a roadmap to revolutionize spl technology by 2025

don batory
department of computer science
university of texas at austin

in 2007

- I began studying the relationship of VCS and SPLs
- Lots of possibilities: propagating changes from one clone-and-own to another

- My 1st thought was to make a VCS as the engine to produce products of an SPL
  - a VCS has the entire SPL codebase to build a product, why not?
    Interesting variation on DOP
  - is clone-and-own a good idea??
    need could be reduced by a variability-aware VCS

interesting research prospects

back then as now...

- I was coding a lot to build an MDE tool suite in Java
  - a large part of my task was constantly refactoring my codebase
  - recalled desperately wanting to refactor AHEAD and other SPLs log ago, and knew an entirely new generation of SPL tools – (**not based on preprocessors**) was needed

- That's when it dawned on me that future SPL technologies required a pair of fundamental advances. Namely, integration with:

version control

both!

refactorings

why??

- Need I remind you of the today's state of SPL tools
  - preprocessor based – last century technology
  - you can't refactor without type information
  - industrial SPL technology is about 10-15 years behind Research

- This is why SPLs are not "standard" (SPLs are still an oddity)
  - using CPP and all of its bad practices
  - little or no IDE support for SPLs
  - no refactoring (except in last 4 years)
  - with nothing-special VCSs

This is the same environment I used in 1987 to build Genesis - maybe 1st academic SPL

lesson #1: refactorings are not edits!

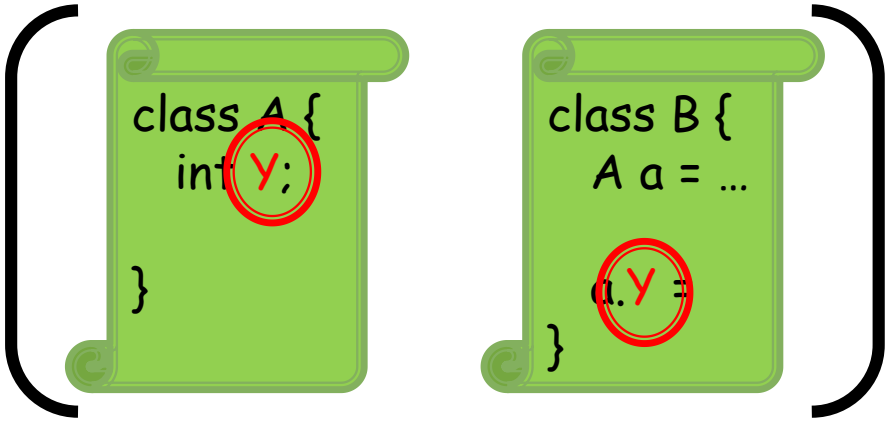
Danny Dig's PhD Thesis, Nov 2007

Batory keynote at ETAPS April 2007

refactorings are a problem for vcs

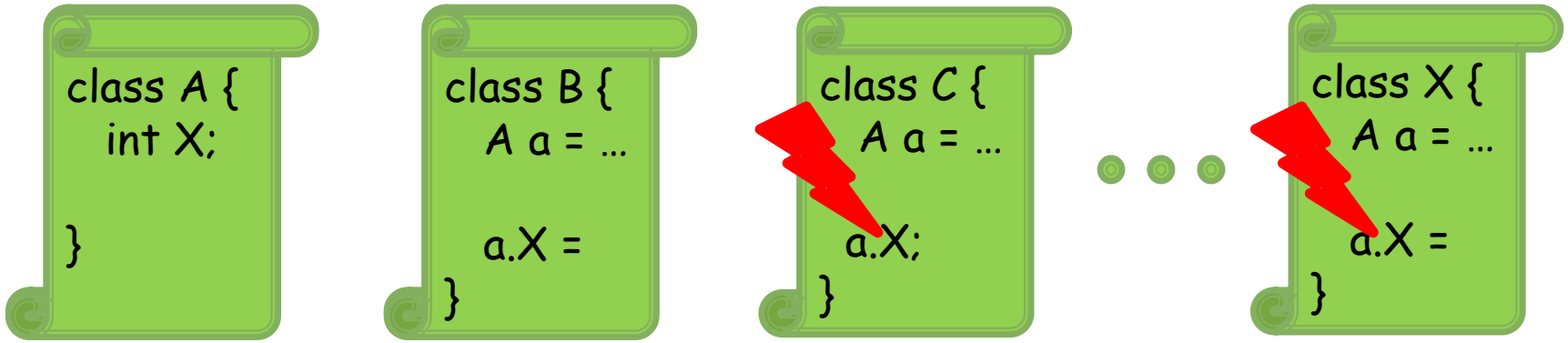
- Check out a module from a project and refactor

rename
X→Y



consistency of repository is broken not all references to X were renamed

repository



a partial solution

- Check-out the entire repository, refactor, and check-in. That works…
- But really it doesn't!

rename
X→Y

```
class A
{
   int Y;
}
```

```
class B {
   A a =
   a.Y =
}
```

repository

```
class A
{
   int X;
}
```

```
class B {
   A a =
   a.X =
}
```

```
class C {
   A a = …
   a.X;
}
```

```
class X {
   A a = …
   a.X;
}
```

danny dig's 2007 solution

- Make VCS refactoring aware!
- I found a way to explain one of his results using a preliminary version of a feature-algebra
- Danny liked my explanation and used it in his thesis
- See Dig Thesis (2 chapters) or my ETAPS 2007 keynote

