# Seamless Variability Management With the Virtual Platform


**Wardah Mahmood**

*Chalmers | University of Gothenburg, Sweden*

**Daniel Strüber**

*Chalmers | University of Gothenburg, Radboud University, Netherlands*

**Thorsten Berger**

*Chalmers | University of Gothenburg, Sweden & Ruhr University Bochum, Germany*

**Ralf Lämmel**

*University of Koblenz-Landau, Germany*

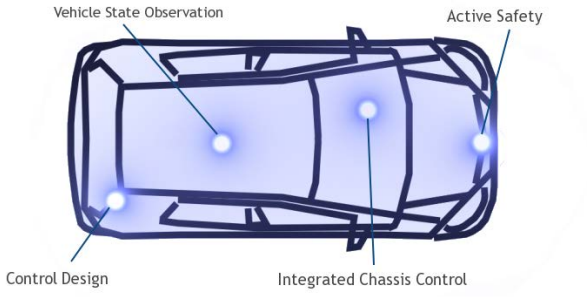
**Mukelabai Mukelabai**

*Chalmers | University of Gothenburg, Sweden*


*Dec 14, 2021*

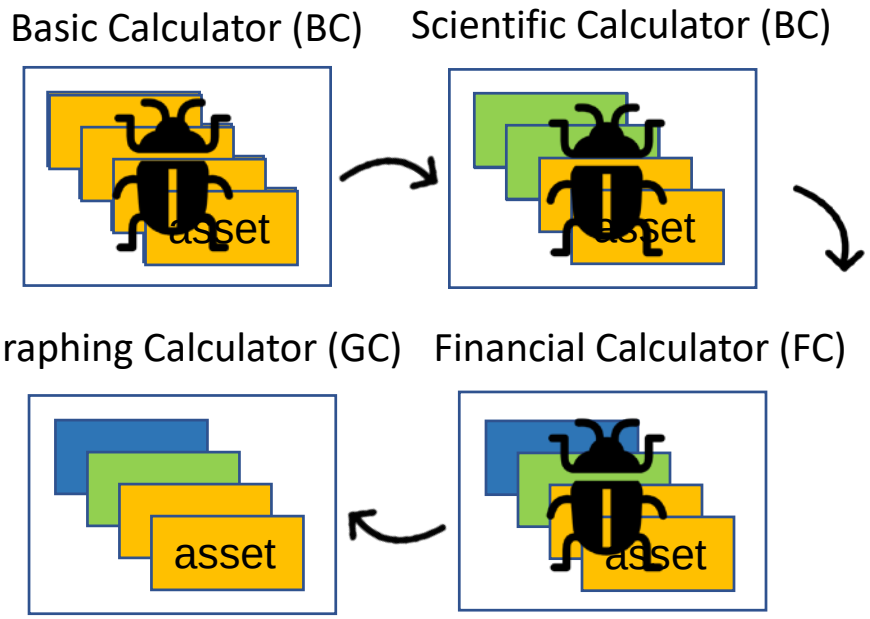
# Variant-rich Systems

- Software systems exist in many variants
- Variant-rich systems prevalent in:
  - Automotive/avionics control systems domain
  - Robotics
  - Highly configurable systems e.g., Linux kernel




