Recovering Traceability Links between Artifacts of Software Variants in the Context of Software Product Line Engineering

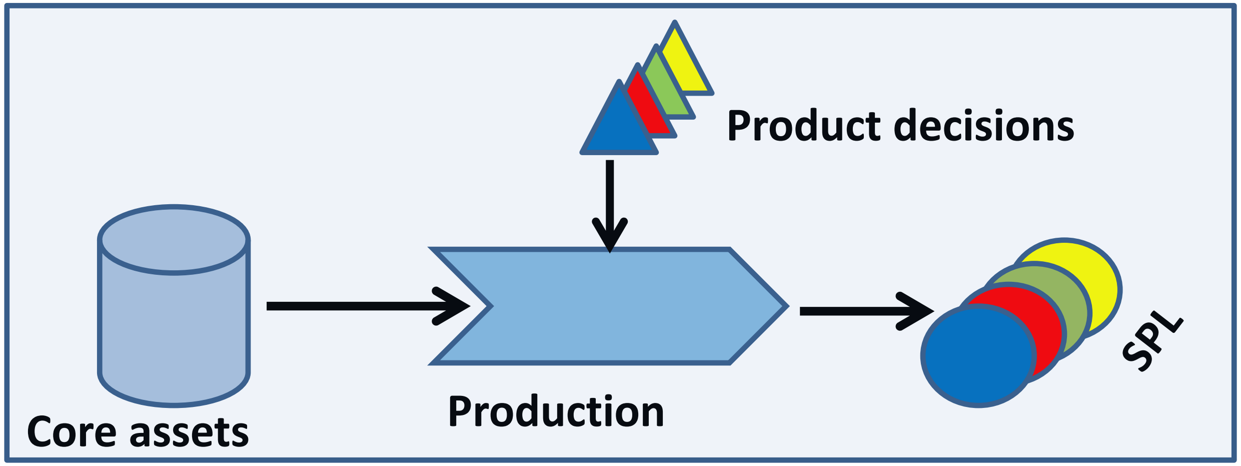
Présentation extraite de la soutenance de thèse de

M. Hamzeh Eyal Salman

Software Product Line (SPL)

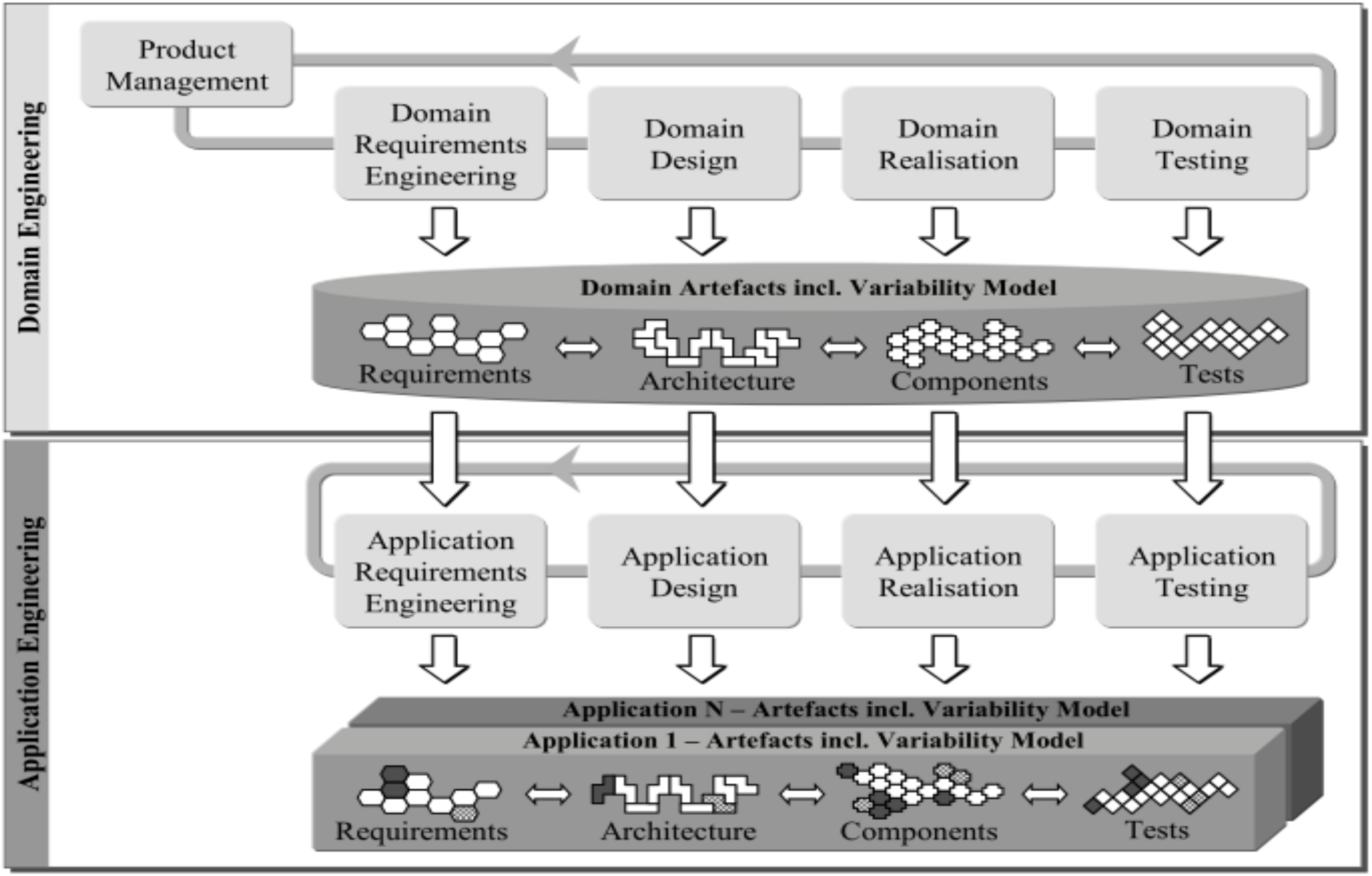
- **A family of software-intensive systems sharing a common, managed set of features developed from a common set of core assets in prescribed way**

- **A feature is a prominent or distinctive user-visible aspect, quality or characteristic of a software system** *[Kang et al. 1990]*



Basics SPL concepts *[Charles Kruger 2006]*

Software Product Line Engineering Framework

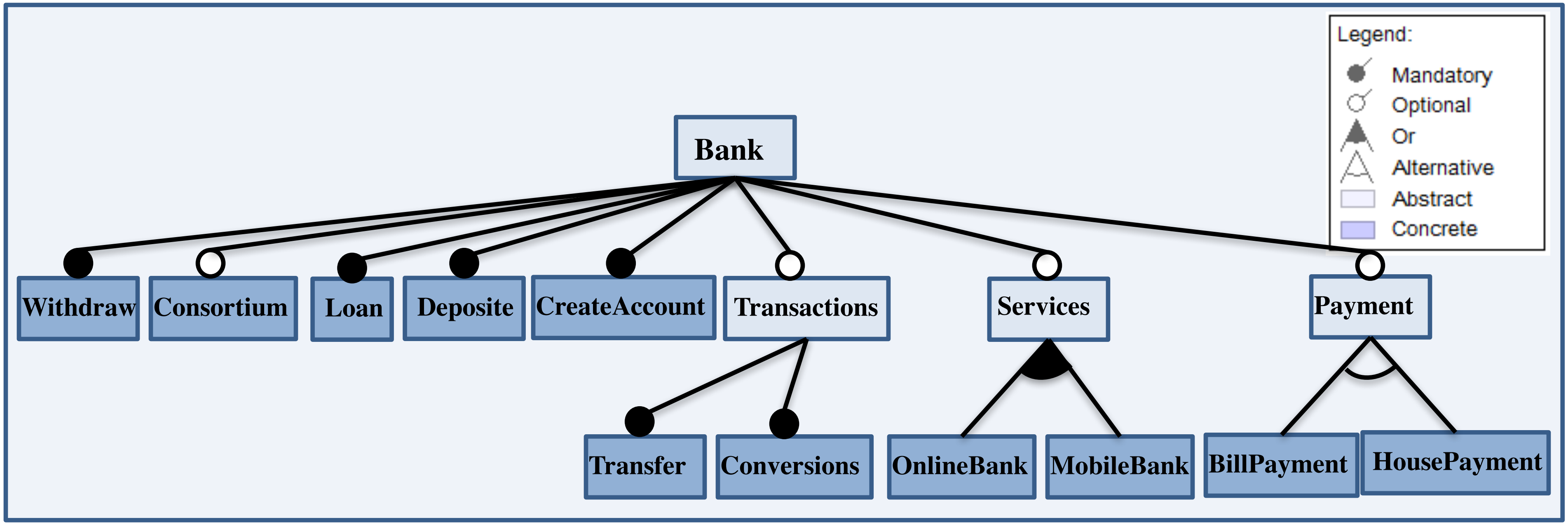


SPLE framework ***[Phol 2010]***

Variability Management: Feature Model (FM)

