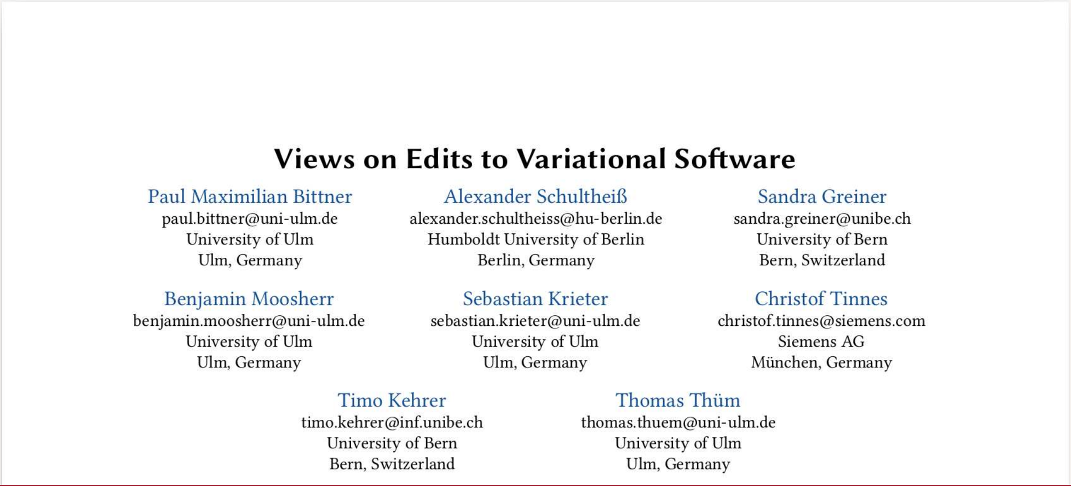


Views on Edits to Variational Software

Paul Maximilian Bittner
paul.bittner@uni-ulm.de
University of Ulm
Ulm, Germany

Alexander Schultheiß
alexander.schultheiss@hu-berlin.de
Humboldt University of Berlin
Berlin, Germany

Sandra Greiner
sandra.greiner@unibe.ch
University of Bern
Bern, Switzerland

Benjamin Moosherr
benjamin.moosherr@uni-ulm.de
University of Ulm
Ulm, Germany

Sebastian Krieter
sebastian.krieter@uni-ulm.de
University of Ulm
Ulm, Germany

Christof Tinnes
christof.tinnes@siemens.com
Siemens AG
München, Germany

Timo Kehrer
timo.kehrer@inf.unibe.ch
University of Bern
Bern, Switzerland

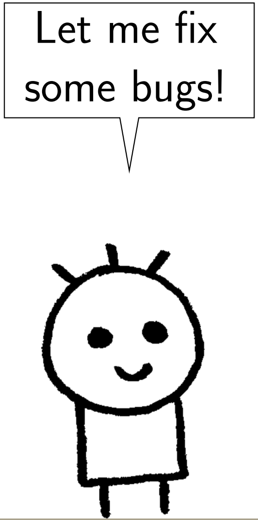
Thomas Thüm
thomas.thuem@uni-ulm.de
University of Ulm
Ulm, Germany

SPLC'23 | Tokyo, Japan

speaker: Paul Bittner

This is Bob,
a software developer.

This is Bob,
a software developer.



```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

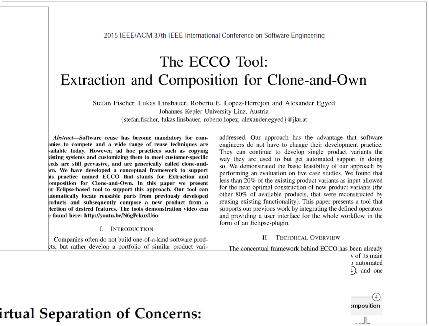
```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

```
void prep
  Itm* new
  newHead
#if Ring
  if
    I
#er
  la
```

Concepts of Variation Control Systems

Lukas Linsbauer[a], Felix Schwägerl[b], Thorsten Berger[c], Paul Grünbacher[d]

[a]*ISF, Technische Universität Braunschweig, Germany*
E-Mail: l.linsbauer@tu-braunschweig.de
[b]*Applied Computer Science I, University of Bayreuth, Germany*
E-Mail: felix.schwaegerl@uni-bayreuth.de
[c]*Chalmers | University of Gothenburg, Sweden*
E-Mail: thorsten.berger@cse.gu.se
[d]*Institute for Software Systems Engineering, Johannes Kepler University Linz, Austria*
E-Mail: paul.gruenbacher@jku.at



2015 IEEE/ACM 37th IEEE International Conference on Software Engineering

The ECCO Tool:
Extraction and Composition for Clone-and-Own

Stefan Fischer, Lukas Linsbauer, Roberto E. Lopez-Herrejon and Alexander Egyed
Johannes Kepler University Linz, Austria
{stefan.fischer, lukas.linsbauer, roberto.lopez, alexander.egyed}@jku.at

Abstract—**Software reuse has become mandatory for com-
anies to compete and a wide range of reuse techniques are
vailable today. However, ad hoc practices such as copying
xisting systems and customizing them to meet customer-specific
eeds are still pervasive, and are generically called clone-and-
wn. We have developed a conceptual framework to support
is practice named ECCO that stands for Extraction and
omposition for Clone-and-Own. In this paper we present
ur Eclipse-based tool to support this approach. Our tool can
utomatically locate reusable parts from previously developed
roducts and subsequently compose a new product from a
lection of desired features. The tools demonstration video can
e found here: http://youtu.be/N6gPekuxU6o**

I. Introduction

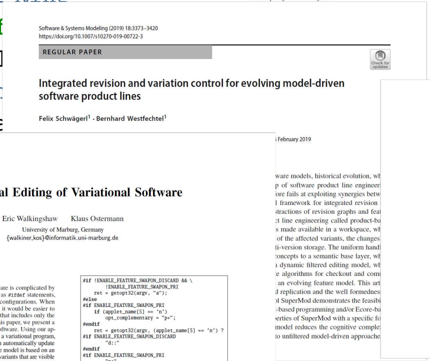
Companies often do not build one-of-a-kind software prod-
cts, but rather develop a portfolio of similar product vari-

addressed. Our approach has the advantage that software engineers do not have to change their development practice. They can continue to develop single product variants the way they are used to but get automated support in doing so. We demonstrated the basic feasibility of our approach by performing an evaluation on five case studies. We found that less than 20% of the existing product variants as input allowed for the near optimal construction of new product variants (the other 80% of available products, that were reconstructed by reusing existing functionality). This paper presents a tool that supports our previous work by integrating the defined operators and providing a user interface for the whole workflow in the form of an Eclipse-plugin.