# Variant-rich Systems Realization

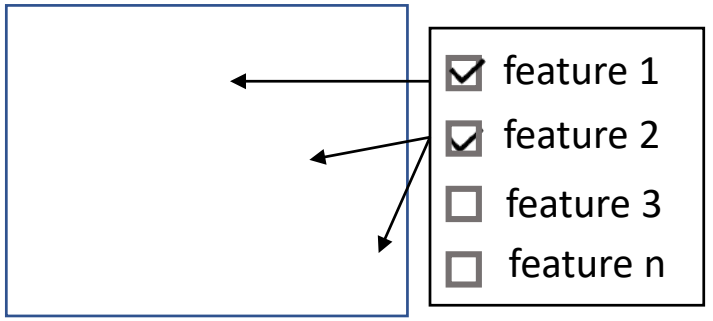
## Clone & own

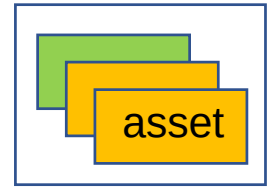



+ Independence
+ Innovation

- Clone detection
- Feature location

## Software product line

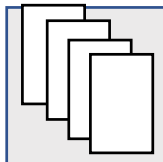



+ Propagation
+ Configuration over Implementation

- Expensive
- Variability-related concepts

# Variant-rich Systems Realization



**Clone & own**

**Software product line**

**Migration?**

- risky
- error-prone
- heuristic-based

Basic Calculator (BC)

Scientific Calculator (BC)

Graphi

Integrated platform

feature 1
feature 2
feature 3
feature n

Configurator

asset

Variant

+ Independence

+ Innovation

- Expensive

- Variability-related concepts

**Cloned product variants: from ad-hoc to managed software product lines**

Julia R

**Enhancing Clone-and-Own with Systematic Reuse for Developing Software Variants**

Stefan Fisc

*Abstract*—To keep pace wi
custom-tailored software system
ice called clone-and-own, wher
system is built by coping and ad
of a single and configurable sys
hoc product portfolios of multip
become impossible to maintain
videspread industrial use becau
nvestments and is intuitive, but i
tic reuse. In this work we prop

**Bottom-Up Adoption of Software Product Lines - A Generic and Extensible Approach**

**Synchronizing Software Variants with VariantSync**

**Tuning GitHub for SPL development: branching models & repository operations for product engineers**


Leticia Montalvillo
University of the Basque Country (UPV/EHU)
ONEKIN Research Group - Facultad de Informática - San Sebastián, Spain
leticia.montalvillo@ehu.es

Oscar Díaz
University of the Basque Country (UPV/EHU)
ONEKIN Research Group - Facultad de Informática - San Sebastián, Spain
oscar.diaz@ehu.es



**ABSTRACT**

SPLs distinguish between domain engineering (DE) and application engineering (AE). Though each realm has its own lifecycle, they might need to be regularly synchronized to avoid SPL erosion during evolution. This introduces two sync paths: *update propagation* (from DE to AE ) and *feed-*

changes) [4] & product upgrade (where latest versions of reusable assets are propagated to products) [2, 20]. As for the feedback path, it is being proposed to extend the scope of the product line to emerging application engineering requirements [15]. However, synchronization is achieved not between artifacts but artifact versions. This requires prop-

# Exploit the spectrum*

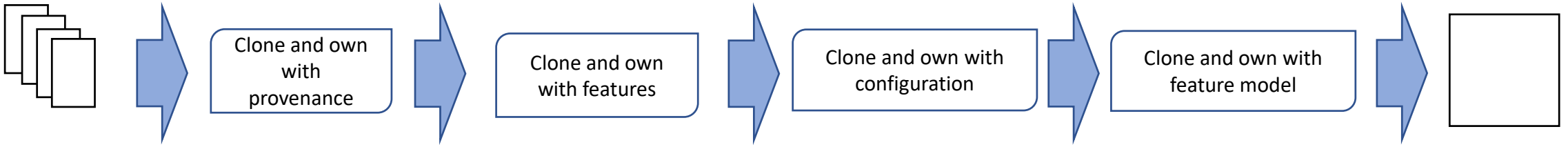
Bridge gap using different governance levels

Incremental benefits for incremental effort



Antkiewicz, Michał, et al. "Flexible product line engineering with a virtual platform." *Companion Proceedings of the 36th International Conference on Software Engineering*. 2014.

