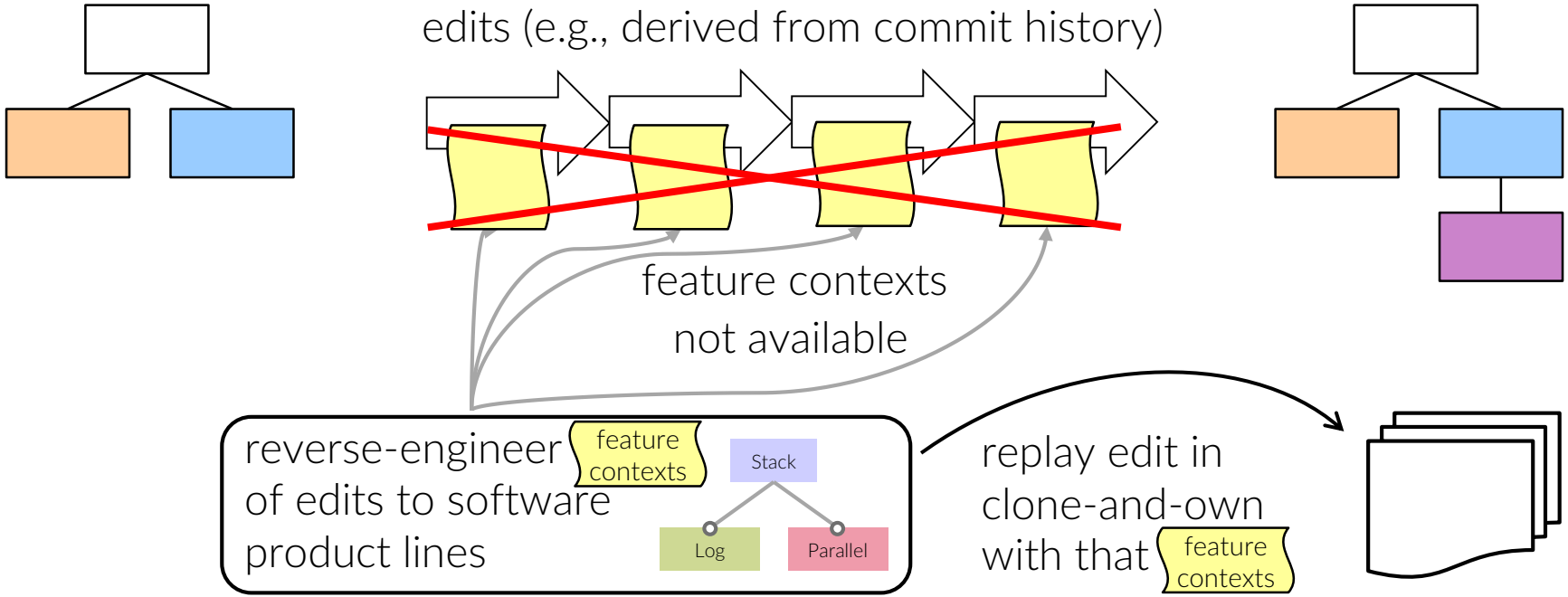


feature contexts
not available

To evaluate feature trace recording we need …



To evaluate feature trace recording we need …



Can we reproduce typical edits to variability?

Reverse-engineer the feature contexts of the 8 edit patterns from:

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

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


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

type of edit to software product line

Software Engineering
Programming Languages

Can we reproduce typical edits to variability?

Reverse-engineer the feature contexts of the 8 edit patterns from:

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

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


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

decompose

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

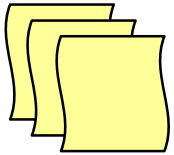
type of edit to software product line

type of edit to variants

Results

RQ1 – Can we reproduce all considered kinds of edits?

RQ2 – How many feature contexts are necessary?



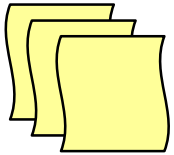
RQ3 – How complex are the feature contexts?

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?

less or as many as when manually specifying mappings

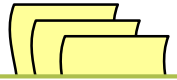
RQ3 – How complex are the feature contexts?

Results

RQ1 – Can we reproduce all considered kinds of edits?

Yes

RQ2 – How many feature contexts are necessary?