II. Technical Overview

The conceptual framework behind ECCO has been already
v of its main
o automated
4), and one

mposition



Software & Systems Modeling (2019) 18:3373–3420
https://doi.org/10.1007/s10270-019-00722-3

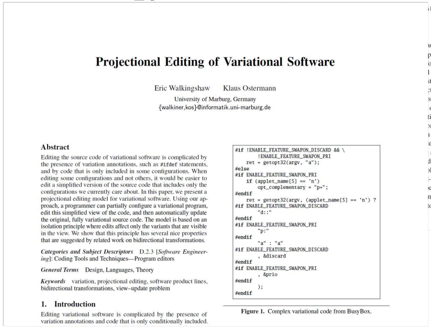
REGULAR PAPER

Integrated revision and variation control for evolving model-driven software product lines

Felix Schwägerl[1] · Bernhard Westfechtel[1]

February 2019

ware models, historical evolution, wh
p of software product line engineeri
ore fails at exploiting synergies betw
l framework for integrated revision
stractions of revision graphs and feat
ct line engineering called product-ba
s made available in a workspace, wh
of the affected variants, the changes
ti-version storage. The uniform handl
concepts to a semantic base layer, wh
a dynamic filtered editing model, wh
e algorithms for checkout and comi
an evolving feature model. This art
d replication and the well formednes
ol SuperMod demonstrates the feasibi
-based programming and/or Ecore-ba
erties of SuperMod with a specific fo
model reduces the cognitive comple
to unfiltered model-driven approache



Projectional Editing of Variational Software

Eric Walkingshaw Klaus Ostermann
University of Marburg, Germany
{walkiner,kos}@informatik.uni-marburg.de

Abstract

Editing the source code of variational software is complicated by the presence of variation annotations, such as `#ifdef` statements, and by code that is only included in some configurations. When editing some configurations and not others, it would be easier to edit a simplified version of the source code that includes only the configurations we currently care about. In this paper, we present a projectional editing model for variational software. Using our approach, a programmer can partially configure a variational program, edit this simplified view of the code, and then automatically update the original, fully variational source code. The model is based on an isolation principle where edits affect only the variants that are visible in the view. We show that this principle has several nice properties that are suggested by related work on bidirectional transformations.

Categories and Subject Descriptors D.2.3 [*Software Engineering*]: Coding Tools and Techniques—Program editors

General Terms Design, Languages, Theory

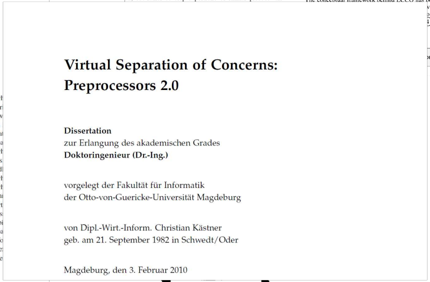
Keywords variation, projectional editing, software product lines, bidirectional transformations, view–update problem

1. Introduction

Editing variational software is complicated by the presence of variation annotations and code that is only conditionally included.

```
#if !ENABLE_FEATURE_SWAPON_DISCARD && \
        !ENABLE_FEATURE_SWAPON_PRI
    ret = getopt32(argv, "a");
#else
#if ENABLE_FEATURE_SWAPON_PRI
    if (applet_name[5] == 'n')
        opt_complementary = "p+";
#endif
    ret = getopt32(argv, (applet_name[5] == 'n') ?
#if ENABLE_FEATURE_SWAPON_DISCARD
        "d::"
#endif
#if ENABLE_FEATURE_SWAPON_PRI
        "p:"
#endif
        "a" : "a"
#if ENABLE_FEATURE_SWAPON_DISCARD
        , &discard
#endif
#if ENABLE_FEATURE_SWAPON_PRI
        , &prio
#endif
        );
#endif
```

Figure 1. Complex variational code from BusyBox.



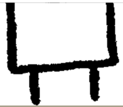
**Virtual Separation of Concerns:
Preprocessors 2.0**

Dissertation
zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt der Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von Dipl.-Wirt.-Inform. Christian Kästner
geb. am 21. September 1982 in Schwedt/Oder

Magdeburg, den 3. Februar 2010



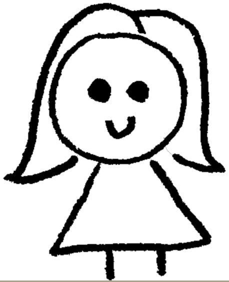
```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;




 last->suc = newHead;
#if DoubleLink
 head->prev = head;



#endif
 head = newHead;
}
```

One week later. . .

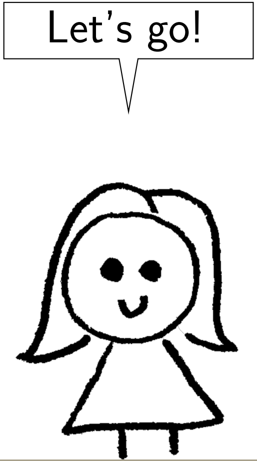
This is Alice.

She has to do a **code review** on recent changes to the `prepend` method.

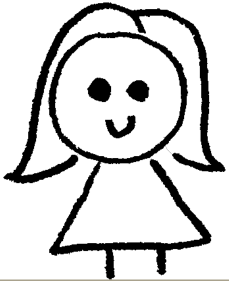


This is Alice.

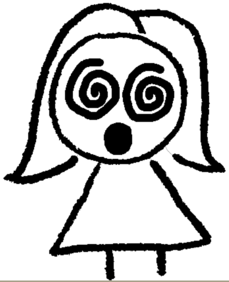
She has to do a **code review** on recent changes to the `prepend` method.