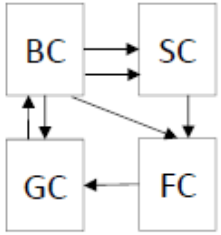
# Bridge the gap




**Clone & own**

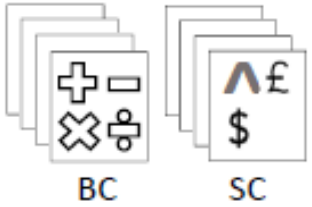
BC SC GC FC

**Store traceability**



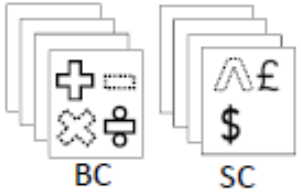

+ No clone detection

**Add features**



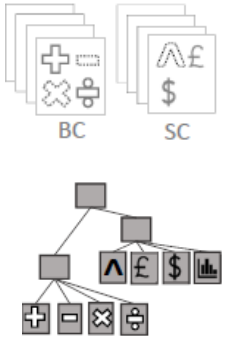

+ No feature location

**Introduce variability mechanisms**




+ Optional features
+ Variation points

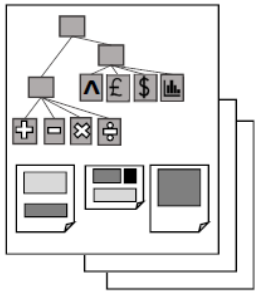
**Structure features into a feature model**