- Variability represented in FM as
  - Optional features
  - Feature groups
    - Exclusive alternative (XOR)
    - Inclusive alternative (OR)
    - Inclusive (AND)

- Feature groups are variation points (VPs)



An example of feature model

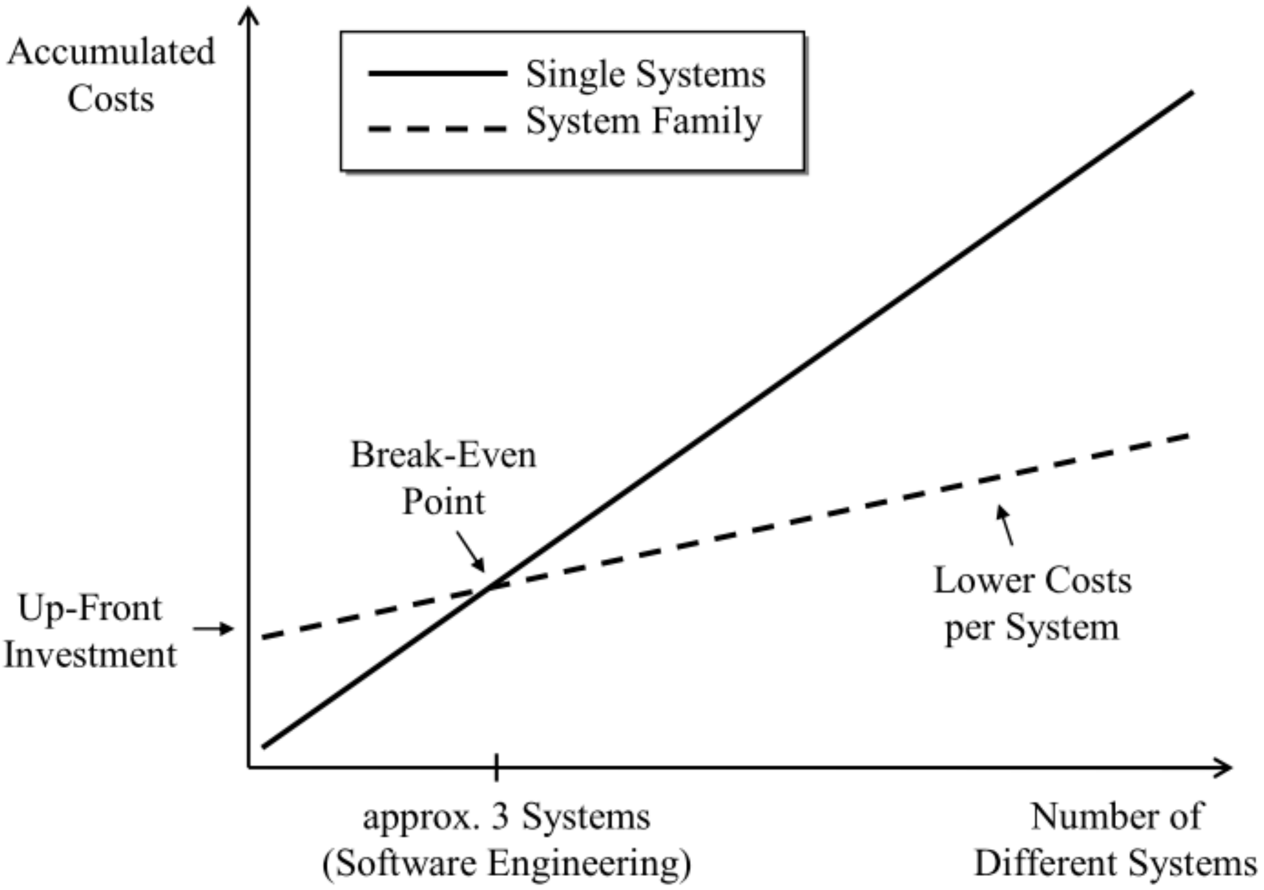
Software Product Variants

- ❑ **Software product variants**
  - ✓ **A collection of similar software products**
  - ✓ **Developed by ad-hoc reuse techniques**
  - ✓ **Share some features and differ in others**

- ❑ **Drawbacks of ad-hoc Development**
  - ✓ **Reusing features (resp. their implementations) is time-consuming**
  - ✓ **Changes made to code of common features must be repeated**
  - ✓ **Evolving product variants lack prescribed planning**

General Goal: Supporting Re-engineering SPL from Product Variants [1/2]

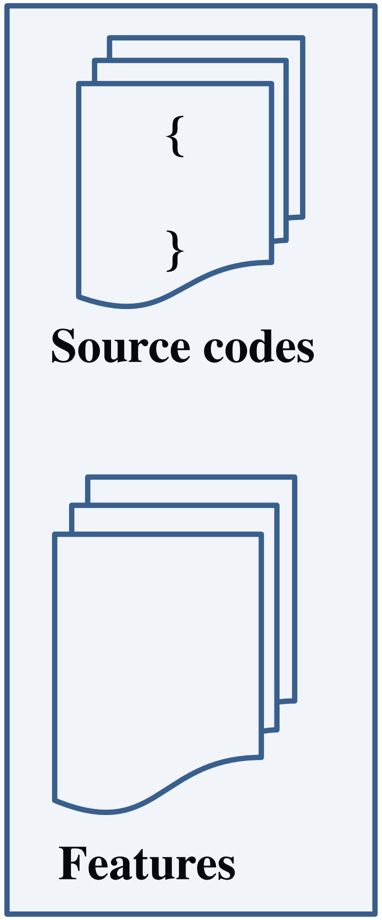
❑ Product variants versus SPL



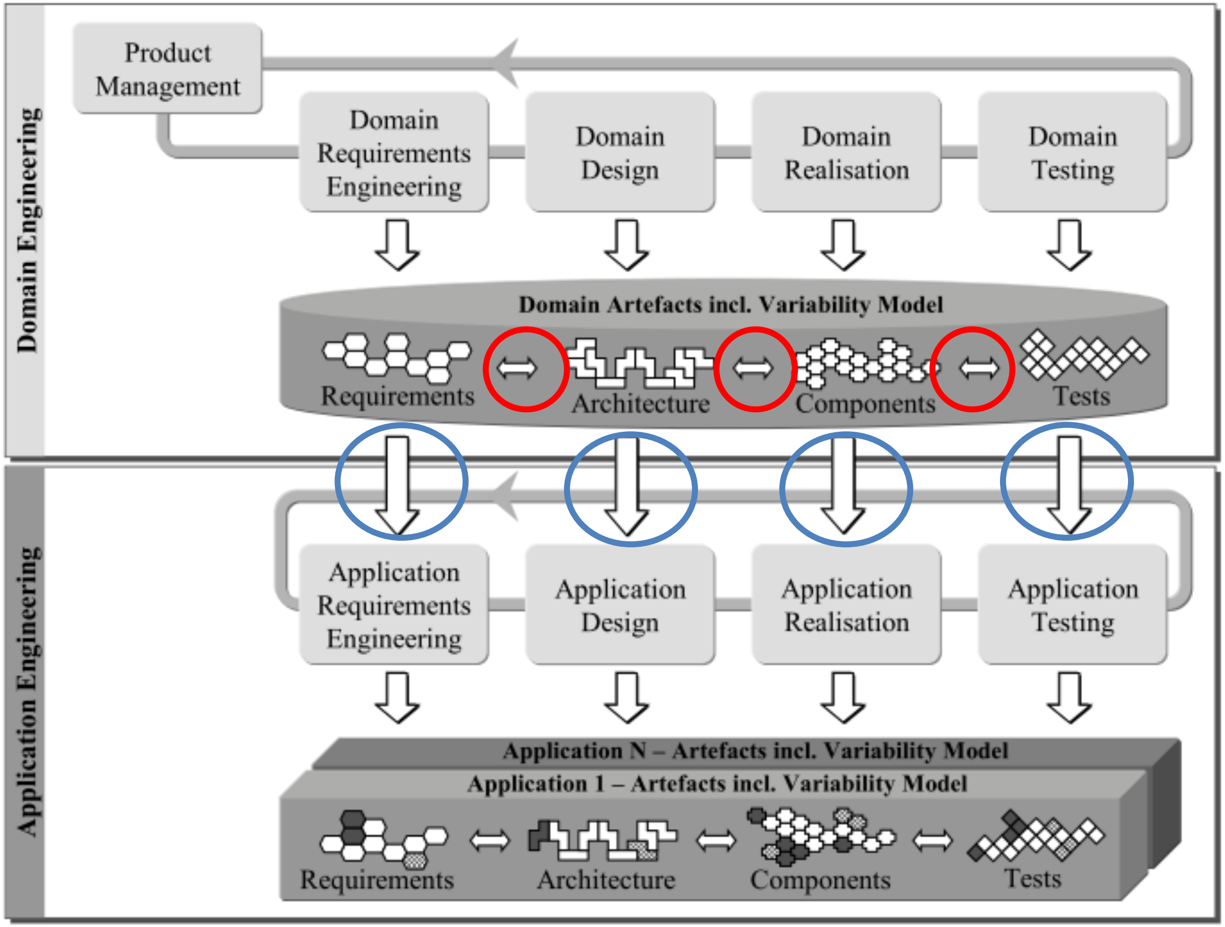
Accumulated costs for SPL development and traditional development [Phol 2010]