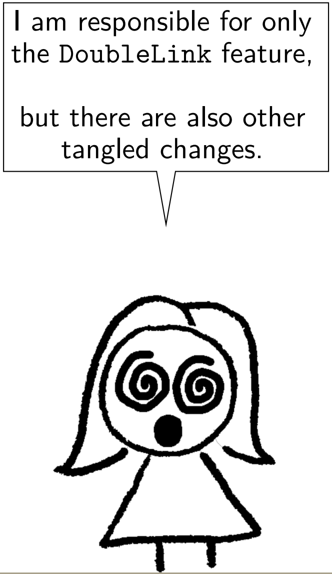
```
 void prepend(T e) {
  Itm* newHead = new Itm(e);
  newHead->suc = head;
 #if Ring
  if (empty())
-   last = head;
+   last = newHead;
-#endif
  last->suc = newHead;
+#endif
 #if DoubleLink
+ if (head) {
- head->prev = head;
+   head->prev = newHead;
+ }
  #if Ring
   newHead->prev = last;
  #endif
 #endif
  head = newHead;
 }
```



```
 void prepend(T e) {
  Itm* newHead = new Itm(e);
  newHead->suc = head;
 #if Ring
  if (empty())
-   last = head;
+   last = newHead;
-#endif
  last->suc = newHead;
+#endif
 #if DoubleLink
+ if (head) {
- head->prev = head;
+   head->prev = newHead;
+ }
  #if Ring
   newHead->prev = last;
  #endif
 #endif
  head = newHead;
 }
```



```
 void prepend(T e) {
  Itm* newHead = new Itm(e);
  newHead->suc = head;
 #if Ring
  if (empty())
-   last = head;
+   last = newHead;
-#endif
  last->suc = newHead;
+#endif
 #if DoubleLink
+ if (head) {
- head->prev = head;
+   head->prev = newHead;
+ }
  #if Ring
   newHead->prev = last;
  #endif
 #endif
  head = newHead;
 }
```


I am responsible for only the DoubleLink feature,
but there are also other tangled changes.

An Open Problem!

```
  head
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = head;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = head;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

$\xrightarrow{\text{view on state before}}$

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;



 last->suc = newHead;
#if DoubleLink
 head->prev = head;



#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = head;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

view on state before

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;


 last->suc = newHead;
#if DoubleLink
 head->prev = head;



#endif
 head = newHead;
}
```

edit

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;




#if DoubleLink
 if (head) {
  head->prev = newHead;
 }



#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = head;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

view on state before

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;




 last->suc = newHead;
#if DoubleLink
 head->prev = head;



#endif
 head = newHead;
}
```

edit

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;






#if DoubleLink
 if (head) {
  head->prev = newHead;
 }



#endif
 head = newHead;
}
```

commit to variation control

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
 last->suc = newHead;
#endif
#if DoubleLink
 if (head) {
  head->prev = newHead;
 }
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = head;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

view on state before

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;



 last->suc = newHead;
#if DoubleLink
 head->prev = head;



#endif
 head = newHead;
}
```

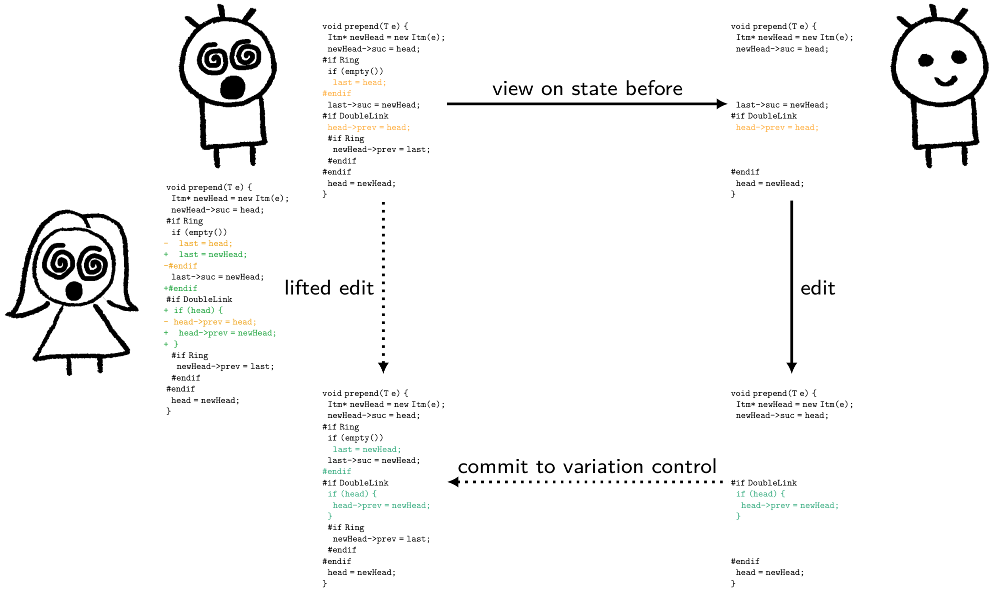
lifted edit

edit

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
 last->suc = newHead;
#endif
#if DoubleLink
 if (head) {
  head->prev = newHead;
 }
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

commit to variation control

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;





#if DoubleLink
 if (head) {
  head->prev = newHead;
 }



#endif
 head = newHead;
}
```



```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = head;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;



 last->suc = newHead;
#if DoubleLink
 head->prev = head;



#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
-  last = head;
+  last = newHead;
-#endif
 last->suc = newHead;
+#endif
#if DoubleLink
+ if (head) {
- head->prev = head;
+  head->prev = newHead;
+ }
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

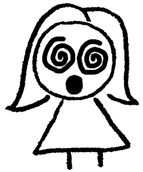
```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
 last->suc = newHead;
#endif
#if DoubleLink
 if (head) {
  head->prev = newHead;
 }
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;





#if DoubleLink
 if (head) {
  head->prev = newHead;
 }



#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = head;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

view on state before

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;



 last->suc = newHead;
#if DoubleLink
 head->prev = head;



#endif
 head = newHead;
}
```



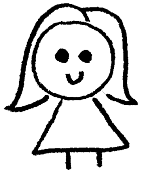
```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
-  last = head;
+  last = newHead;
-#endif
 last->suc = newHead;
+#endif
#if DoubleLink
+ if (head) {
- head->prev = head;
+  head->prev = newHead;
+ }
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

lifted edit

Open Problem

edit

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;



- last->suc = newHead;