+ Overview
+ Validity

**Level 6: Software product line**

**Reduce redundancy**

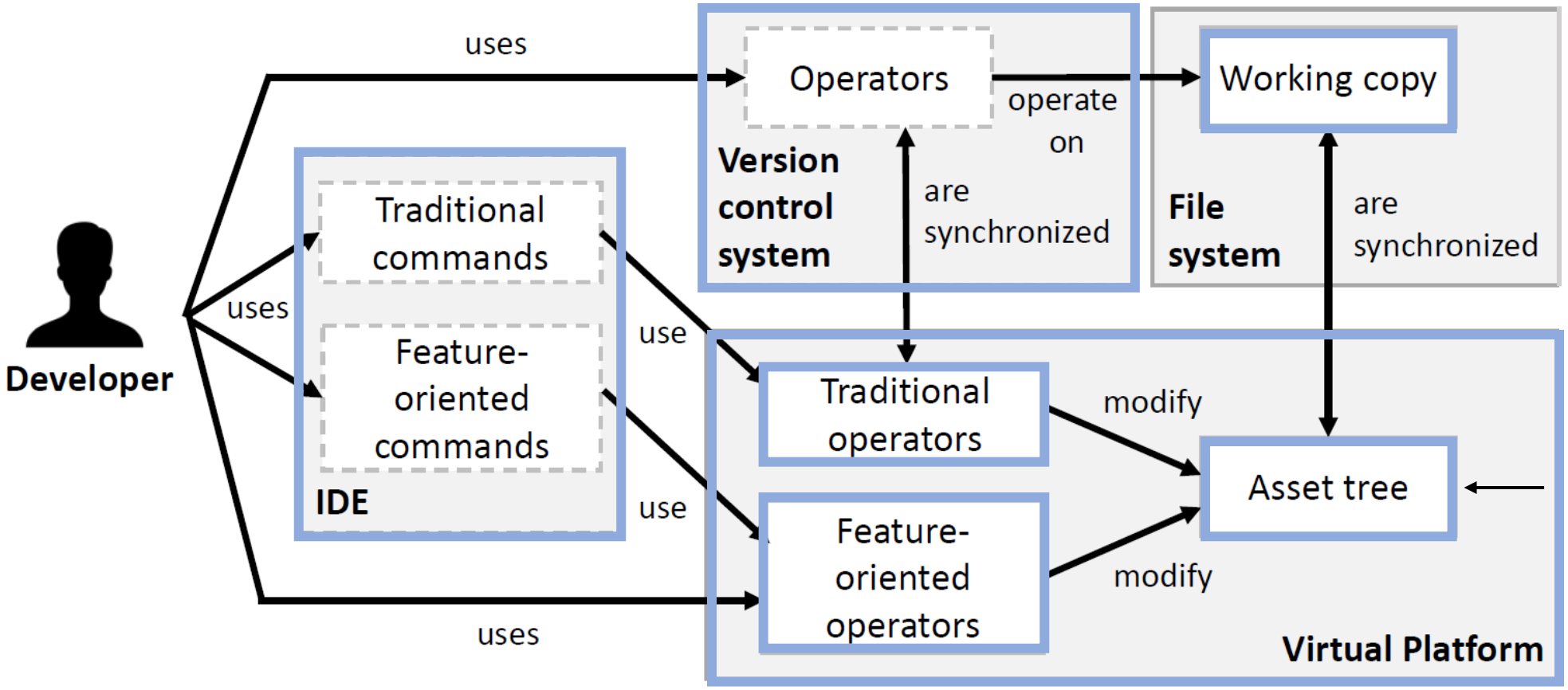


+ Consistency
+ Quality

# Virtual Platform

- Framework allowing to use subset of product line concepts over the spectrum
- Clone management and incremental migration
- **Operators:** dedicated support to move along the spectrum
  - work on **conceptual structures** (semi-structured representations of source code)
  - store and exploit **metadata** that otherwise gets lost (features & mappings, clone traces)
  - language-independent
- Formalization and proof-of-concept implementation
- Evaluation of cost and benefit by replaying development history of a real-world variant-rich system.

# Overview


uses
Operators
operate on
Working copy
Version control system
are synchronized
File system
are synchronized
Traditional commands
uses
Feature-oriented commands
IDE
Developer
use
use
Traditional operators
modify
Asset tree
Feature-oriented operators
modify
uses
Virtual Platform
Conceptual structure

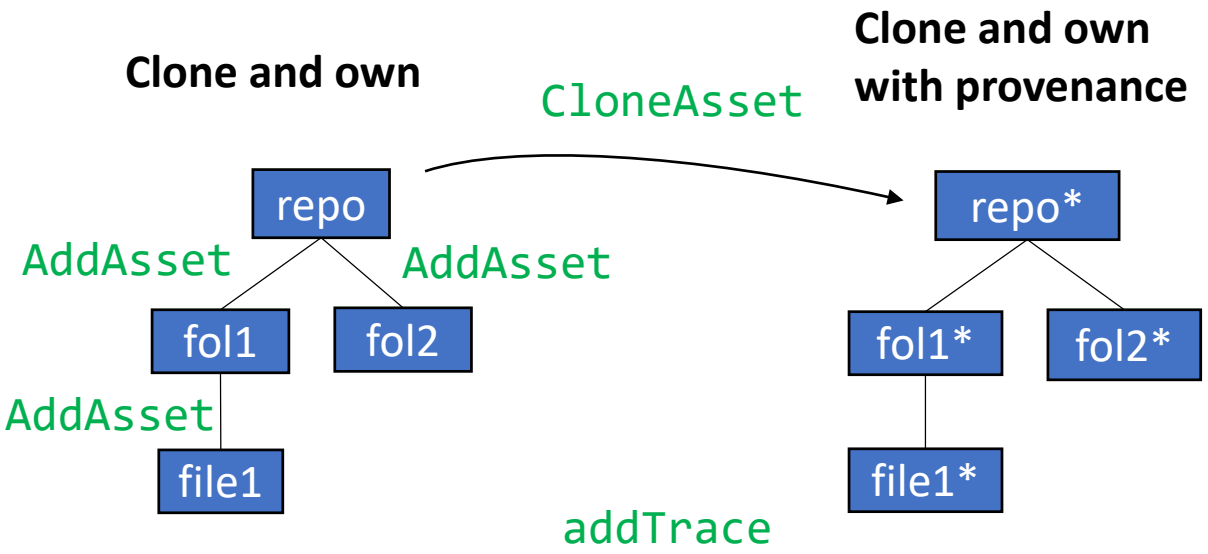
# Conceptual Structures

- **Asset Tree:** hierarchy of assets (Repository, Folder, File, Class, Method, Block)
  - *Asset tree = asset*
  - Versioned