General Goal: Supporting Re-engineering SPL from Product Variants[2/2]

- **Traceability is the ability to relate software artifacts developed during the life cycle to describe the system from different perspectives and at different levels of abstraction**



Product variants



SPLE framework *[Phol 2010]*

Problem1: Finding Traceability Links between Features and their Implementing Source Code Elements [1/2]

Implementation of *BillPayment* Feature

```
public class PayPartially
{
private Date PaymentDate;

PayPartially()
{
……
}

private void monthlyPayment (String month)
{
......
}

private void electricityBill (int BillID)
{
......
}

private void telephoneBill (int BillID)
{
......
}
}
```

```
public class PaymentMethod
{
private String price;
 PaymentMethod ()
{
......
}
 private void billInfo (int billNo)
 {
.......
 }
 private void printPaymentReport (int billNo)
 {
 ......
 }
 private void Postpaidbilling ( )
 {
 …..
 }
 private void PrepaiBilling ( )
 {
 …..
 }
 }
```

```
public class BillAccount
{
private int accountType;
private double taxBill;
BillAccount ()
{
……
}

public void payment ( )
{
……
}

public void pricingPllicy ( )
{
…..
}

public double taxesComputing (  )
{
return taxBill;
}
}
```

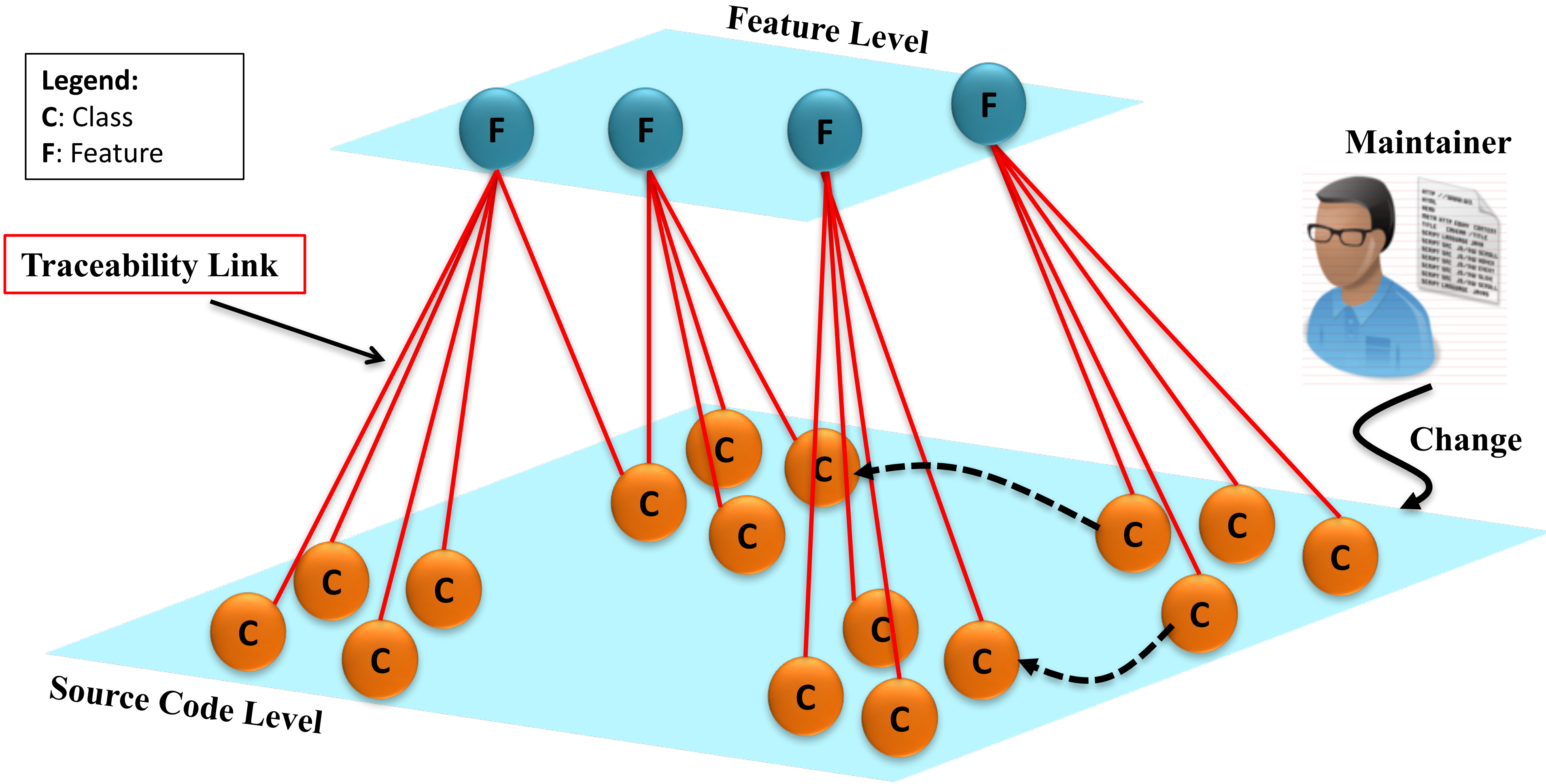
Problem1: Finding Traceability Links between Features and their Implementing Source Code Elements [2/2]

1. **Traceability links between features and their implementing source code elements for:**

   - Understanding source code of product variants
   - Reusing right features (resp. their implementations)
   - Facilitating and Automating new product derivation from SPL's core assets