#if DoubleLink
+ if (head) {
- head->prev = head;
+  head->prev = newHead;
+ }



#endif
 head = newHead;
}
```

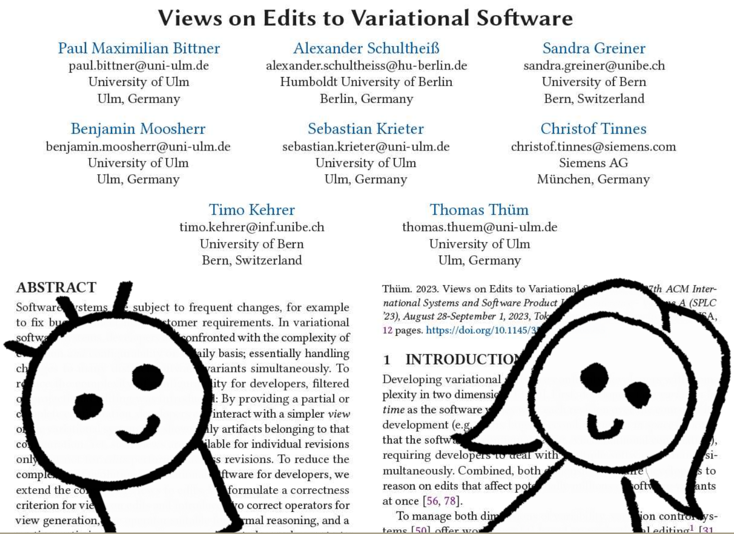


```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
 last->suc = newHead;
#endif
#if DoubleLink
 if (head) {
  head->prev = newHead;
 }
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

commit to variation control

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;




#if DoubleLink
 if (head) {
  head->prev = newHead;
 }



#endif
 head = newHead;
}
```

Our Contribution



Views on Edits to Variational Software

Paul Maximilian Bittner
paul.bittner@uni-ulm.de
University of Ulm
Ulm, Germany

Alexander Schultheiß
alexander.schultheiss@hu-berlin.de
Humboldt University of Berlin
Berlin, Germany

Sandra Greiner
sandra.greiner@unibe.ch
University of Bern
Bern, Switzerland

Benjamin Moosherr
benjamin.moosherr@uni-ulm.de
University of Ulm
Ulm, Germany

Sebastian Krieter
sebastian.krieter@uni-ulm.de
University of Ulm
Ulm, Germany

Christof Tinnes
christof.tinnes@siemens.com
Siemens AG
München, Germany

Timo Kehrer
timo.kehrer@inf.unibe.ch
University of Bern
Bern, Switzerland

Thomas Thüm
thomas.thuem@uni-ulm.de
University of Ulm
Ulm, Germany

ABSTRACT

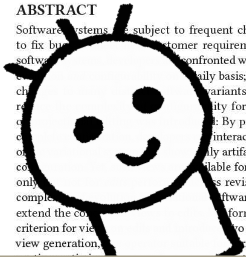
Software ystems e subject to frequent changes, for example to fix bu stomer requirements. In variational softwa confronted with the complexity of e aily basis; essentially handling ch variants simultaneously. To re ity for developers, filtered o : By providing a partial or c interact with a simpler *view* of ly artifacts belonging to that co ilable for individual revisions only s revisions. To reduce the comple ftware for developers, we extend the co formulate a correctness criterion for vie o correct operators for view generation, rmal reasoning, and a

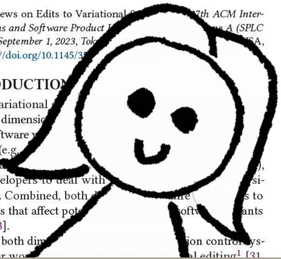
Thüm. 2023. Views on Edits to Variational S 7th ACM Inter- *national Systems and Software Product L e A (SPLC '23), August 28-September 1, 2023, Tok SA,* 12 pages. https://doi.org/10.1145/3

1 INTRODUCTIO

Developing variational plexity in two dimensio *time* as the software development (e.g. that the softwa), requiring developers to deal with si- multaneously. Combined, both re s to reason on edits that affect pot oftw ants at once [56, 78].

To manage both dim ion control sys- tems [50] offer wo l editing[1] [31

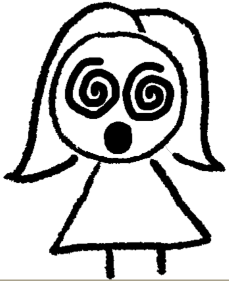




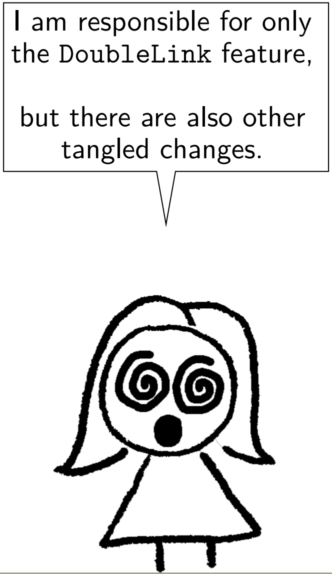
```
 void prepend(T e) {
  Itm* newHead = new Itm(e);
  newHead->suc = head;
 #if Ring
  if (empty())
-   last = head;
+   last = newHead;
-#endif
  last->suc = newHead;
+#endif
 #if DoubleLink
+ if (head) {
- head->prev = head;
+   head->prev = newHead;
+ }
  #if Ring
   newHead->prev = last;
  #endif
 #endif
  head = newHead;
 }
```

I am responsible for only the `DoubleLink` feature,

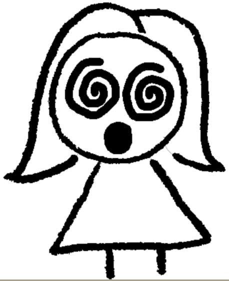
but there are also other tangled changes.



```
 void prepend(T e) {
  Itm* newHead = new Itm(e);
  newHead->suc = head;
 #if Ring
  if (empty())
-   last = head;
+   last = newHead;
-#endif
  last->suc = newHead;
+#endif
 #if DoubleLink
+ if (head) {
- head->prev = head;
+   head->prev = newHead;
+ }
  #if Ring
   newHead->prev = last;
  #endif
 #endif
  head = newHead;
 }
```

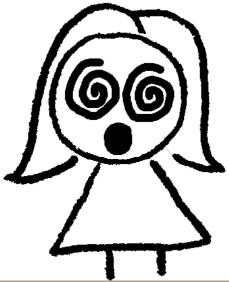

I am responsible for only the DoubleLink feature, but there are also other tangled changes.