- **Feature:** Set of functionality meaningful to a customer
  - If location not stored, makes maintenance difficult
  - Facilitates migration
  - Can be added in many ways e.g., *#ifdef*
  - Features are mapped to assets via presence conditions

- **Feature model:** Tree of features
  - Structure, relationships, constraints
  - Assets (of all types e.g., repository, folder etc.) can have a feature model

# Operators



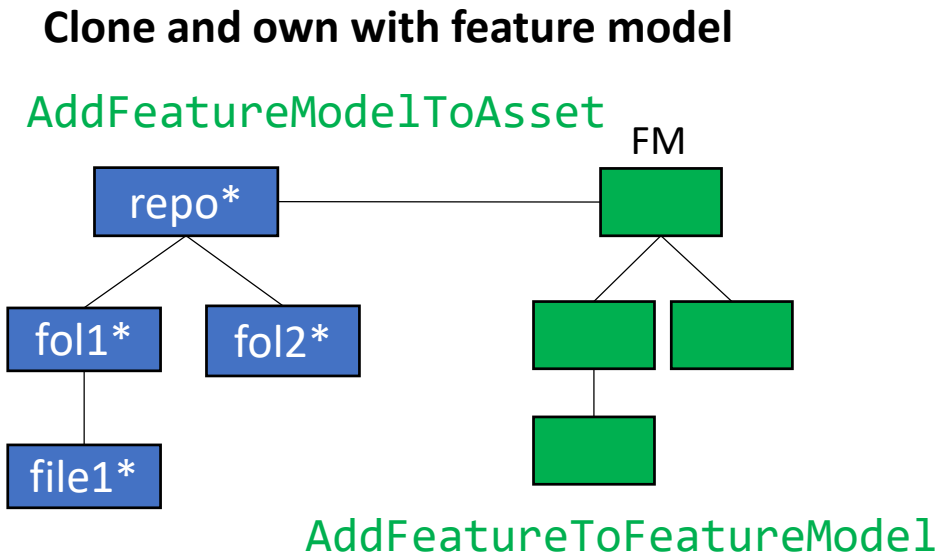

RemoveAsset

ChangeAsset

MoveAsset

**Trace database**

| source | target | version |
|---|---|---|
| repo | repo* | v |
| fol1 | fol1* | v |
| fol2 | fol2 | v |
| file1 | file2 | v |



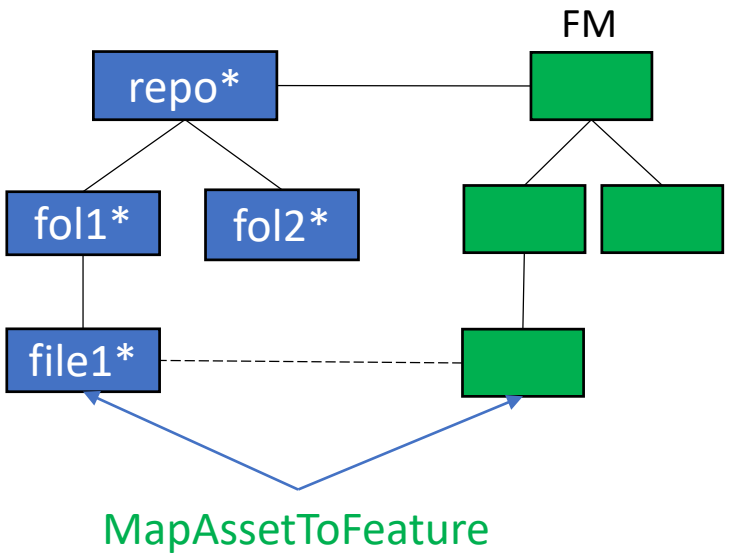

RemoveFeature(feature)

ChangeFeature(feature)

MoveFeature(feature)