Problem 2 : Feature-Level Change Impact Analysis

- Change management from SPL manager point of view



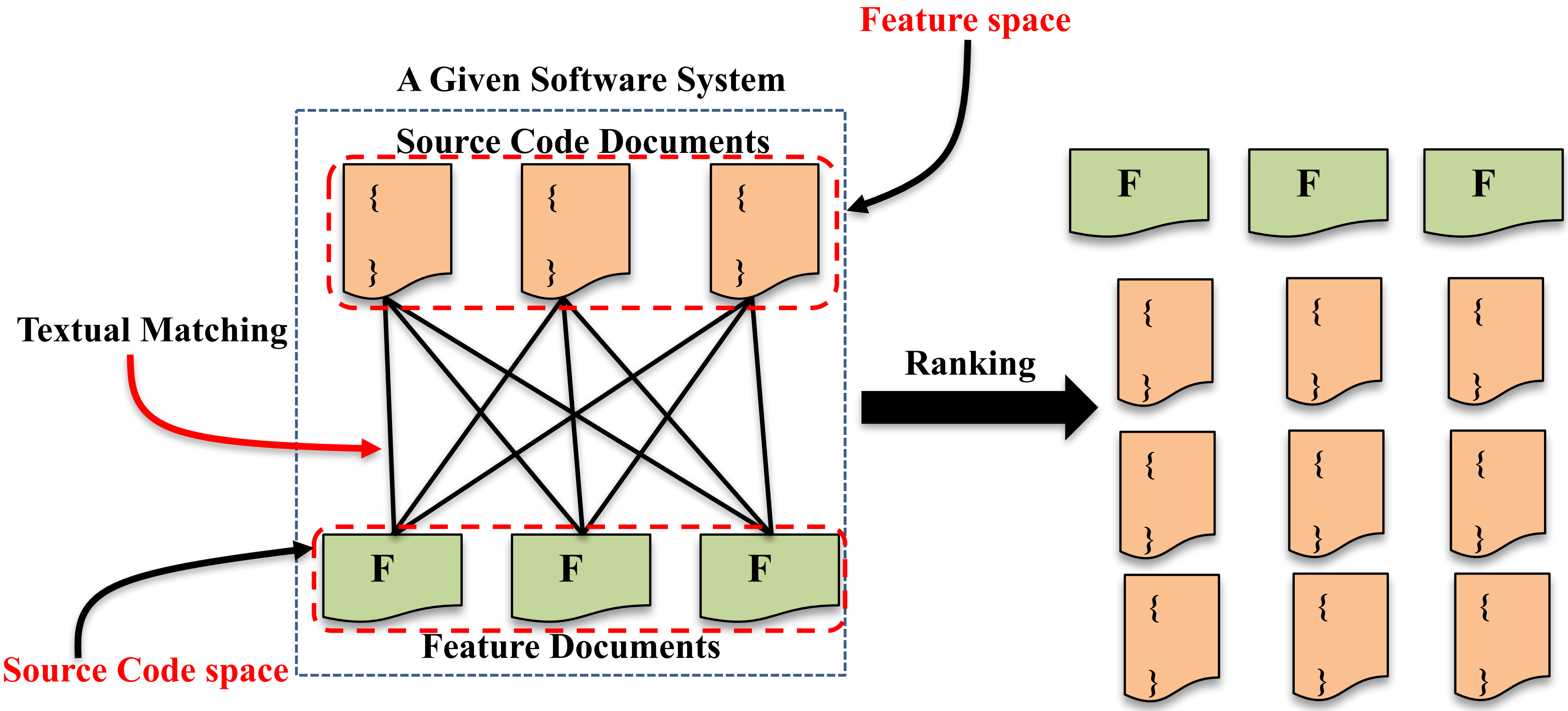
Outline

Feature Location in a Collection of Product Variants with Information Retrieval

Feature Location based on Information Retrieval (IR)

- ❑ Textual matching between feature descriptions and source code information

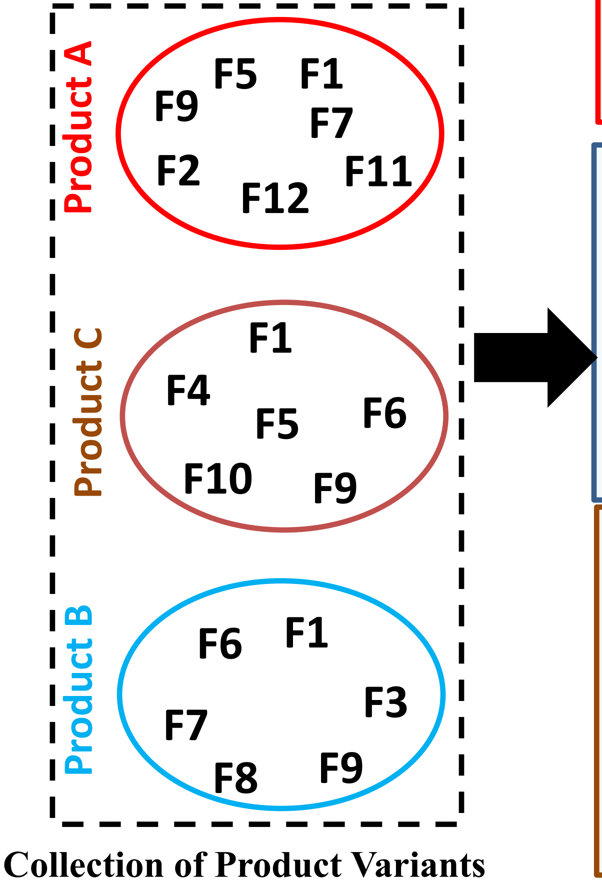
- ❑ Using a threshold mechanism for selection code documents



Conventional Application of IR for Feature Location in a Collection Product Variants

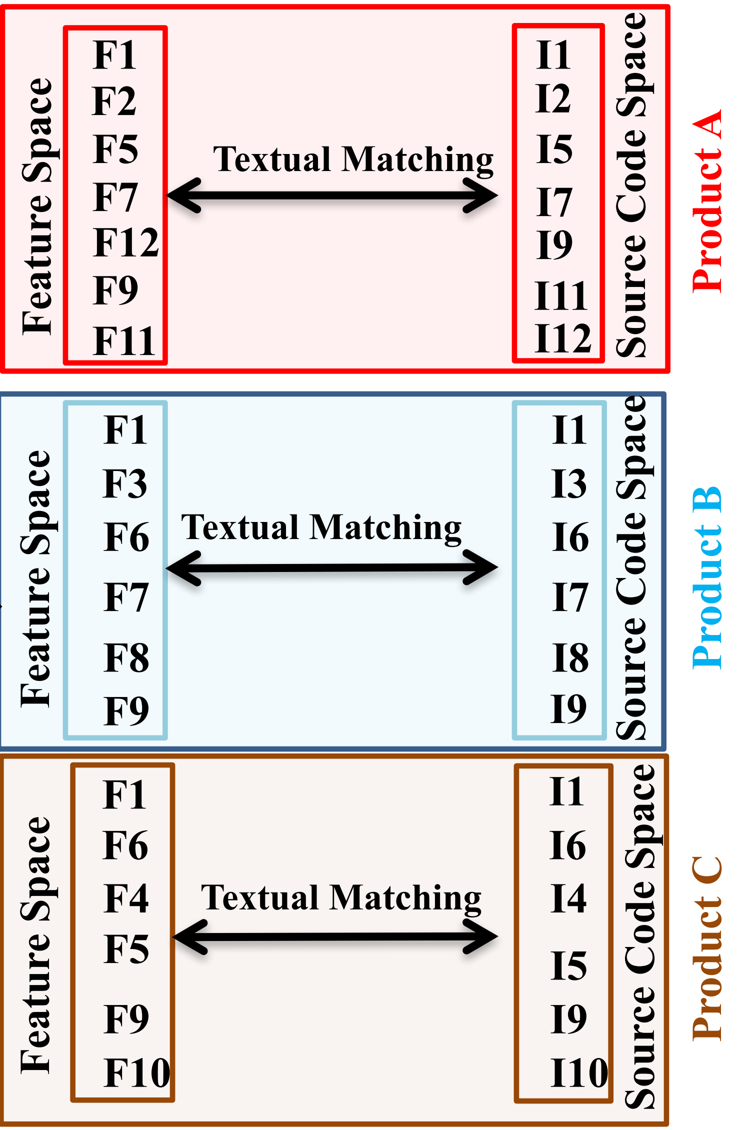
- *[Peng et al., 2013]*
- *[Ali et al., 2011]*
- *[Poshyvanyk et Marcus, 2007]*
- *[Lucia et al., 2008]*
- *[Marcus et al., 2004]*
- *[Antoniol et al., 2002]*