```
void prepend(T e) {
  Itm* newHead = new Itm(e);
  newHead->suc = head;
#if Ring




-#endif
  last->suc = newHead;
+#endif
#if DoubleLink
+ if (head) {
- head->prev = head;
+   head->prev = newHead;
+ }




#endif
  head = newHead;
}
```

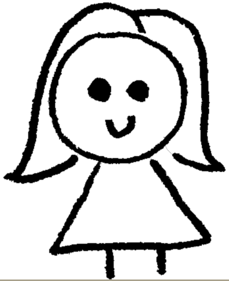


```
 void prepend(T e) {
  Itm* newHead = new Itm(e);
  newHead->suc = head;
 #if Ring



-#endif
  last->suc = newHead;
+#endif
 #if DoubleLink
+ if (head) {
-  head->prev = head;
+   head->prev = newHead;
+ }



 #endif
  head = newHead;
 }
```

feature annotation
changed from
true to Ring



```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;





- last->suc = newHead;

#if DoubleLink
+ if (head) {
- head->prev = head;
+  head->prev = newHead;
+ }



#endif
 head = newHead;
}
```



How does it work?

What is a view (on state / a single revision)?

A view is a self-contained subset of a system

What is a view (on state / a single revision)?

A view is a self-contained subset of a system

that consists exactly of all relevant parts of the system

What is a view (on state / a single revision)?

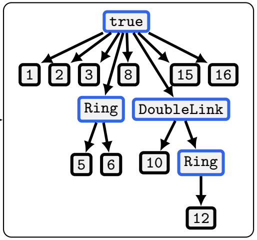
A view is a self-contained subset of a system

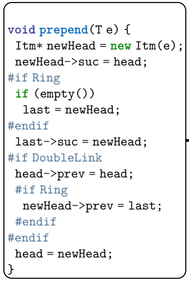
that consists exactly of all relevant parts of the system

where relevance is decided by an oracle

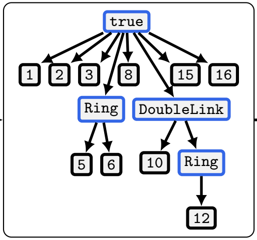
(e.g., a developer or analysis tool).

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```





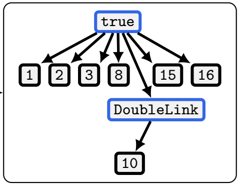
```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```



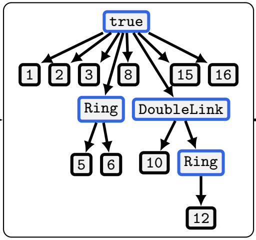
Predicate over nodes

Remove all nodes that do not satisfy the predicate.

(Keep ancestors of selected nodes for self-containedness.)



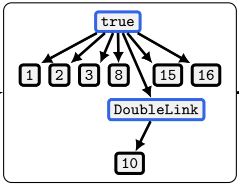
```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```



Predicate over nodes

Remove all nodes that do not satisfy the predicate.

(Keep ancestors of selected nodes for self-containedness.)



```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;




 last->suc = newHead;
#if DoubleLink
 head->prev = head;



#endif
 head = newHead;
}
```

View on a Variant (Set)
by (partial) configuration

[Walkingshaw and Ostermann, 2014]
[Kästner, 2010]
variation control systems

View on Features
i.e., feature traces

[Kästner, 2010]

View on Artifacts
search in code

View on a Variant (Set)
by (partial) configuration

[Walkingshaw and Ostermann, 2014]
[Kästner, 2010]
variation control systems

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

View on Features
i.e., feature traces

[Kästner, 2010]

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

View on Artifacts
search in code

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

View on a Variant (Set)
by (partial) configuration

[Walkingshaw and Ostermann, 2014]
[Kästner, 2010]
variation control systems

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

SAT($FM \land \neg$Ring $\land$ PC(v))

View on Features
i.e., feature traces

[Kästner, 2010]

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

View on Artifacts
search in code

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

View on a Variant (Set)
by (partial) configuration

[Walkingshaw and Ostermann, 2014]
[Kästner, 2010]
variation control systems

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

$\mathrm{SAT}(FM \land \neg\texttt{Ring} \land \mathrm{PC}(v))$

View on Features
i.e., feature traces

[Kästner, 2010]

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

View on Artifacts
search in code

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

View on a Variant (Set)
by (partial) configuration

[Walkingshaw and Ostermann, 2014]
[Kästner, 2010]
variation control systems

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

$\mathrm{SAT}(\mathit{FM} \land \neg\texttt{Ring} \land \mathrm{PC}(v))$

View on Features
i.e., feature traces

[Kästner, 2010]

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

$\texttt{DoubleLink} \in \mathit{vars}(\mathrm{PC}(v))$

View on Artifacts
search in code

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

View on a Variant (Set)
by (partial) configuration

[Walkingshaw and Ostermann, 2014]
[Kästner, 2010]
variation control systems

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

$\mathrm{SAT}(FM \land \neg\texttt{Ring} \land \mathrm{PC}(v))$

View on Features
i.e., feature traces

[Kästner, 2010]

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

$\texttt{DoubleLink} \in vars(\mathrm{PC}(v))$

View on Artifacts
search in code

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

View on a Variant (Set)
by (partial) configuration

[Walkingshaw and Ostermann, 2014]
[Kästner, 2010]
variation control systems

```
void prepend(T e) {
 Itm∗ newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

$\mathrm{SAT}(FM \land \neg\texttt{Ring} \land \mathrm{PC}(v))$

View on Features
i.e., feature traces

[Kästner, 2010]

```
void prepend(T e) {
 Itm∗ newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

$\texttt{DoubleLink} \in vars(\mathrm{PC}(v))$

View on Artifacts
search in code

```
void prepend(T e) {
 Itm∗ newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

$label(v) = \texttt{"last = newHead;"}$

View on a Variant (Set)
by (partial) configuration

[Walkingshaw and Ostermann, 2014]
[Kästner, 2010]
variation control systems

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

$\mathsf{SAT}(FM \land \neg\texttt{Ring} \land \mathsf{PC}(v))$

View on Features
i.e., feature traces

[Kästner, 2010]

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

$\texttt{DoubleLink} \in vars(\mathsf{PC}(v))$

View on Artifacts
search in code

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

$label(v) = \texttt{"last = newHead;"}$

… but what is a view on an edit …

```
 void prepend(T e) {
  Itm* newHead = new Itm(e);
  newHead->suc = head;
 #if Ring
  if (empty())
-   last = head;
+   last = newHead;
-#endif
  last->suc = newHead;
+#endif
 #if DoubleLink
+ if (head) {
-  head->prev = head;
+   head->prev = newHead;
+ }
  #if Ring
   newHead->prev = last;
  #endif
 #endif
  head = newHead;
 }
```