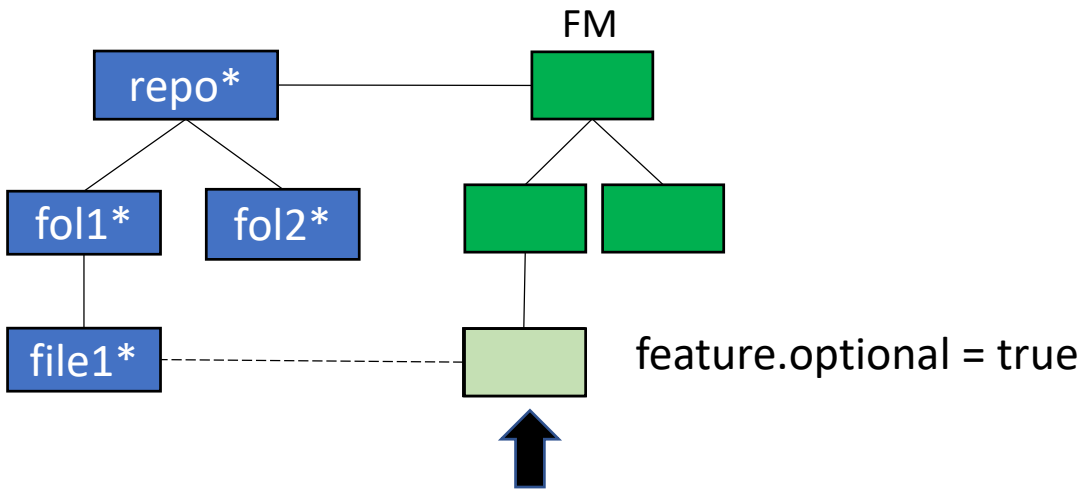
# Operators

**Clone and own with features**




MapAssetToFeature

**Clone and own with configuration**




MakeFeatureOptional

file1.presencecondition = feature & True

# Benefits

- Query system (convenience operators)
  - E.g., getMappedAssets, findClones, detectChanges.
- Clone assets with features (*OP-12.* CloneAsset)
- Clone features with assets (*OP-13.* CloneFeature)
- Propagate changes (*OP-14.* PropagateToAsset and *OP-15.* PropagateToFeature)

# Evaluation

- Scala-based prototype
- Testing of scenarios covering the spectrum (governance levels)

- **Case study:** 4 variants
- Cost-benefit analysis
- Simulate development (special branch)
  - Git diff → virtual platform operators
- Count operator invocations

| operator | freq. | operator | freq. |
|---|---|---|---|
| AddAsset | 3,527 | AddFeature | 229 |
| ChangeAsset | 1,191 | AddFeatureModelToAsset | 4 |
| RemoveAsset | 1,060 | MapAssetToFeature | 368 |
| MoveAsset | 303 | RemoveFeature | 40 |
| CloneAsset | 48 | MoveFeature | 22 |
| PropagateToAsset | 8 | CloneFeature | 54 |
| | | PropagateToFeature | 7 |

# Evaluation

- Incurred cost
  - Adding features and mappings ($cost_{feat}$) + fixing forgotten mappings ($cost_{miss}$)
- Saved cost
  - cost of clone detection ($cost_{clone}$) and feature location ($cost_{loc}$)