Legend:
I: Implementation
F: Feature



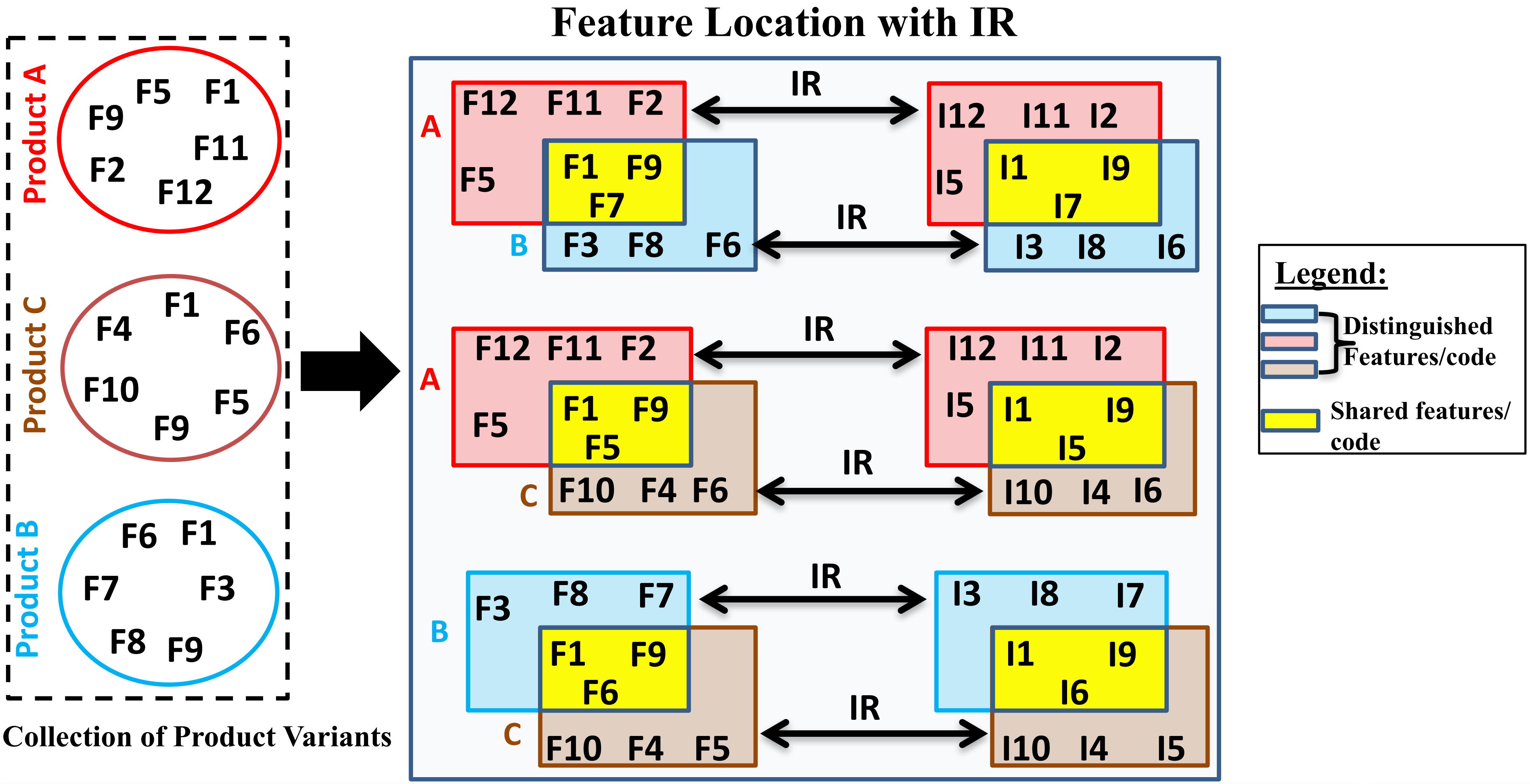
Collection of Product Variants

Feature Location with IR



Variability Analysis for Reducing IR Search Spaces

- ❑ **Exploiting only variability and ignore commonality across product variants pair-wisely** *[Rubin and chechik 2012]*



The Proposed Approach

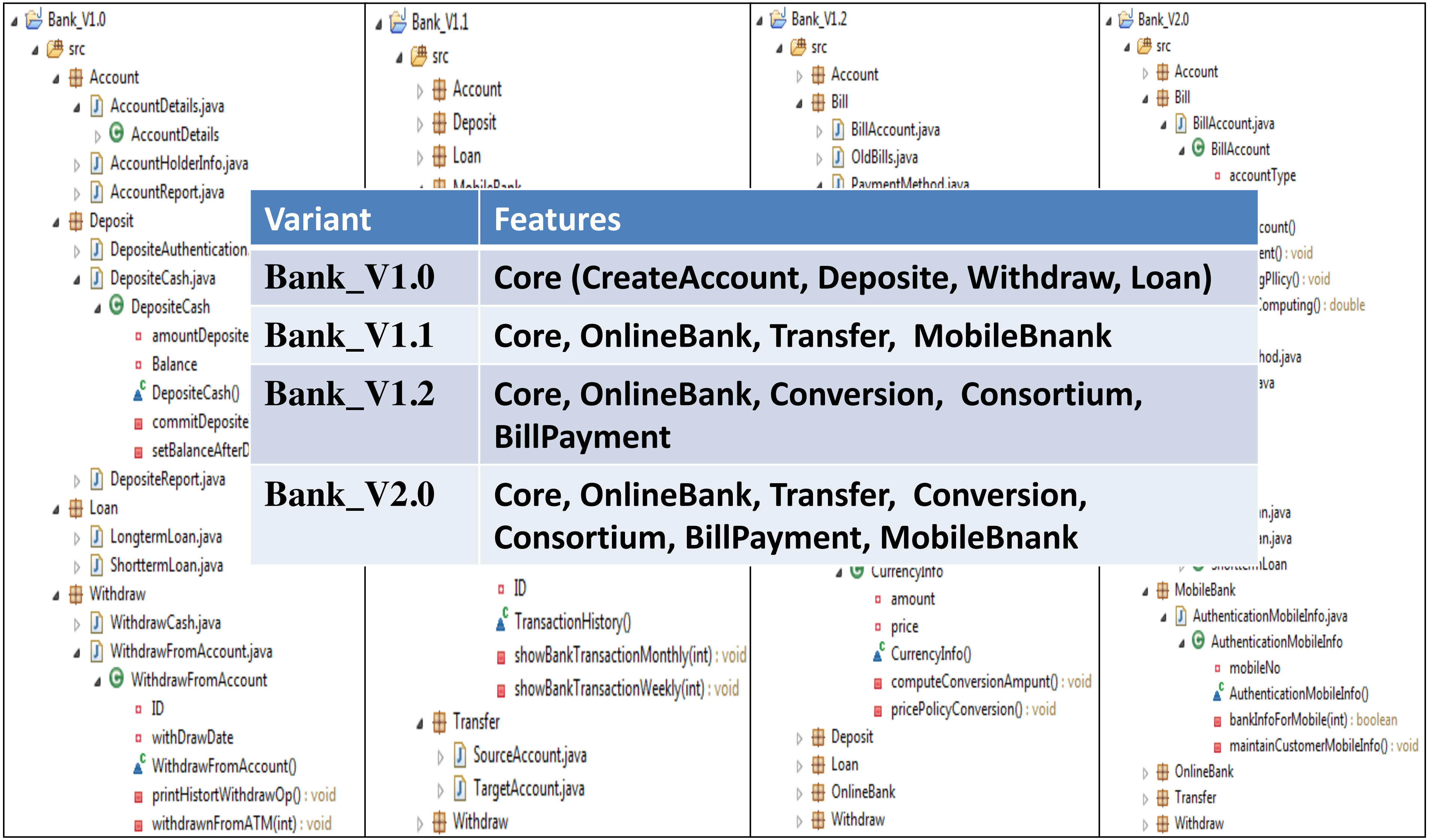
❑ Two strategies to improve the effectiveness of IR-based feature location

1. Reduction the IR search spaces into minimal disjoint sets

2. Reduction the abstraction gap between feature and source code

Illustrative Example

❑ **Four product variants of a software bank system**



```
Bank_V1.0
  src
    Account
      AccountDetails.java
        AccountDetails
      AccountHolderInfo.java
      AccountReport.java
    Deposit
      DepositeAuthentication
      DepositeCash.java
        DepositeCash
          amountDeposite
          Balance
          DepositeCash()
          commitDeposite
          setBalanceAfterD
      DepositeReport.java
    Loan
      LongtermLoan.java
      ShorttermLoan.java
    Withdraw
      WithdrawCash.java
      WithdrawFromAccount.java
        WithdrawFromAccount
          ID
          withDrawDate
          WithdrawFromAccount()
          printHistortWithdrawOp() : void
          withdrawnFromATM(int) : void
```

```
Bank_V1.1
  src
    Account
    Deposit
    Loan
    MobileBank
          ID
          TransactionHistory()
          showBankTransactionMonthly(int) : void
          showBankTransactionWeekly(int) : void
    Transfer
      SourceAccount.java
      TargetAccount.java
    Withdraw
```

```
Bank_V1.2
  src
    Account
    Bill
      BillAccount.java
      OldBills.java
      PaymentMethod.java
        CurrencyInfo
          amount
          price
          CurrencyInfo()
          computeConversionAmpunt() : void
          pricePolicyConversion() : void
    Deposit
    Loan
    OnlineBank
    Withdraw
```

```
Bank_V2.0
  src
    Account
    Bill
      BillAccount.java
        BillAccount
          accountType
          count()
          ent() : void
          gPllicy() : void
          omputing() : double
      hod.java
      ava
      n.java
      n.java
        ShorttermLoan
    MobileBank
      AuthenticationMobileInfo.java
        AuthenticationMobileInfo
          mobileNo
          AuthenticationMobileInfo()
          bankInfoForMobile(int) : boolean
          maintainCustomerMobileInfo() : void
    OnlineBank
    Transfer
    Withdraw
```

Variant	Features
Bank_V1.0	**Core (CreateAccount, Deposite, Withdraw, Loan)**
Bank_V1.1	**Core, OnlineBank, Transfer, MobileBnank**
Bank_V1.2	**Core, OnlineBank, Conversion, Consortium, BillPayment**
Bank_V2.0	**Core, OnlineBank, Transfer, Conversion, Consortium, BillPayment, MobileBnank**

Strategy 1: Reducing IR Spaces [1/2]

1. **Determining common and variable partitions at the feature and source code levels**