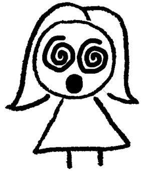
… and how can we compute it? $\longrightarrow$

```
 void prepend(T e) {
  Itm* newHead = new Itm(e);
  newHead->suc = head;




- last->suc = newHead;

 #if DoubleLink
+ if (head) {
- head->prev = head;
+   head->prev = newHead;
+ }



 #endif
  head = newHead;
 }
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = head;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

view on state before

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;


 last->suc = newHead;
#if DoubleLink
 head->prev = head;


#endif
 head = newHead;
}
```



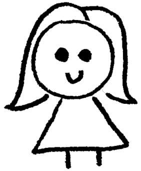
```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
-  last = head;
+  last = newHead;
-#endif
 last->suc = newHead;
+#endif
#if DoubleLink
+ if (head) {
- head->prev = head;
+  head->prev = newHead;
+ }
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

complex edit

… and how can we compute it?

edit

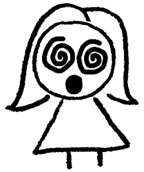
```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;



- last->suc = newHead;

#if DoubleLink
+ if (head) {
- head->prev = head;
+  head->prev = newHead;
+ }


#endif
 head = newHead;
}
```



```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
 last->suc = newHead;
#endif
#if DoubleLink
 if (head) {
  head->prev = newHead;
 }
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

commit to variation control

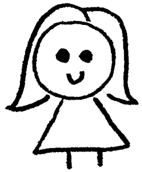
```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;




#if DoubleLink
 if (head) {
  head->prev = newHead;
 }


#endif
 head = newHead;
}
```



```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = head;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

view on state before

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;


 last->suc = newHead;
#if DoubleLink
 head->prev = head;


#endif
 head = newHead;
}
```



```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
-  last = head;
+  last = newHead;
-#endif
 last->suc = newHead;
+#endif
#if DoubleLink
+ if (head) {
-  head->prev = head;
+   head->prev = newHead;
+ }
  #if Ring
   newHead->prev = last;
  #endif
#endif
 head = newHead;
}
```

complex edit

Viewing an edit to the SPL should be equivalent to editing a view on the SPL.

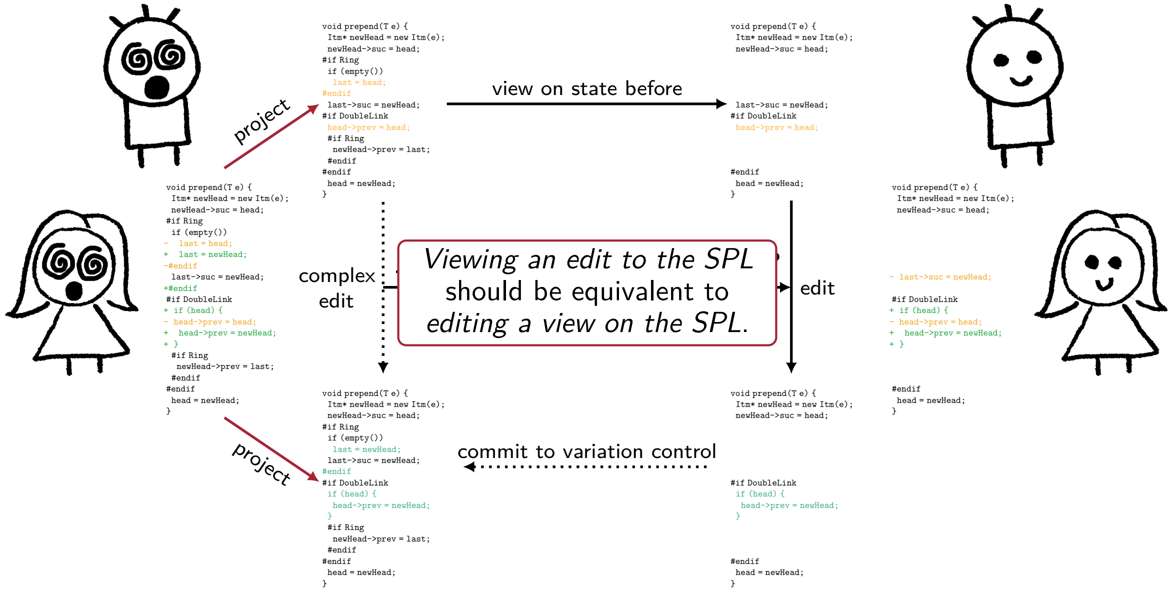
edit

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;

- last->suc = newHead;

#if DoubleLink
+ if (head) {
-  head->prev = head;
+   head->prev = newHead;
+ }


#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
 last->suc = newHead;
#endif
#if DoubleLink
 if (head) {
  head->prev = newHead;
 }
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

commit to variation control

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;



#if DoubleLink
 if (head) {
  head->prev = newHead;
 }


#endif
 head = newHead;
}
```



```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
-  last = head;
+  last = newHead;
-#endif
 last->suc = newHead;
+#endif
#if DoubleLink
+ if (head) {
-  head->prev = head;
+   head->prev = newHead;
+ }
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = head;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

