a roadmap to revolutionize spl technology by 2025

don batory
department of computer science
university of texas at austin

May19-Dagstuhl-1
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!


Batory keynote at ETAPS April 2007
refactorings are a problem for vcs

- Check out a module from a project and refactor

```java
class A {
    int X;
}

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

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

rename $X \rightarrow Y$

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

```java
class A {
    int Y;
}

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

```java
class C {
    A a = ...
    a.X =
}
```

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

May19-Dagstuhl-6
a partial solution

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

rename $X \rightarrow Y$

May 19 - Dagstuhl 7
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

```java
class A {
    int X;
}

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

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

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

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

rename X → Y

150% SPL code base
my 2017 solution

• 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 to annotate classes, fields, methods

configuration file as a Custom Java annotation

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

```java
interface Graphics {}
@Feature(X)
class Square implements Graphics {
    @Feature(X)
    int i, j;
}
```
significant benefit

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

- X15 use Plain Old Java compiler
key: x15 does not back-propagate edits!

Treating refactorings as edits yields the problems of the previous slides.

refactorings are a problem for SPLs.

Check out a program from a SPL and refactor:

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

rename X → Y:

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

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
foundation concepts for this integration
to explain Dig’s results and show the theory underlying VCS and SPLs and refactoring
basic category theory

YOU HAVE CHOSEN... WISELY

• Raise your hand if you know basic CT!
• Raise your hand if you want to know a little CT?

This way to hopeless 😞
This way to hope! 😊
basic category theory

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

• “completing a commuting diagram”

\[ f \circ g = g' \circ f' \]

\[ f \circ g = g' \circ f' \]
foundational concepts

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

```
f • g = g' • f'
```

“completing a commuting diagram”

```
f • g = g' • f'
```

commuting diagram

pushout
foundational concepts

• “completing a commuting diagram”
• Fundamental law in feature algebras on features and feature interactions

\[ f \cdot g = g' \cdot f' \]

pushout

\[ f \cdot g = g \cdot f \]

feature product
foundational concepts: vcs merging edits

standard version control ideas
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

standard VCS pushout

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

refactoring-aware VCS

May19-Dagstuhl-26
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
Variability is a fundamental part of software engineering and software design.

Morpheus ICSE'15
X15 SPLC'17

2025

IDEs
+refactoring
+variability
+version control

Dig's thesis '07
This Dagstuhl
Variability is fundamental part of software engineering and software design.