   – Textual similarity computing

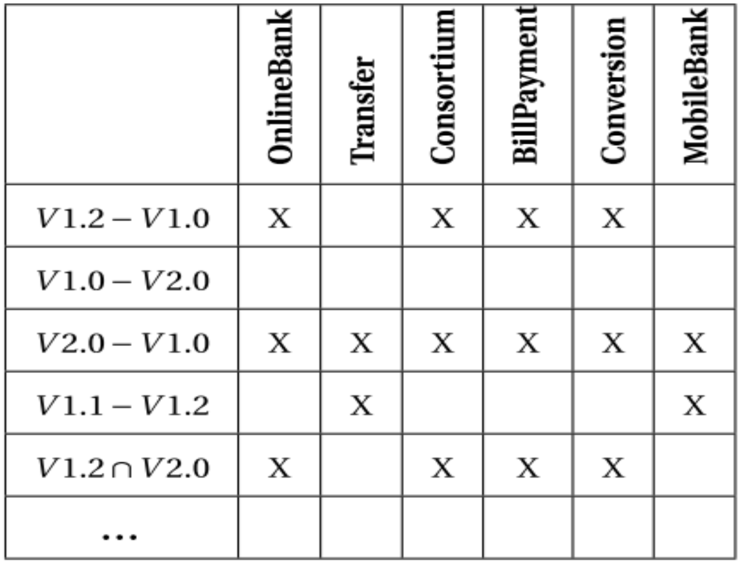
Variant	Features
Bank_V1.0	Core (CreateAccount, Deposite, Withdraw, Loan)
Bank_V1.1	Core, OnlineBank, Transfer, MobileBnank
Bank_V1.2	Core, OnlineBank, Conversion, Consortium, BillPayment
Bank_V2.0	Core, OnlineBank, Transfer, Conversion, Consortium, BillPayment, MobileBnank

Common Partition → Core (CreateAccount, Deposit, Withdraw, Loan)

Variable Partition → OnlineBank, Transfer, MobileBank, Conversation, Consortium, BillPayment

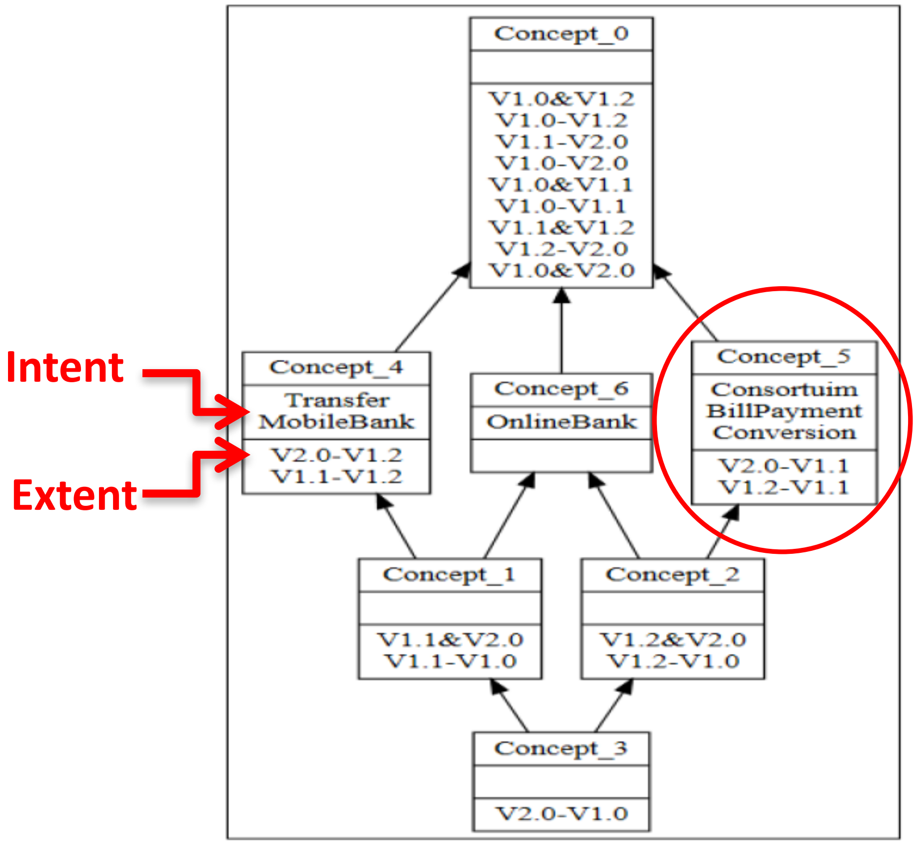
Strategy 1: Reducing IR Spaces [2/2]

2. **Fragmentation of the variable partition at feature and source code levels into minimal disjoint sets**



	OnlineBank	Transfer	Consortium	BillPayment	Conversion	MobileBank
$V1.2 - V1.0$	X		X	X	X	
$V1.0 - V2.0$						
$V2.0 - V1.0$	X	X	X	X	X	X
$V1.1 - V1.2$		X				X
$V1.2 \cap V2.0$	X		X	X	X	
...						

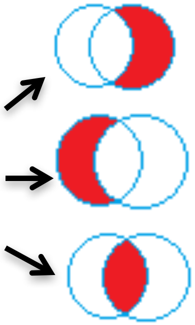
Formal Context



Concept Lattice

Variants-differences

- Bank_V1.2 – Bank_V2.0
- Bank_V2.0 – Bank_V1.2
- Bank_V2.0 ∩ Bank_V1.2



Strategy 2: Reducing Abstraction Gap between Feature and Source Code Levels [1/2]

❑ **What is a code-topic?**

- **It is a cluster of similar classes that have common terms and they also depend on each other.**

❑ **Why code-topic is introduced?**

- **Mainly to get more textual information descripting features implemented by code-topic classes**

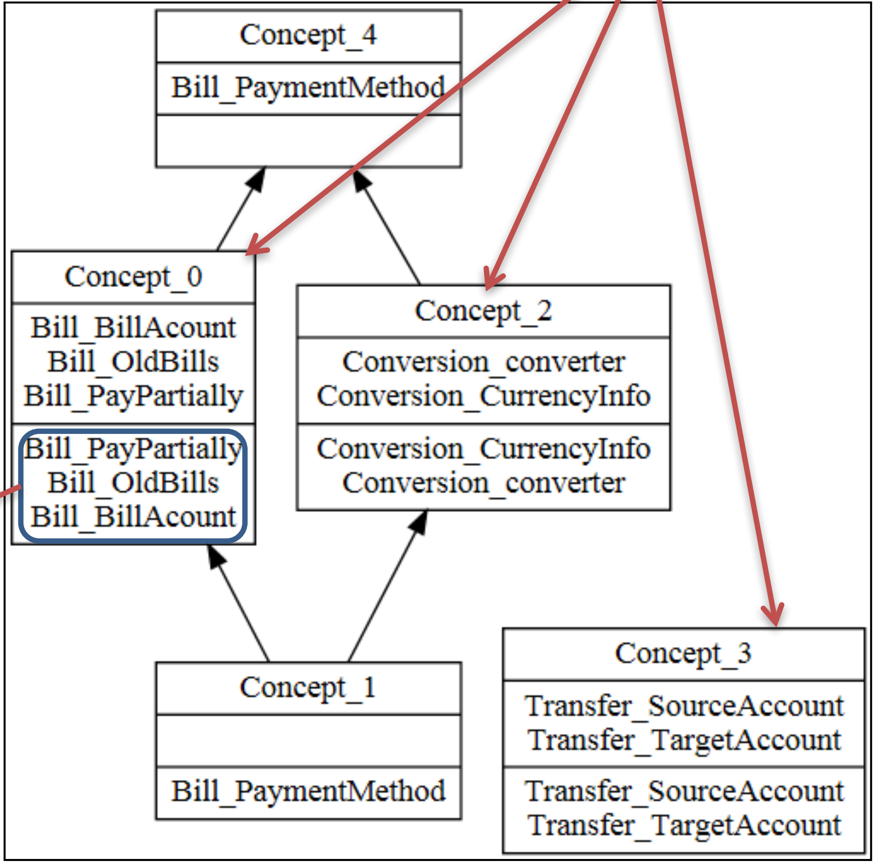
Strategy 2: Reducing Abstraction Gap between Feature and Source Code Levels [2/2]