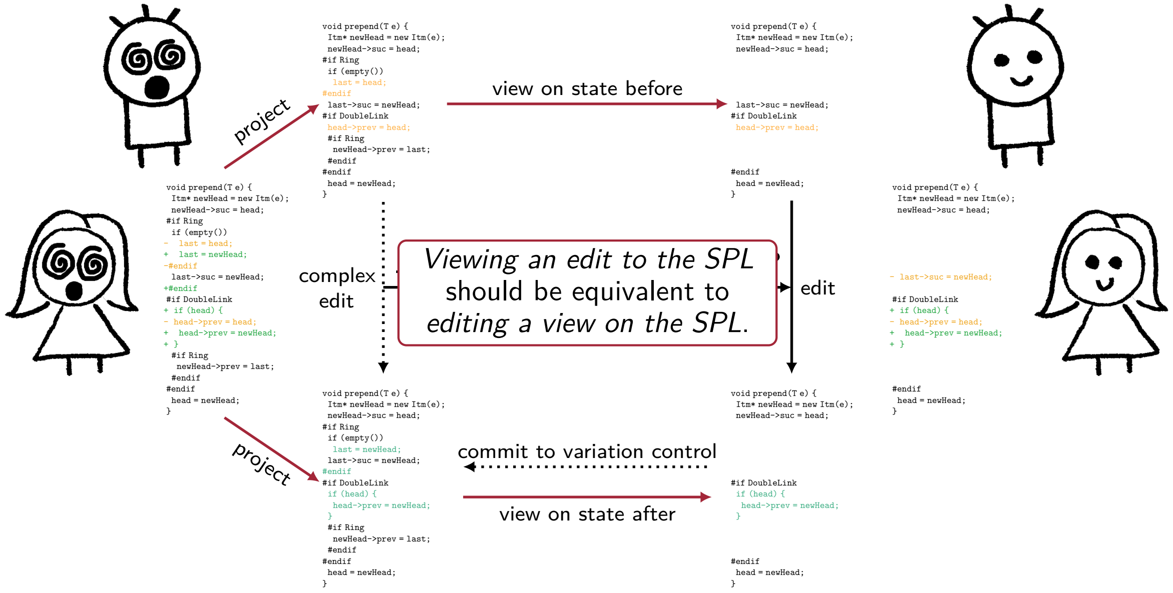
```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;


 last->suc = newHead;
#if DoubleLink
 head->prev = head;


#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;


- last->suc = newHead;

#if DoubleLink
+ if (head) {
-  head->prev = head;
+   head->prev = newHead;
+ }


#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
 last->suc = newHead;
#endif
#if DoubleLink
 if (head) {
  head->prev = newHead;
 }
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;



#if DoubleLink
 if (head) {
  head->prev = newHead;
 }


#endif
 head = newHead;
}
```

Viewing an edit to the SPL
should be equivalent to
editing a view on the SPL.



```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
-  last = head;
+  last = newHead;
-#endif
 last->suc = newHead;
+#endif
#if DoubleLink
+ if (head) {
-  head->prev = head;
+   head->prev = newHead;
+ }
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = head;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

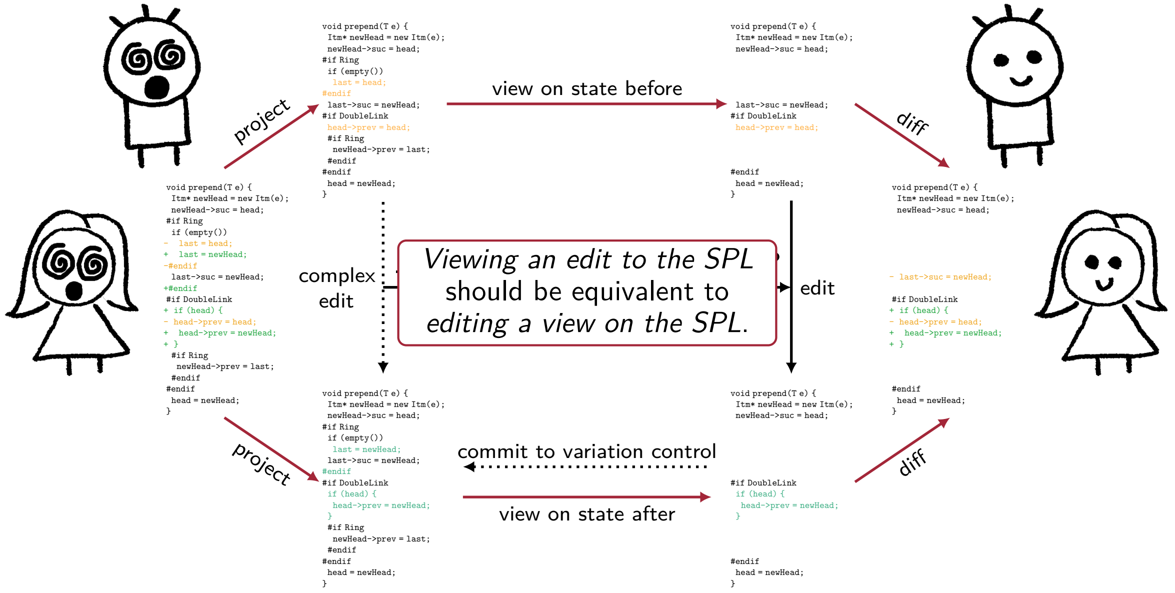
```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;



 last->suc = newHead;
#if DoubleLink
 head->prev = head;




#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;




- last->suc = newHead;

#if DoubleLink
+ if (head) {
- head->prev = head;
+  head->prev = newHead;
+ }




#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
 last->suc = newHead;
#endif
#if DoubleLink
 if (head) {
  head->prev = newHead;
 }
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;





#if DoubleLink
 if (head) {
  head->prev = newHead;
 }




#endif
 head = newHead;
}
```

Viewing an edit to the SPL should be equivalent to *editing a view on the SPL.*



```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
-  last = head;
+  last = newHead;
-#endif
 last->suc = newHead;
+#endif
#if DoubleLink
+ if (head) {
-  head->prev = head;
+   head->prev = newHead;
+ }
  #if Ring
   newHead->prev = last;
  #endif
#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = head;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

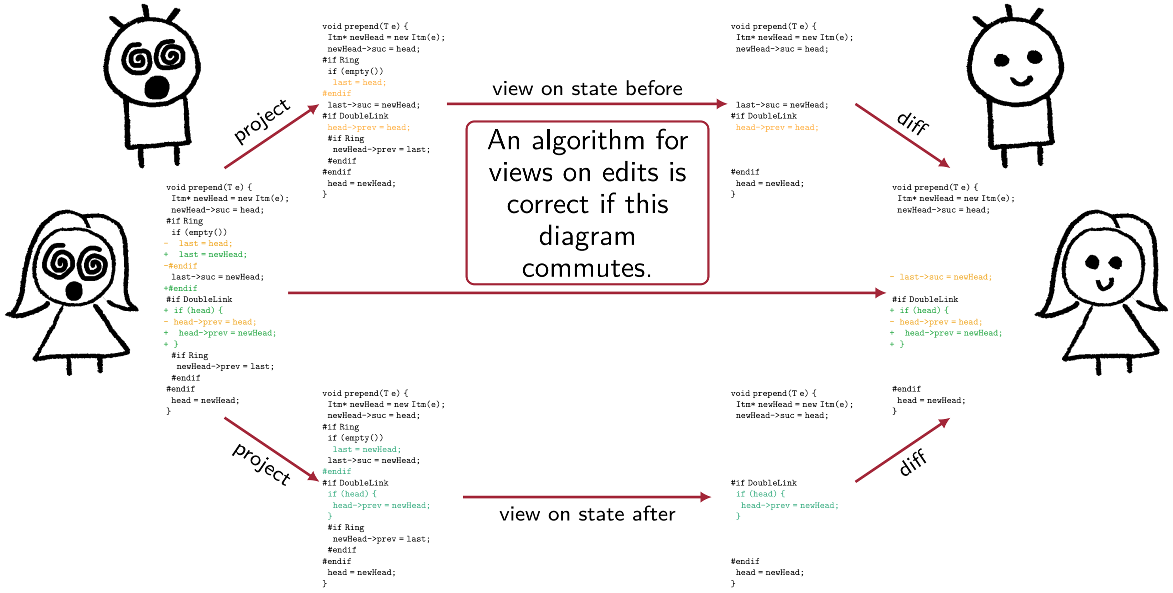
```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;




 last->suc = newHead;
#if DoubleLink
 head->prev = head;




#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;

- last->suc = newHead;

#if DoubleLink
+ if (head) {
- head->prev = head;
+   head->prev = newHead;
+ }



#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
 last->suc = newHead;
#endif
#if DoubleLink
 if (head) {
  head->prev = newHead;
 }
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;





#if DoubleLink
 if (head) {
  head->prev = newHead;
 }



#endif
 head = newHead;
}
```