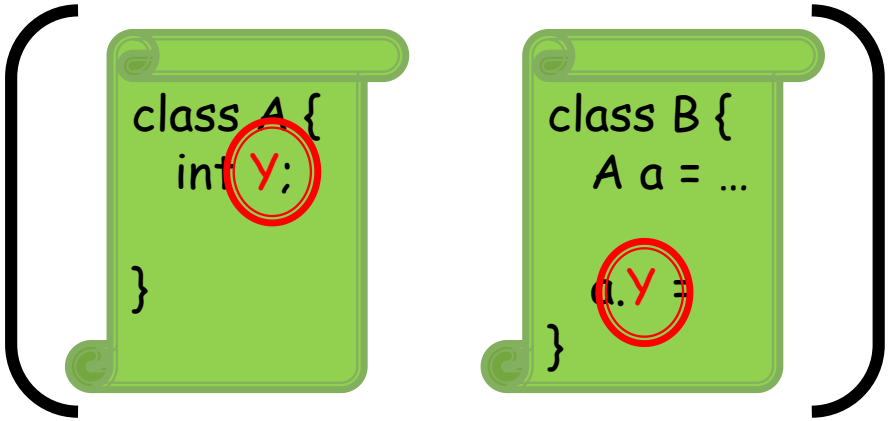
lesson #2: refactorings are not edits!

Kim, Batory, Dig SPLC 2017

refactorings are a problem for spls

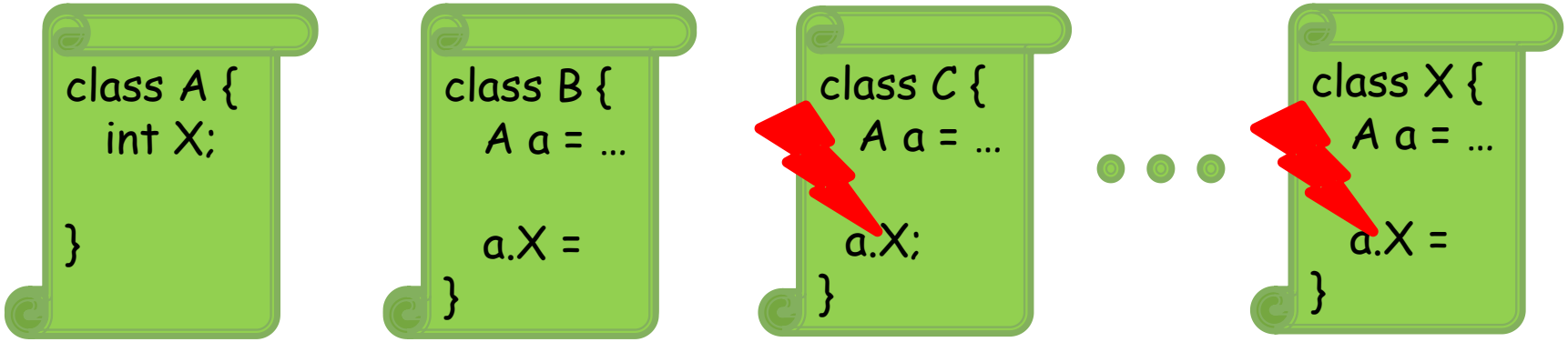
- Build a product of a SPL, refactor it, and back-propagate edits

rename X→Y



consistency of spl codebase is broken not all references to X were renamed

150% spl code base



my 2017 solution

Postdates
Apel's Morpheus 2015

- Make SPL tools refactoring aware!

X15 paper SPLC 2017

- An spl product is a view of a 150% SPL codebase
- Code folding hides irrelevant parts of an SPL codebase
- User can edit SPL program (view) and refactor
    - behind curtains, refactoring is applied to entire code base
    - code folding makes it look as if only the SPL program was refactored
- Correctness: verified by feature algebras extended with distributivity axiom about refactorings

x15... briefly...

- Does not repeat the mistakes of last century – using CPP to encode variability
- Reinterpret Java annotations

```
@interface Feature {
  static final boolean X = true;
  static final boolean Y = false;
  static final boolean Z = false;
  boolean value();
}
```

configuration file as a
Custom Java annotation

annotate classes, fields, methods

```
interface Graphics {}

@Feature(X)
class Square implements Graphics {}
```

```
@Feature(X)
int i, j;
```

for the given configuration

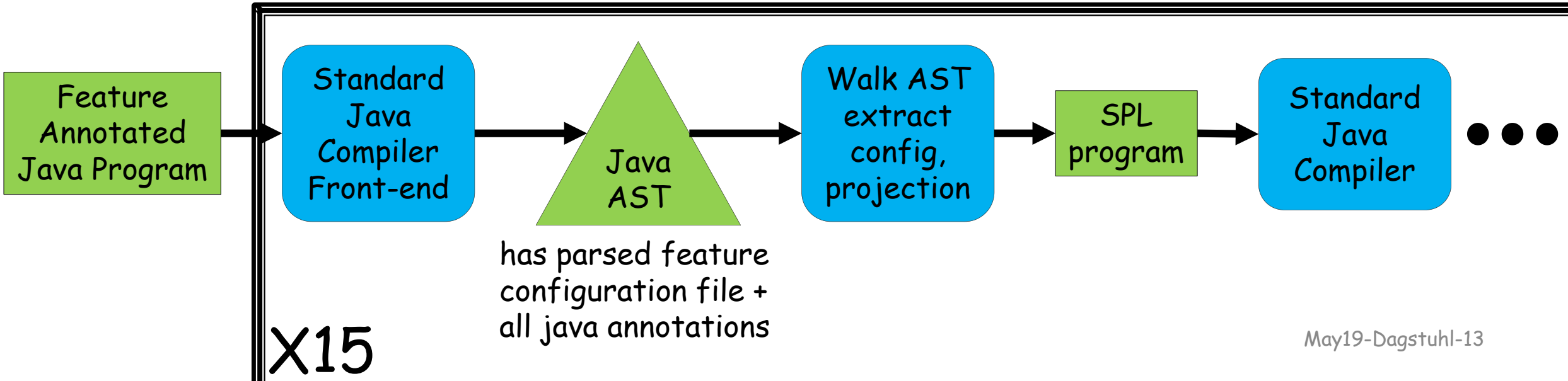
significant benefit

- State of art: build a variability-aware compiler
  - host language's grammar (Java) and integrate CPP constructs
  - build compiler, type checker, etc.