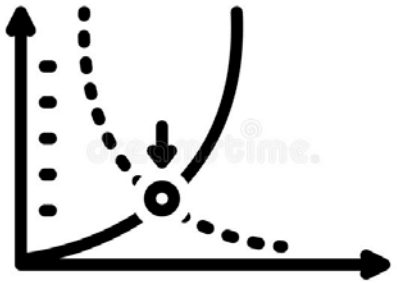
**Benefit** Saved cost – incurred cost

**Assumptions:**

$cost_{loc}$ = 15 minutes*

$cost_{loc} = cost_{clone}$

$cost_{miss} = 10 * cost_{feat}$

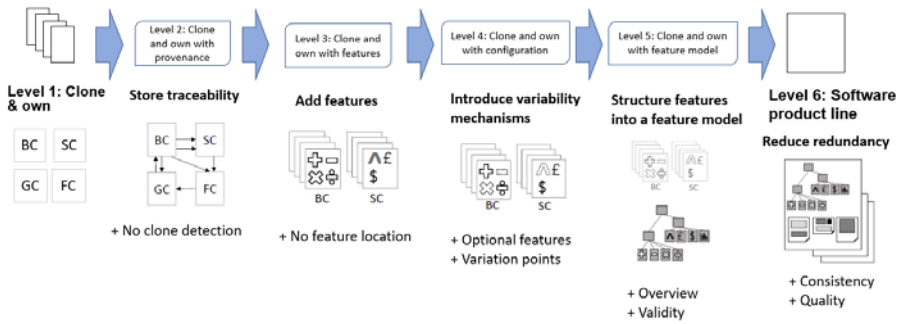


**Break-even point:** 54 seconds

Greater accuracy if used alongside development

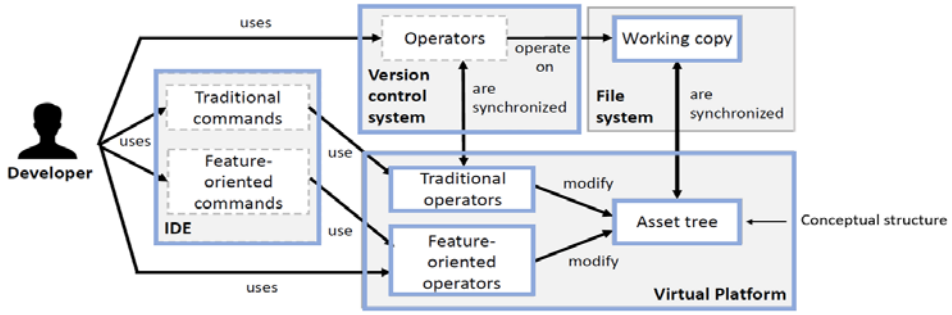
*W. Ji, T. Berger, M. Antkiewicz, and K. Czarnecki, “Maintaining feature traceability with embedded annotations,” in SPLC, 2015.

## Bridge the gap






## Overview

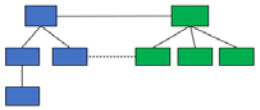





## Operators

**Traditional/Asset-oriented operators:** Conventional operators for asset tree syncing

*OP-1.* AddAsset
*OP-2.* RemoveAsset
*OP-3.* ChangeAsset
*OP-4.* MoveAsset



*OP-11. MapAssetToFeature*

**Feature-oriented operators:** Operators to handle feature models

OP-5. AddFeatureToFeatureModel
OP-6. RemoveFeature
OP-7. ChangeFeature
OP-8. MoveFeature
OP-9. MakeFeatureOptional

Feature model evolution << asset related evolution

OP-10. AddFeatureModelToAsset



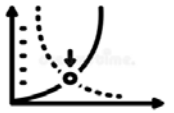
## Evaluation

- Incurred cost
  - Adding features and mappings ($cost_{feat}$) + fixing forgotten mappings ($cost_{miss}$)
- Saved cost
  - cost of clone detection ($cost_{clone}$) and feature location ($cost_{loc}$)

**Benefit** Saved cost – incurred cost

**Assumptions:**

$cost_{loc}$ = 15 minutes*
$cost_{loc} = cost_{clone}$
$cost_{miss} = 10 * cost_{feat}$



**Break-even point:** 54 seconds

Greater accuracy if used alongside development

*W. Ji, T. Berger, M. Antkiewicz, and K. Czarnecki, "Maintaining feature traceability with embedded annotations," in SPLC, 2015.



**Wardah Mahmood**
**Daniel Strüber**
**Thorsten Berger**
**Ralf Lämmel**
**Mukelabai Mukelabai**