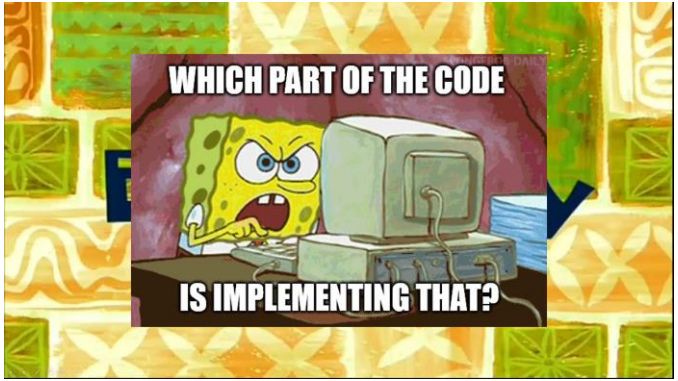


less or as many as when manually specifying mappings

RQ3 – How complex are the feature contexts?

equal to target feature mapping

Feature Trace Recording




```
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;
}
```

update

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

done with single feature context = Functional

Hey Alice, can I merge your changes?

Sure! ☺

But I have another variant!

That's fine, I recorded all feature traces!

Can we reproduce edits to SPLs as edits to variants?


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

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


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

Sören Viegener

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

edit to SPL

decompose

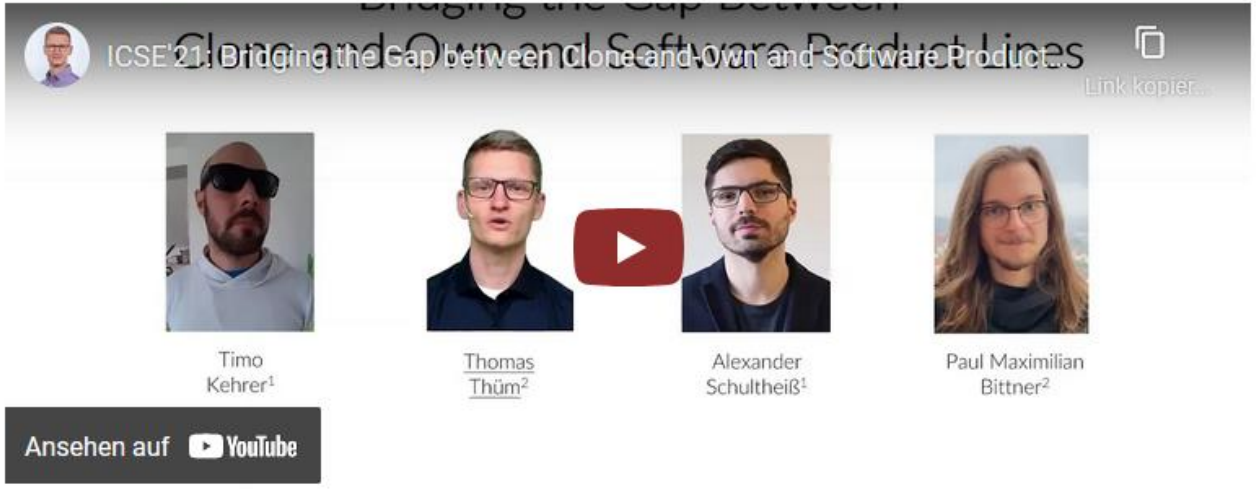
insert code into a variant implementing m (then merge)

m

edit to variants

Software Engineering
Programming Languages

VariantSync



Automating the Synchronisation of Software Variants

Today's software is often released in multiple variants to meet all customer requirements. Software product lines have the potential to decrease development costs and time-to-market, and have been actively researched for more than two decades. Nevertheless, practitioners frequently rely on ad-hoc reuse based on a principle which is known as clone-and-own, where new variants of a software family are created by copying and adapting an existing variant. However, if a critical number of variants is reached, their maintenance and evolution becomes impractical, if not impossible, and the migration to a product line is often infeasible. With the research conducted in VariantSync, we aim to enable a fundamentally new development approach which bridges the gap between clone-and-own and product lines, combining the minimal overhead of clone-and-own with the systematic handling of variability of software product lines in a highly flexible methodology. The key idea is to transparently integrate the central product-line concept of a feature with variant management facilities known from version control

Contact

variantsync[at]uni-ulm.de

Project Members:

Prof. Dr. Thomas Thüm

Prof. Dr. Timo Kehrer

Paul Maximilian Bittner

Alexander Schultheiß

Funding

German Research Foundation: TH 2387/1-1 and KE 2267/1-1

Project Vision Paper

We describe our vision for *Bridging the Gap Between Clone-and-Own and Software Product Lines* with VariantSync