Variability-Aware Parsing in the Presence of Lexical Macros and Conditional Compilation

Christian Kästner Paolo G. Giarrusso Tillmann Rendel Thorsten Berger
Sebastian Erdweg Klaus Ostermann
Philipps University Marburg, Germany
University of Leipzig, Germany

- X15 use Plain Old Java compiler



key: x15 does *not* back-prop ts!

- Treating refactorings as slides

lesson #3:

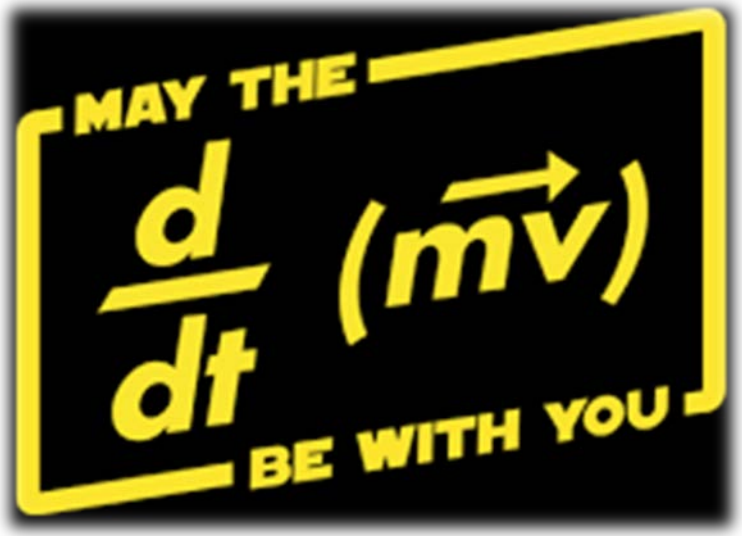
refactorings are not edits!

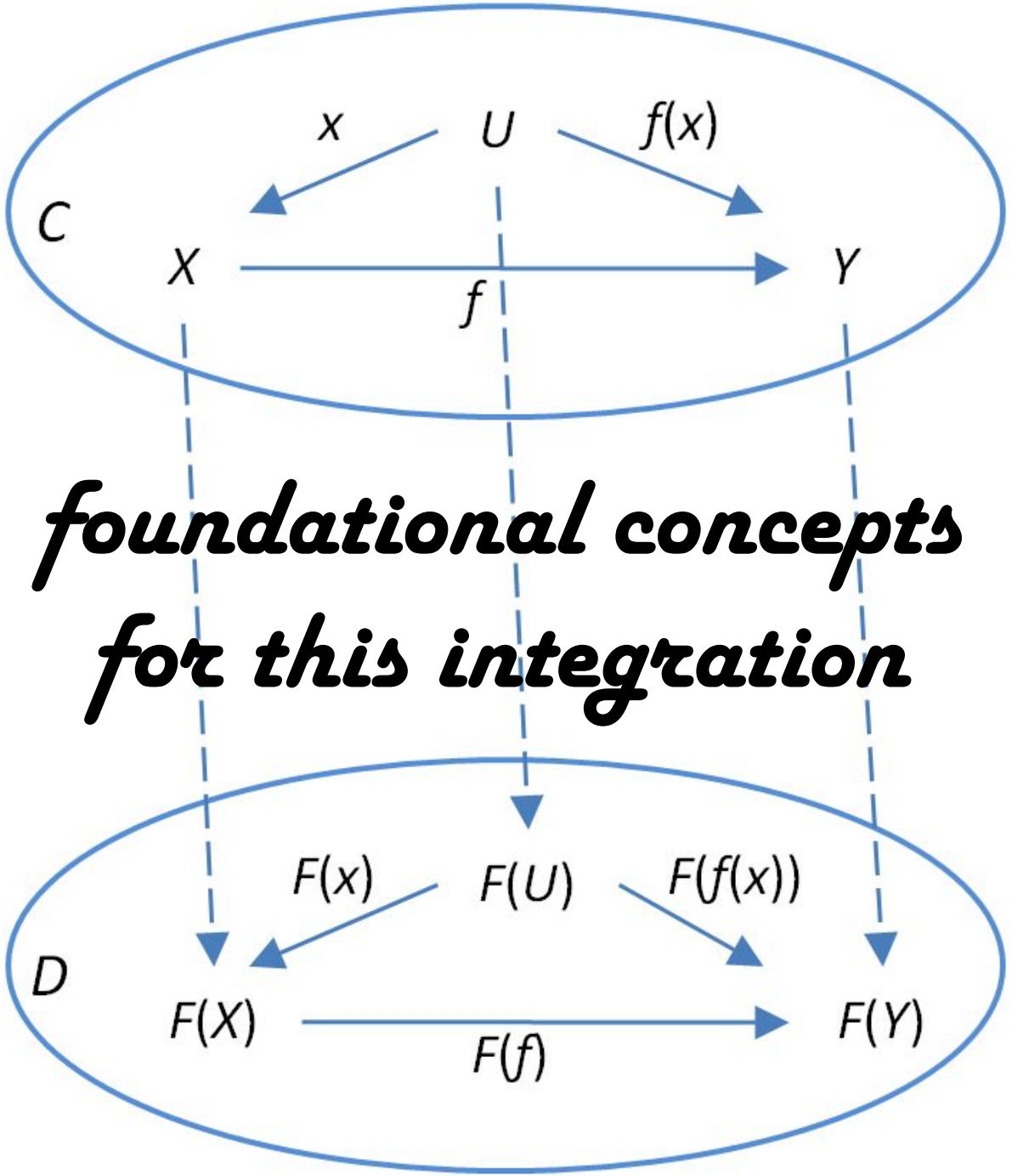
avoid back-propagation of edits!

