# Genetic Improvement for Software Product Lines: An Overview and a Roadmap

Roberto E. Lopez-Herrejon[1], Lukas Linsbauer[1], **Wesley K. G. Assunção [2,3]**, Stefan Fischer[1], Silvia R. Vergilio[2], Alexander Egyed[1]

[1] ISSE - Johannes Kepler University – Austria
[2] DINF - Federal University of Paraná – Brazil
[3] COINF - Technological Federal University of Paraná – Brazil

{roberto.lopez, lukas.linsbauer,stefan.fischer,alexander.egyed}@jku.at
{wesleyk, silvia}@inf.ufpr.br

# Software Product Lines Background

- Software Product Lines (SPLs)
  - Families of software systems that provide different combinations of features

- Some proven advantages of SPL practices:
  - Improved reuse of existing software artefacts
  - Improved quality
  - Faster time to market

# SPLs and Genetic Improvement – a la GISMOE

- Genetic Improvement (GI)
  - Improves systems behaviour using genetic programming
  - Starts from existing programs that are evolved for a given criteria
- Genetic Improvement of Software for Multiple Objective Exploration (GISMOE)

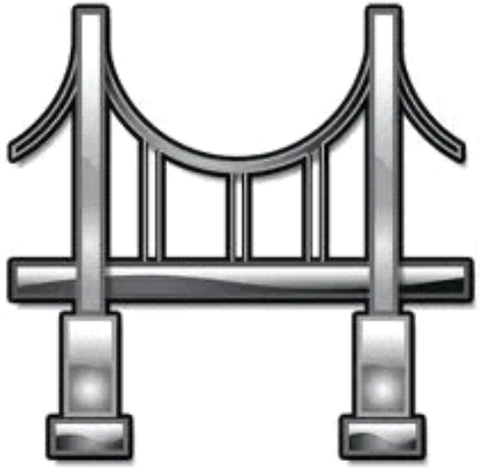
*The GISMOE approach may also offer solutions to some of the issues raised by SPLs. For example, using GISMOE, we can create new branches automatically: the GP engine will evolve the new versions of the product family from existing members of the family. We may also be able to merge versions when the product family becomes large or unwieldy.*
*ASE 2012 keynote*

# The Main Goal of our Paper

ECCO approach

SPL problems

- Reverse Engineering
- Evolution



GISMOE approach

- Genetic Improvement
  - core ideas

- identify synergies
- spark interest
- establish roadmap

# Reverse Engineering SPLs – Motivation

- Prevailing scenario in industry
  - Existence of multiple similar products developed mostly independently
  - Number of products and their complexity prevents adequate maintenance and evolution of each individual product

**Software Product Lines can offer a solution**

# Reverse Engineering SPLs – Challenge

## Current Scenario

Product P1

Requirements | Design | Implementation

Product P2

Requirements | Design | Implementation

. . .

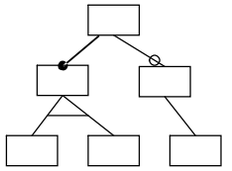
Product Pn

Requirements | Design | Implementation

?

## SPL

Feature Model



Artifacts + Variability

Requirements

Design

Implementation

# ECCO Overview [ICSME14, ICSE15]

- Extraction and Composition for Clone-and-Own (ECCO)
  - Approach for reverse engineering SPLs
  - Incrementally traces features and feature interactions to the artefacts that implement software products
  - Interactively provides software engineers feedback for evolving SPLs
  - Works under two premises:
    - The list of features implemented in each software product is available
    - The artefacts of each software product are available

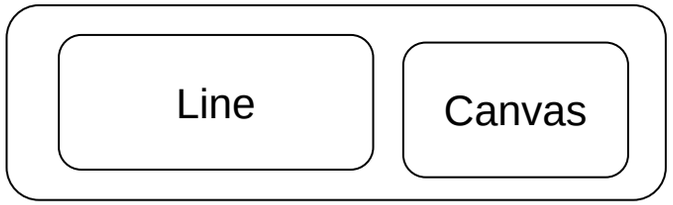
# ECCO Basic Ideas

- Base module
  - Implements a feature regardless of the presence or absence of other features in a product.

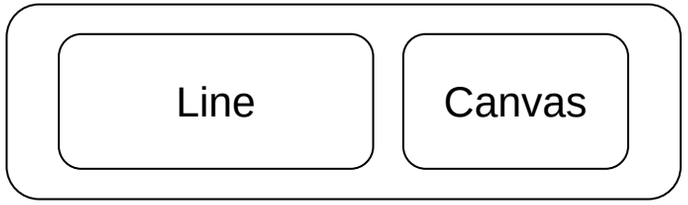
- Derivative module
  - Represents the interaction between features at the structural level $m=\delta^n(c_o,c_1,...,c_n)$ where $c_i$ are selected features F or not selected features ¬F (a.k.a. negative features).