Viewing an edit to the SPL should be equivalent to *editing a view on the SPL.*



```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
-  last = head;
+  last = newHead;
-#endif
 last->suc = newHead;
+#endif
#if DoubleLink
+ if (head) {
-  head->prev = head;
+   head->prev = newHead;
+ }
  #if Ring
   newHead->prev = last;
  #endif
#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = head;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

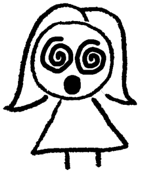
```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;



 last->suc = newHead;
#if DoubleLink
 head->prev = head;



#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;

- last->suc = newHead;

#if DoubleLink
+ if (head) {
- head->prev = head;
+  head->prev = newHead;
+ }

#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
 last->suc = newHead;
#endif
#if DoubleLink
 if (head) {
  head->prev = newHead;
 }
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;



#if DoubleLink
 if (head) {
  head->prev = newHead;
 }


#endif
 head = newHead;
}
```

An algorithm for views on edits is correct if this diagram commutes.



```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
-  last = head;
+  last = newHead;
-#endif
 last->suc = newHead;
+#endif
#if DoubleLink
+ if (head) {
-  head->prev = head;
+   head->prev = newHead;
+ }
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

project

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = head;
#endif
 last->suc = newHead;
#if DoubleLink
 head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

view on state before

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;


 last->suc = newHead;
#if DoubleLink
 head->prev = head;



#endif
 head = newHead;
}
```

diff

An algorithm for views on edits is correct if this diagram commutes.

We propose two algorithms

naive but elegant

runtime optimized

project

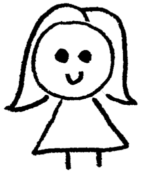
```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
  last = newHead;
 last->suc = newHead;
#endif
#if DoubleLink
 if (head) {
  head->prev = newHead;
 }
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

view on state after

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;



#if DoubleLink
 if (head) {
  head->prev = newHead;
 }


#endif
 head = newHead;
}
```

diff

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;

- last->suc = newHead;

#if DoubleLink
+ if (head) {
-  head->prev = head;
+   head->prev = newHead;
+ }

#endif
 head = newHead;
}
```



Feasibility Study – How efficiently can views be computed?

44 open-source
SPL histories
(incl. 🐧)

Feasibility Study – How efficiently can views be computed?

44 open-source
SPL histories
(incl.)

1.7 million commits
5 million patches

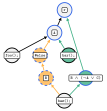
Feasibility Study – How efficiently can views be computed?

44 open-source SPL histories (incl. 🐧)

git

1.7 million commits
5 million patches

parse all patches



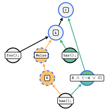
Feasibility Study – How efficiently can views be computed?

44 open-source SPL histories (incl.)

git

1.7 million commits
5 million patches

parse all patches



generate one predicate for

- Partial Conf.
- Feature Trace
- Code Search

Artifacts Available acm V1.1

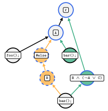
Artifacts Evaluated acm Functional V1.1

Feasibility Study – How efficiently can views be computed?

44 open-source SPL histories (incl.)

1.7 million commits
5 million patches

parse all patches



generate one predicate for

- Partial Conf.
- Feature Trace
- Code Search

measure runtime of both algorithms

naive but elegant

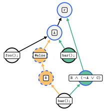
runtime optimized

Feasibility Study – How efficiently can views be computed?

44 open-source SPL histories (incl.)

1.7 million commits
5 million patches

parse all patches



generate one predicate for

- Partial Conf.
- Feature Trace
- Code Search

measure runtime of both algorithms

naive but elegant

runtime optimized

99.9% of views can be generated in $\leq 1s$ (median 1ms).
is $\geq$37x faster for instances where takes $\geq 1s$.

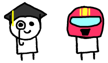
Summary

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
-   last = head;
+   last = newHead;
-#endif
 last->suc = newHead;
+#endif
#if DoubleLink
+ if (head) {
- head->prev = head;
+   head->prev = newHead;
+ }
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
-   last = head;
-#endif
 last->suc = newHead;
#if DoubleLink
- head->prev = head;
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

view on state before

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;


 last->suc = newHead;
#if DoubleLink
- head->prev = head;


#endif
 head = newHead;
}
```



```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;
#if Ring
 if (empty())
+   last = newHead;
 last->suc = newHead;
+#endif
#if DoubleLink
+ if (head) {
+   head->prev = newHead;
+ }
 #if Ring
  newHead->prev = last;
 #endif
#endif
 head = newHead;
}
```

view on state after

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;



#if DoubleLink
+ if (head) {
+   head->prev = newHead;
+ }


#endif
 head = newHead;
}
```

```
void prepend(T e) {
 Itm* newHead = new Itm(e);
 newHead->suc = head;


- last->suc = newHead;

#if DoubleLink
+ if (head) {
- head->prev = head;
+   head->prev = newHead;
+ }


#endif
 head = newHead;
}
```

Kästner, C. (2010).
Virtual Separation of Concerns: Toward Preprocessors 2.0.
PhD thesis, University of Magdeburg.

Walkingshaw, E. and Ostermann, K. (2014).
Projectional Editing of Variational Software.
In *Proc. Int'l Conf. on Generative Programming: Concepts & Experiences (GPCE)*, pages 29–38. ACM.