- **An example for identifying code-topic using FCA**

	Bill_BillAcount	Conversion_converter	Bill_OldBills	Transfer_TargetAccount	Bill_PayPartially	Conversion_CurrencyInfo	Bill_PaymentMethod	Transfer_SourceAccount	…
Bill_BillAcount	X		X		X		X		
Conversion_converter		X				X	X		
Bill_OldBills	X		X		X		X		
Transfer_TargetAccount				X				X	
Bill_PayPartially	X		X		X		X		
Conversion_CurrencyInfo		X				X	X		
Bill_PaymentMethod	X	X	X		X	X	X		
Transfer_SourceAccount				X				X	
…									

Formal Context

Square Concepts

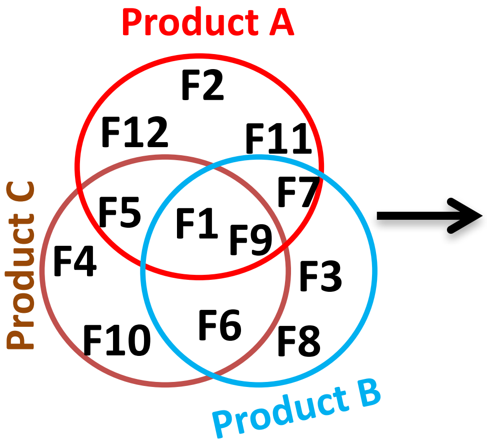


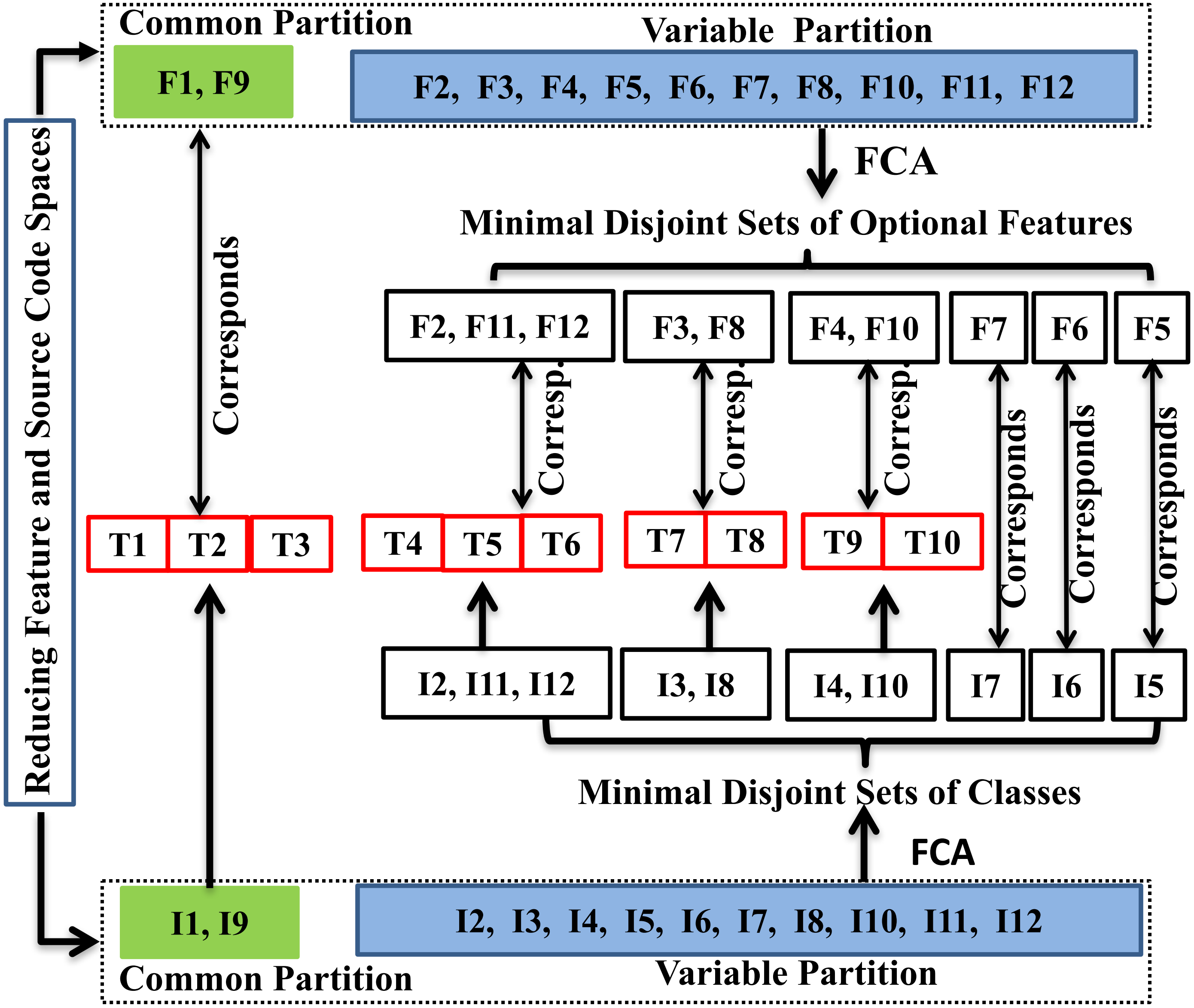
Candidate Code-Topic

Concept Lattice

Graphical Representation of Proposed Strategies

Legend:
I: Implementation Element
F: Feature
T: Code-Topic





Reducing Feature and Source Code Spaces

Common Partition: F1, F9

Variable Partition: F2, F3, F4, F5, F6, F7, F8, F10, F11, F12

FCA

Minimal Disjoint Sets of Optional Features: F2, F11, F12 | F3, F8 | F4, F10 | F7 | F6 | F5

Corresponds

T1 | T2 | T3

T4 | T5 | T6

T7 | T8

T9 | T10

Corresp.

Corresponds

I2, I11, I12 | I3, I8 | I4, I10 | I7 | I6 | I5

Minimal Disjoint Sets of Classes

FCA

Common Partition: I1, I9

Variable Partition: I2, I3, I4, I5, I6, I7, I8, I10, I11, I12

Locating Features by LSI

1. **Linking each feature to their corresponding code-topics using LSI**

   - **For each code-topic there is a document**
   - **For each a feature there is a document**

2. **Decomposing each code-topic to its classes**

Case Studies

❑ Case studies used

- Seven product variants of ***ArgoUML-SPL***
  - » Large-scale system
  - » Well-known case study in our context.
- Five product variants of ***MobileMedia***
  - » Small-scale system

Evaluation Measures

❑ The effectiveness of IR is commonly measured by:

- **Precision:** the percentage of retrieved traceability links that are relevant to the total number of retrieved links

- **Recall:** the percentage of retrieved traceability links that are relevant to the total number of relevant links

- **F-measure:** to find the best possible compromise between recall and precision

Experimental Results

- Comparing our approach (**FCT**) and conventional application of IR (**Conv**)

ArgoUML-SPL						
	Precision		Recall		F-measure	
K	FCT	Conv	FCT	Conv	FCT	Conv
0.01	51%	21%	99%	91%	68%	34%
0.02	52%	22%	86%	82%	65%	35%
0.03	52%	29%	85%	59%	65%	39%
0.04	52%	42%	87%	39%	65%	40%
0.05	63%	56%	73%	25%	63%	36%

Experimental Results [Cont.]

- Comparing our approach (**FCT**) and the most relevant work on the subject (**FL-PV**) *[Xue et al. 2012]*

ArgoUML-SPL						
	Precision		Recall		F-measure	
K	FCT	FL-PV	FCT	FL-PV	FCT	FL-PV
0.1	70%	34%	40%	29%	51%	31%
0.2	57%	07%	09%	04%	16%	05%
0.3	57%	02%	05%	01%	09%	02%
0.4	62%	01%	04%	00%	08%	01%
0.5	57%	00%	02%	00%	03%	00%

Outline

Feature-Level Change Impact Analysis

Step1: Determining the Impact Set of Classes

❑ Statically analyzing the source code of features

- Using abstract syntax tree (AST)