# ECCO Running Example – Trace Extraction

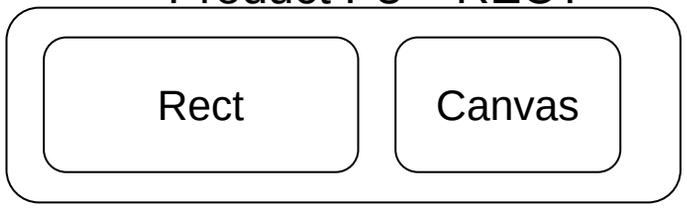
Product P1 – LINE




Product P2 – LINE, WIPE




Product P3 – RECT




- ECCO's traceability extraction algorithm
  - identifies artefacts that implement features and feature interactions
  - uses structural diffing
  - considers hierarchy and all levels of granularity

- For our example, we need to identify
  - line,wipe, rect, $\delta^1$(line,wipe), ...

# ECCO Traceability Example (1)

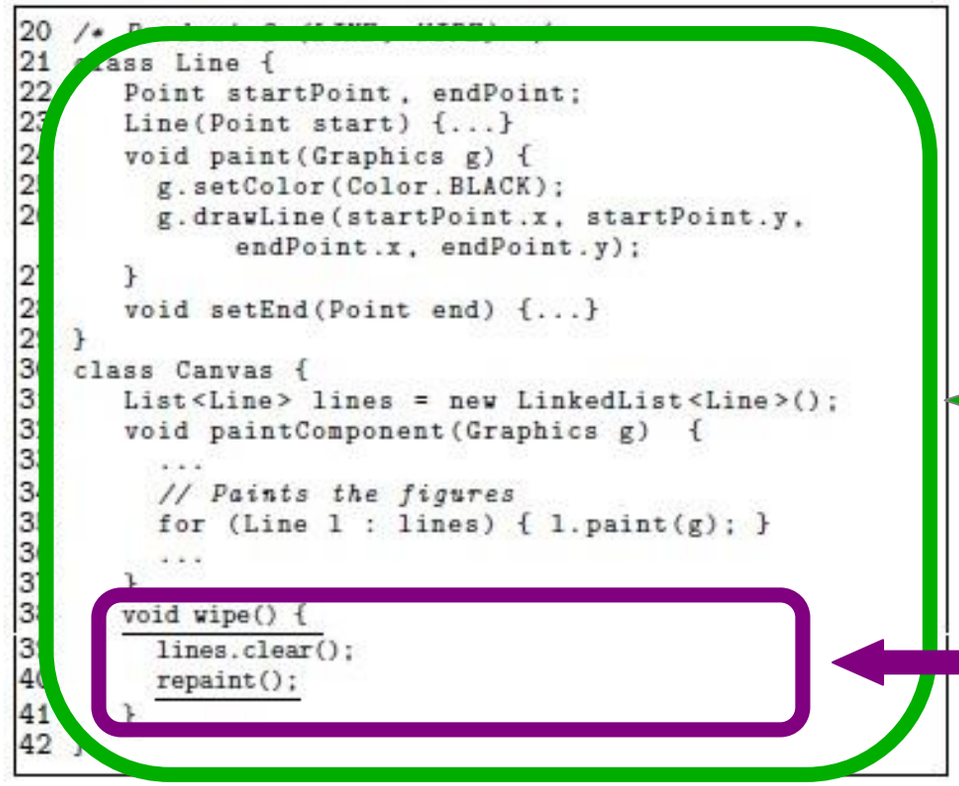
```
1  // Product 1 (LINE)
2  class Line {
3      Point startPoint, endPoint;
4      Line(Point start) {...}
5      void paint(Graphics g) {
6        g.setColor(Color.BLACK);
7        g.drawLine(startPoint.x, startPoint.y,
               endPoint.x, endPoint.y);
8      }
9      void setEnd(Point end) {...}
10 }
11 class Canvas {
12     List<Line> lines = new LinkedList<Line>();
13     void paintComponent(Graphics g)  {
14        ...
15        // Paints the figures
16        for (Line l : lines) { l.paint(g); }
17        ...
18     }
19 }
```

line

Product P1
With feature LINE

base
module

# ECCO Traceability Example (2)



```
20 /*
21 class Line {
22    Point startPoint, endPoint;
23    Line(Point start) {...}
24    void paint(Graphics g) {
25      g.setColor(Color.BLACK);
26      g.drawLine(startPoint.x, startPoint.y,
27           endPoint.x, endPoint.y);
28    }
29    void setEnd(Point end) {...}
30 }
31 class Canvas {
32    List<Line> lines = new LinkedList<Line>();
33    void paintComponent(Graphics g)  {
34      ...
35      // Paints the figures
36      for (Line l : lines) { l.paint(g); }
37      ...
38    }
39    void wipe() {
40      lines.clear();
41      repaint();
42    }
```

line

wipe

$\delta^1$(line,wipe)