random thoughts

- Why not extend Eclipse refactoring engine (ERE)?
  - ERE is broken beyond repair (ICSE 2016)
  - Roach infested
  - Slow – 300 refactorings took 5 minutes to execute
  - X15 – 4 seconds for 300 refactorings (10x-100x faster)

- May you have better luck with version control systems





foundational concepts for this integration

to explain
Dig's results
and show
the theory
underlying
VCS and SPLs
and refactoring

basic category theory

- Raise
- Raise

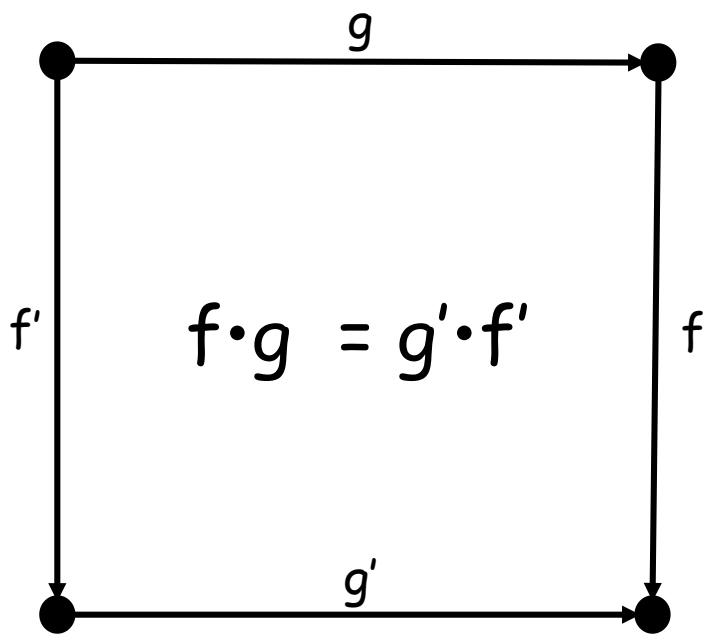
to hopeless ☹

hope! ☺

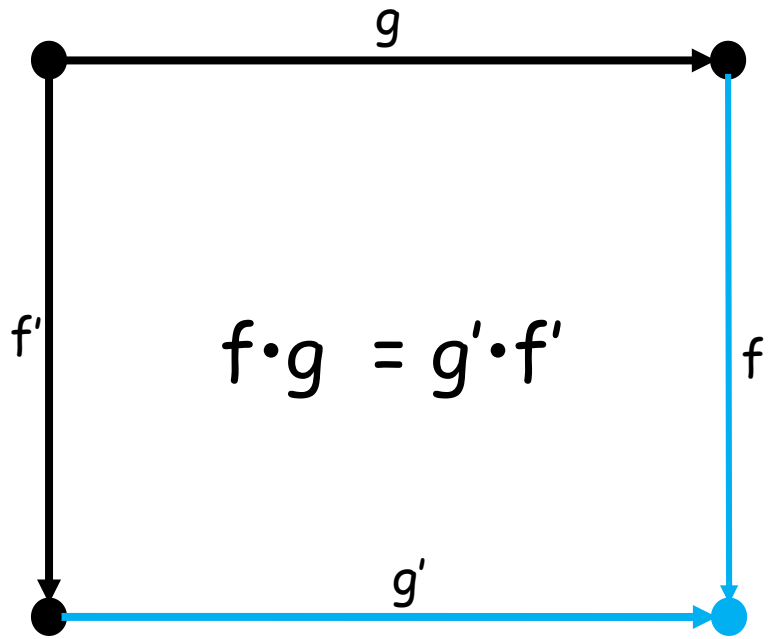
basic category theory

- Unifies VCS, refactorings, and SPLs
- Why? All are functional paradigms
- CT is theory of functions

- "completing a commuting diagram"