Product P2
With feature LINE and WIPE

base and
derivative modules

# ECCO Traceability Example (3)

```
43  /* Product 3 (RECT) */
44  class Rect {
45      int x, y, width, height;
46      Rect(int x, int y) {...}
47      void paint(Graphics g) {
48        g.setColor(Color.BLACK);
49        g.drawRect(x, y, width, height);
50      }
51      void setEnd(int newX, int newY) {...}
52  }
53  class Canvas {
54      List<Rect> rects = new LinkedList<Rect>();
55      void paintComponent(Graphics g)  {
56        ...
57        // Paints the figures
58        for (Rect r : rects) { r.paint(g); }
59        ...
60      }
61  }
```

rect

Product P3
With feature RECT

base
module

# ECCO meets GISMOE - Overview

- Drawing connections
  - Types of sensitivity analysis provided and needed
  - Test case generation
  - The need of co-evolution
  - Human-in-the loop

- Open challenges, interesting questions, and some wild speculations ...

# 1. Feature-Level Sensitivity Analysis

- ECCO can provide traces to features and feature interactions – *new form of sensitivity analysis*
- Potential benefits for GISMOE
  - Focus better where to target the evolution of artefacts in a SPL context to:
    - Repair a bug in a feature
    - Graft new functionality (feature)
    - Evolve a property of a feature
- ECCO traceability currently focuses on structural interactions
  - Clone detection technologies, control and data flow

# 2. Automated Test Case Generation

- Most common form of SPL testing is *Combinatorial Interaction Testing (CIT)*
  - Current research focuses on selection of products from variability models (i.e. feature models).
  - Open challenge is the automated test case generation for the selected products.
- ECCO
  - Can extend traceability extraction to test artefacts.
- GISMOE
  - Could genetically improve  test artefacts for testing the CIT selected products

# 3. Feature-level non-functional sensitivity analysis

- Non-functional properties in SPLS state-of-the-art:
  - Analytical model on single *measured* properties based on covering arrays (Siegmund).
  - *Multi-objective* analysis based on synthetic values (Sayyad, Pascual).

- Our speculation
  - The non-functional sensitivity analysis that GISMOE advocates for SPLs would come from both research trends
    - measured at the right level of granularity and multi-objective

# 4. The need of co-evolution

- GISMOE
  - Relies on co-evolution of the programs and their tests
- ECCO
  - SPLs need to keep multiple types of artefacts in synch across all the products (i.e. variability)
- A promising area of research
  - Multi-view consistency checking (Lopez-Herrejon)
    - Maps constraints to propositional logic to verify that structural dependencies across artefacts are satisfied

# 5. Human in the loop

- GISMOE advocates user involvement to:
  - Lower adoption barrier
  - Employ knowledge only available from the experts
- ECCO:
  - Provides hints for missing or surplus modules that need to be added or removed for SPL maintenance or evolution
  - Employs user feedback to refine traces
- Open issues:
  - Include other sources of domain knowledge – e.g. ontologies

# The Road Ahead

- Improve the traceability capacity of ECCO
  - Exploit clone detection techniques
  - Investigate control and data flow analysis for SPLs
- Applicability ECCO-GISMOE beyond source code
  - Grafting variability annotations into UML models for SPL architectures
- Adapting GenProg into ECCO
  - Explore how automated repair in GenProg can be adapted into ECCO
  - At first explore using ASTOR – Java-based

We hope to see you on the road!!

# Genetic Improvement for Software Product Lines: An Overview and a Roadmap

Roberto E. Lopez-Herrejon[1], Lukas Linsbauer[1], **Wesley K. G. Assunção [2,3]**, Stefan Fischer[1], Silvia R. Vergilio[2], Alexander Egyed[1]

[1] ISSE - Johannes Kepler University – Austria
[2] DINF - Federal University of Paraná – Brazil
[3] COINF - Technological Federal University of Paraná – Brazil

{roberto.lopez, lukas.linsbauer,stefan.fischer, alexander.egyed}@jku.at
{wesleyk, silvia}@inf.ufpr.br

**This work was supported by:**
**- Brazilian Agencies CAPES: 007126/2014-00 and CNPQ: 453678/2014-9**
**- The Austrian Science Fund (FWF): P 25289-N15**