commuting diagram

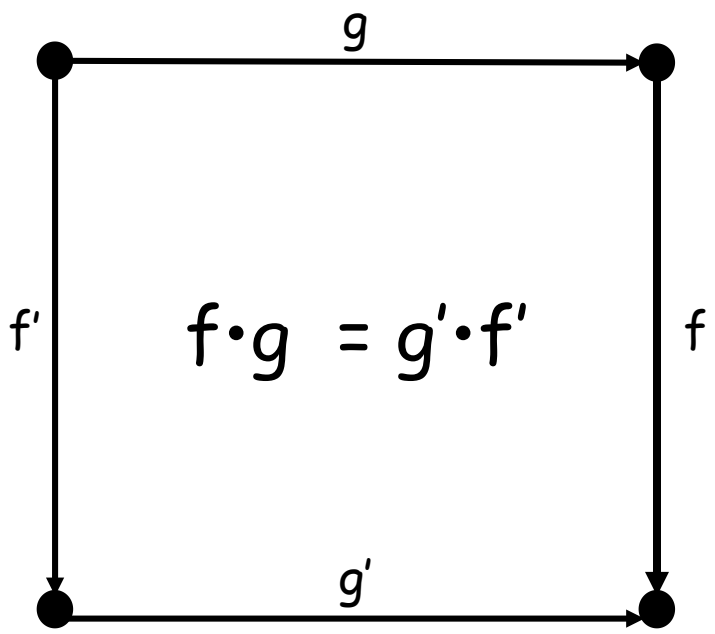


pushout

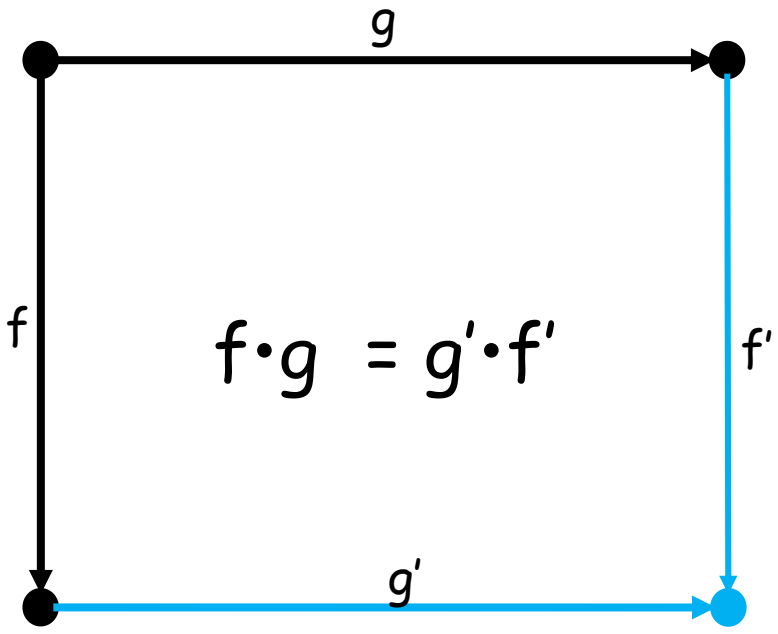
foundational concepts

- Unifies VCS, refactorings, and SPLs
- Why? All are functional paradigms
- CT is theory of functions

- "completing a commuting diagram"



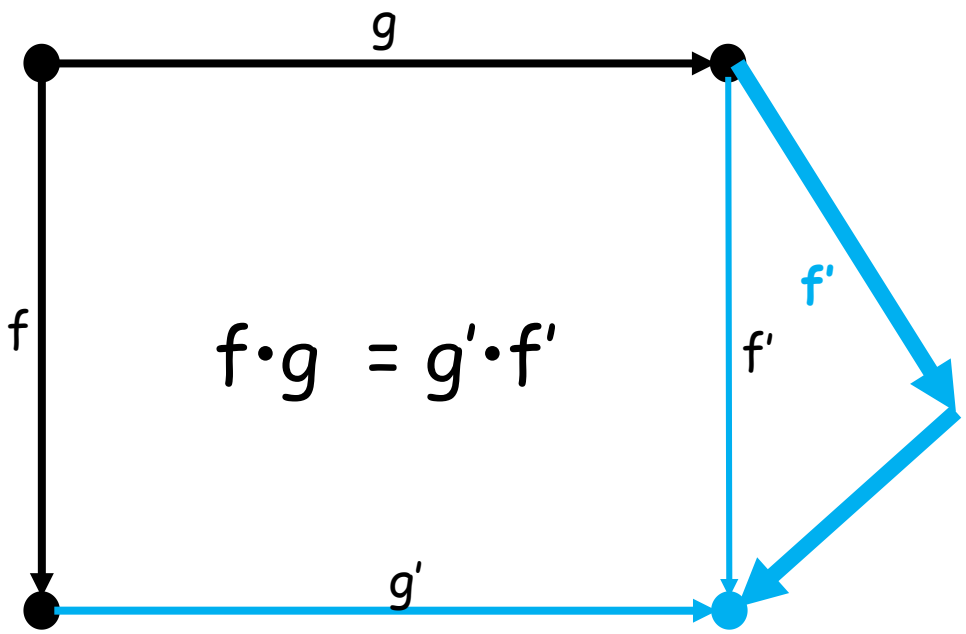
commuting diagram



pushout

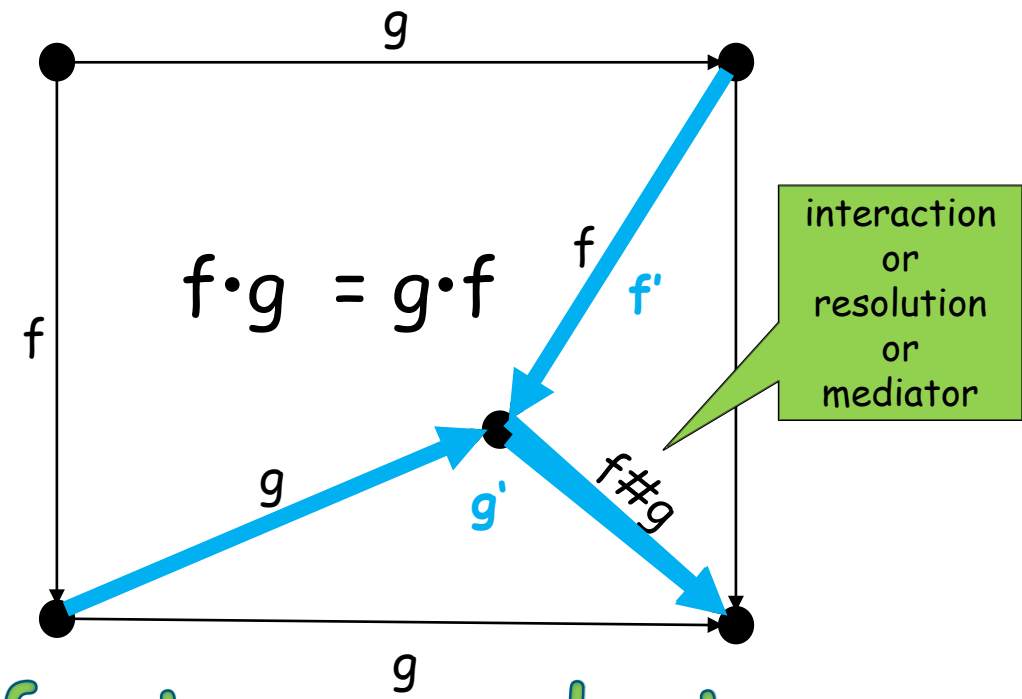
foundational concepts

- "completing a commuting diagram"



pushout

- Fundamental law in feature algebras on features and feature interactions

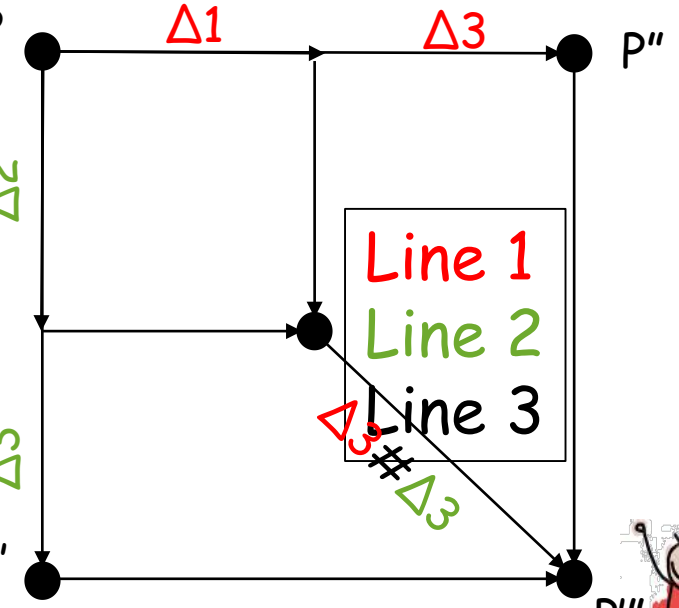


feature product

foundational concepts: vcs merging edits

standard version control ideas

```
Line 1
Line 2
>>>>
Line 3
<<<<
Line 3
>>>>
```

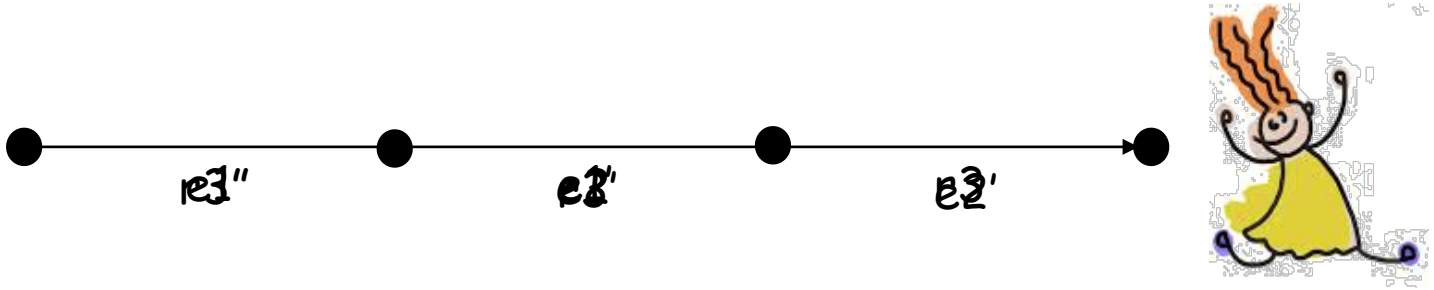


Line 1
Line 2
Line 3

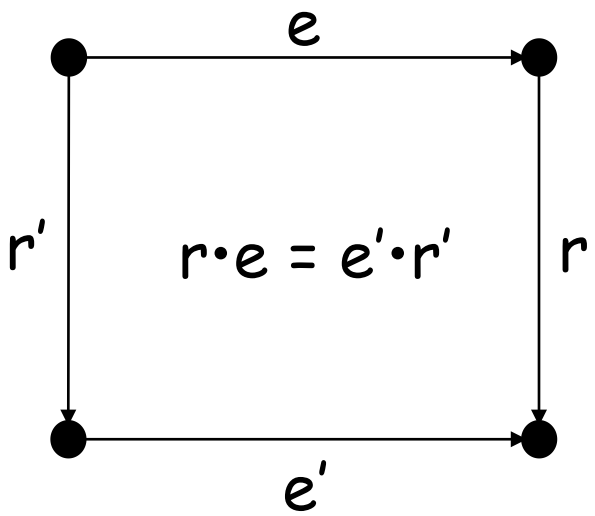
Line 1
Line 2
Line 3

Line 1
Line 2
Line 3

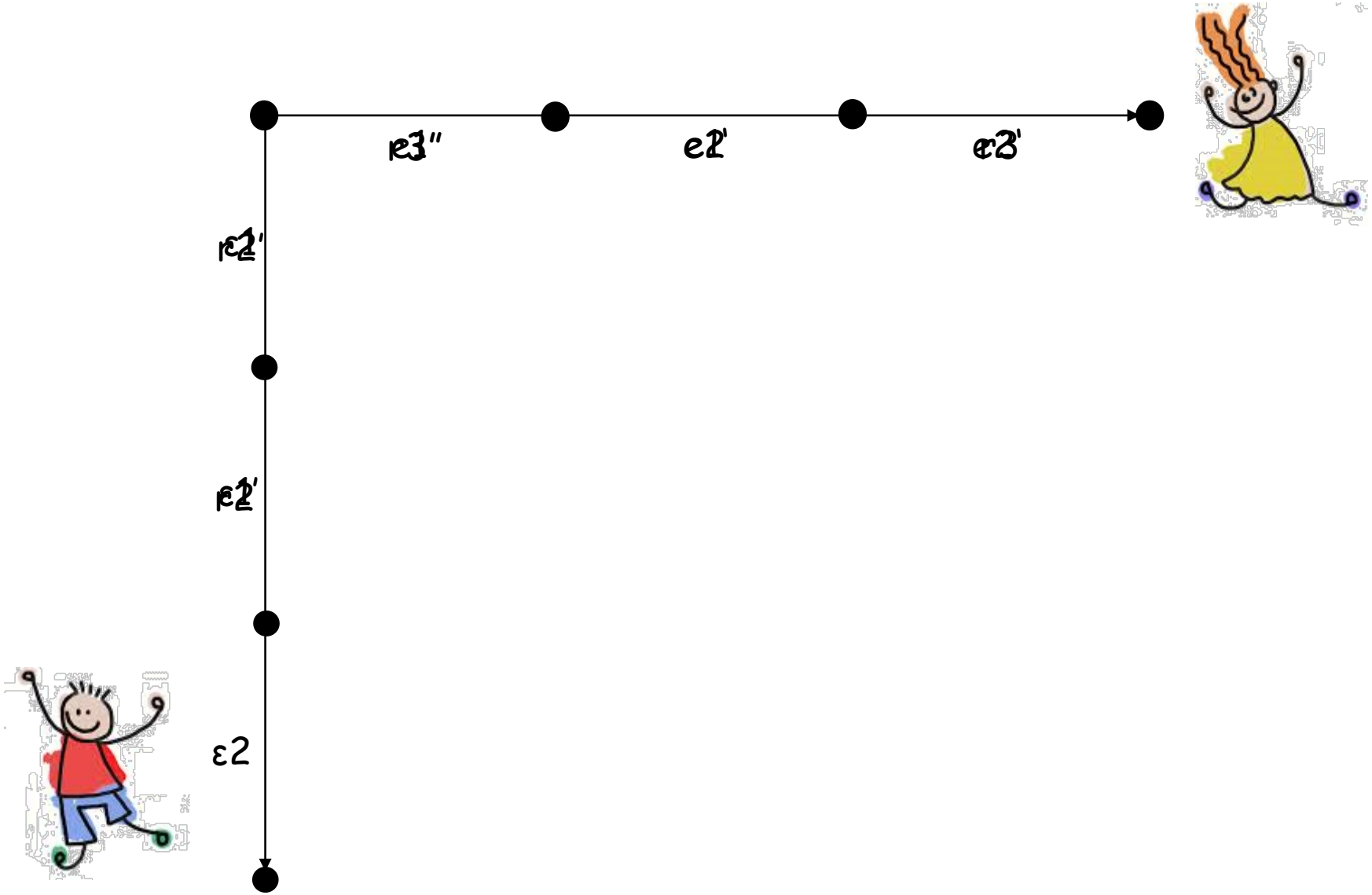
dig's approach to a refactoring-aware vcs



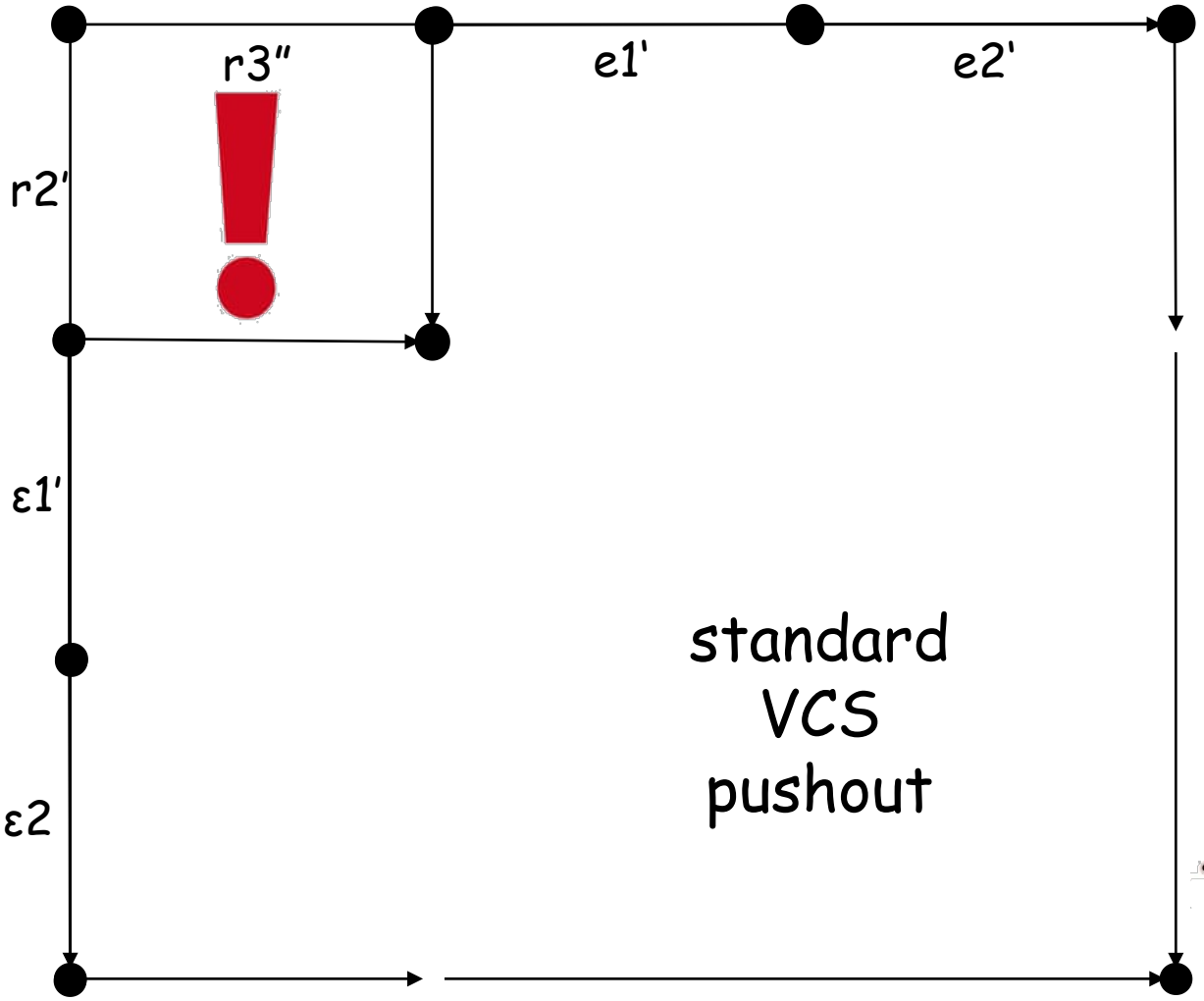
change order of edits
and refactorings without
altering semantics!



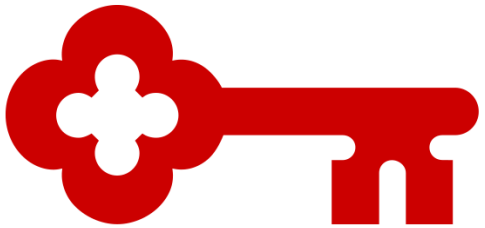
dig's approach to a refactoring-aware vcs



dig's approach to a refactoring-aware vcs



to my knowledge
Danny never finished
this part of his work
more of a conjecture



refactoring-aware VCS

recap... so what?

it has been abundantly clear to me

since the beginning of this century

- Today's IDE support for SPLs is pfft…

- Imagine that we created a 21st century programming environment (IDE)
  - true refactoring support for product lines and their programs
  - true refactoring support for version control

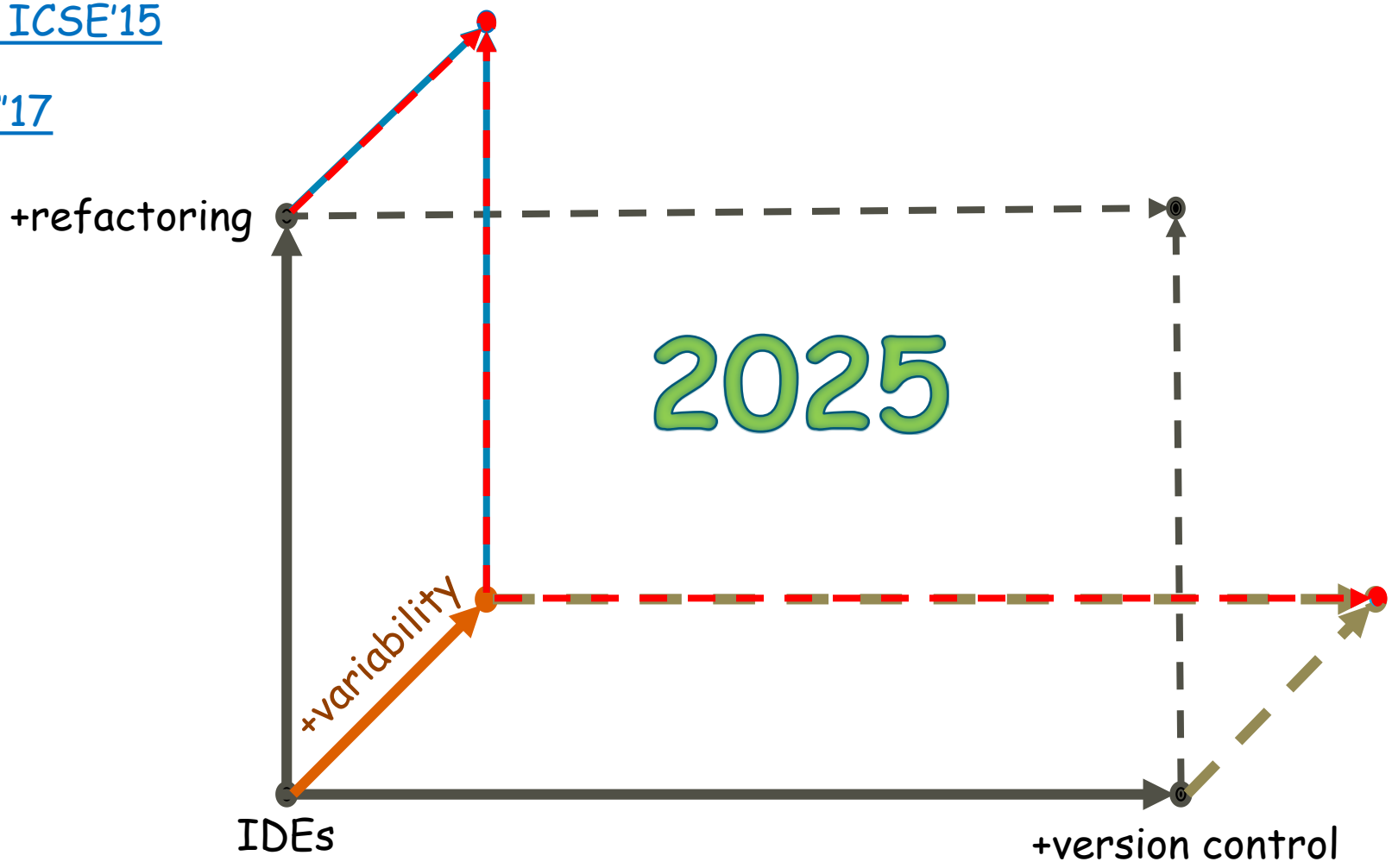
- Building product-lines would be standard
  - one-of-a-kind designs would be the oddity
  - teaching SPLs would be standard fare in undergraduate curriculums
  - variability would not be an afterthought to SE

a history and future of spl technologies

last slide

Morpheus ICSE'15

X15 SPLC'17



Variability is A fundamental part of software engineering and software design

Dig's thesis '07

This Dagstuhl

a history and future of spl technologies

last slide

Mo

X1

Variability is
damental
f software
eering and
vare design

hesis '07

Dagstuhl

P

+version control