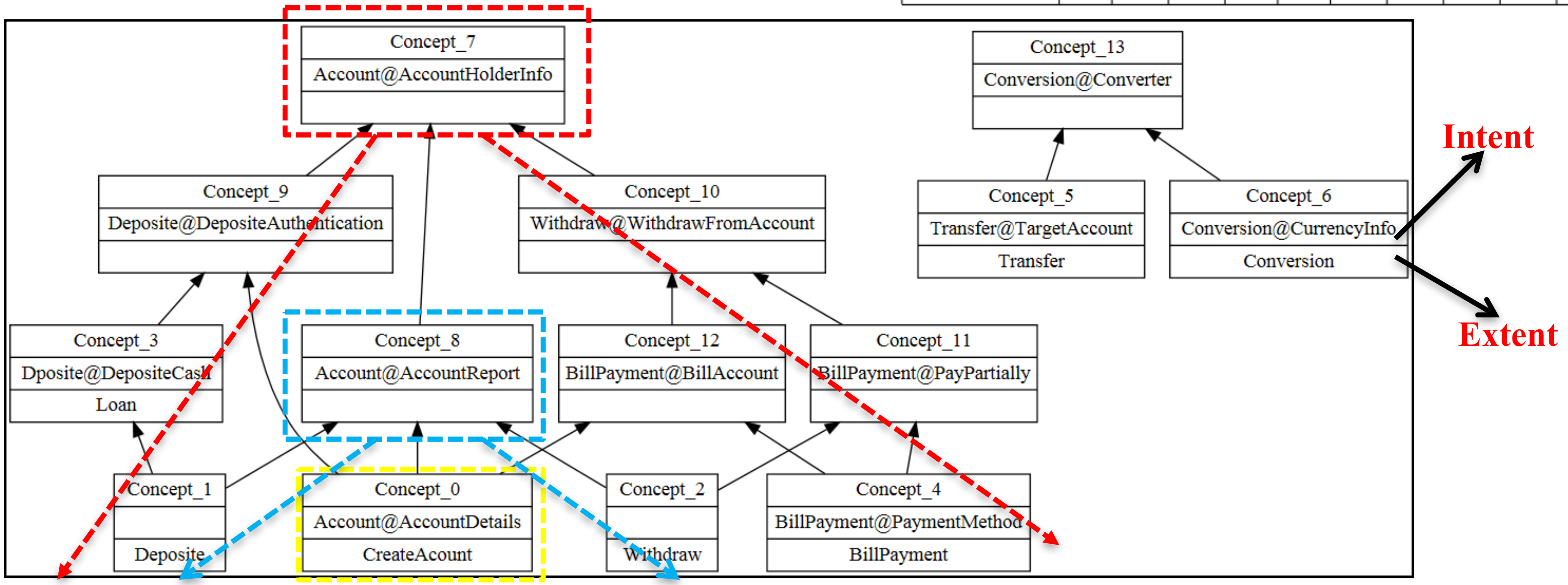
❑ Determining coupled classes based on

1. Inheritance relationship
2. Method call
3. Attribute access
4. Shared attribute access

Step2: Determining Coupled Features Using FCA

- **An example of determining coupled features using FCA**

	Account@AccountHolderInfo	Account@AccountReport	Deposite@DepositeAuthentication	Account@AccountDetails	Withdraw@WithdrawFromAccount	Dposite@DepositeCash	BillPayment@PayPartially	BillPayment@PaymentMethod	BillPayment@BillAccount	Conversion@Converter	Transfer@TargetAccount	Conversion@CurrencyInfo
CreateAcount	X	X	X	X	X				X			
Deposite	X	X	X			X						
Withdraw	X	X			X		X					
Loan	X		X			X						
BillPayment	X				X		X	X	X			
Transfer										X	X	
Conversion										X		X



Concept_7
Account@AccountHolderInfo

Concept_13
Conversion@Converter

Concept_9
Deposite@DepositeAuthentication

Concept_10
Withdraw@WithdrawFromAccount

Concept_5
Transfer@TargetAccount
Transfer

Concept_6
Conversion@CurrencyInfo
Conversion

Concept_3
Dposite@DepositeCash
Loan

Concept_8
Account@AccountReport

Concept_12
BillPayment@BillAccount

Concept_11
BillPayment@PayPartially

Concept_1
Deposite

Concept_0
Account@AccountDetails
CreateAcount

Concept_2
Withdraw

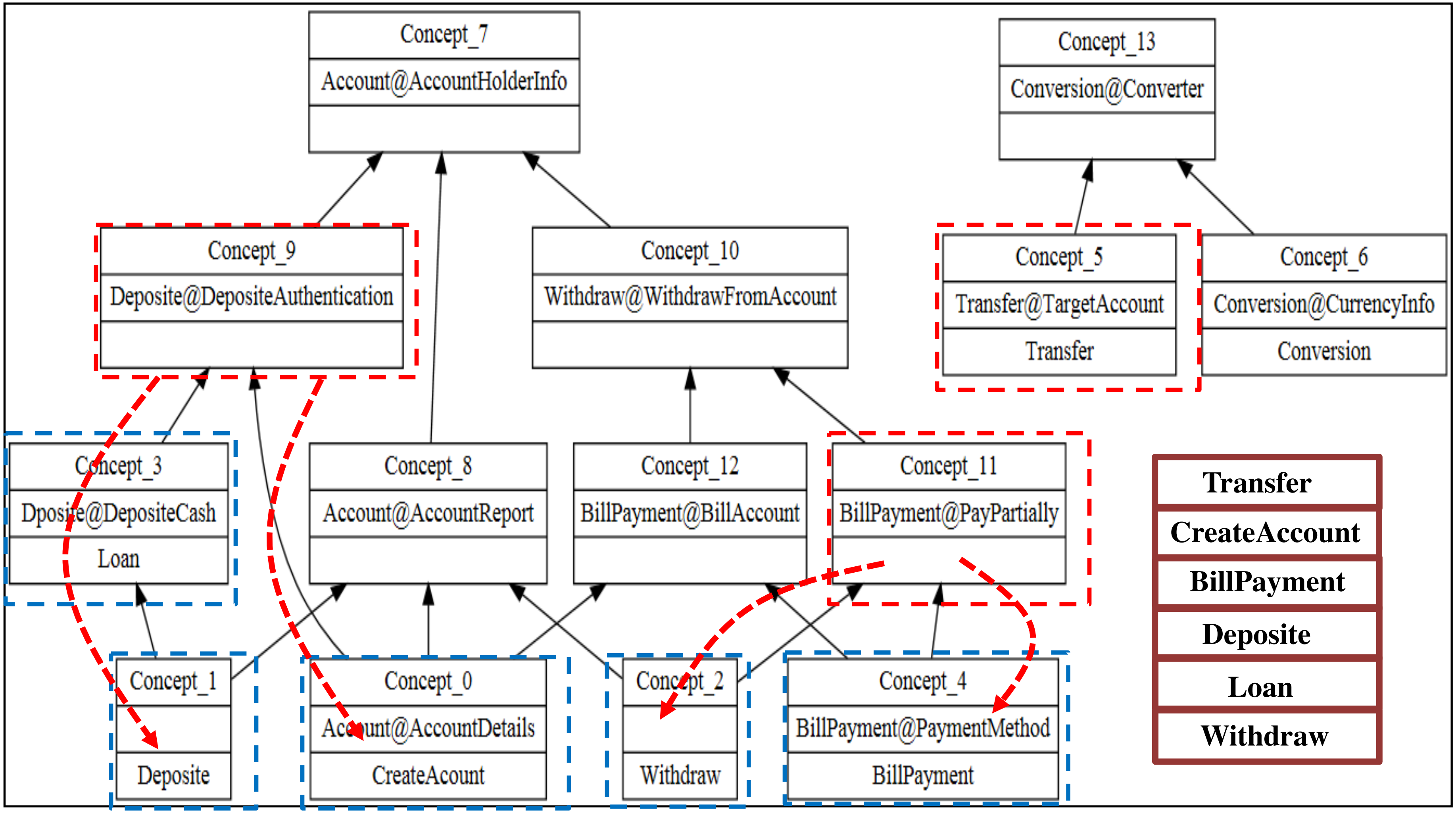
Concept_4
BillPayment@PaymentMethod
BillPayment

Intent

Extent

Step3: Querying concept lattice for determining a ranked list of affected features using impact set of classes

- Consider the impact set of classes consists of {***DeositeAuthentication, TargetAccount, PayPartially***}



Transfer
CreateAccount
BillPayment
Deposite
Loan
Withdraw

Ranking the Affected Features using Impact Degree Metric (IDM)

- We propose two metrics to support feature-level CIA:

1. **Impact Degree Metric (IDM)**
   - To measure the degree to which the implementation of a given feature can be affected.
2. **Changeability Assessment Metric (CAM)**
   - To measure the percentage of features that are affected by a given change.

Concept	Features	IDM	Rank	CAM
Concept_5	Transfer	50%	1	85%
Concept_2	Withdraw	50%	1	
Concept_4	BillPayment	40%	2	
Concept_0	CreateAccount	33%	3	
Concept_3	Loan	33%	3	
Concept_1	Deposite	25%	4	

Experimental Evaluation

❑ Case studies

Case Studies	# Features	# Classes
ArgoUML-SPL	**8**	**515**
MobileMedia	**5**	**28**
BerkeleyDB-SPL	**25**	**227**

Subject core assets and their respective information

– Evaluation measures

1. **Precision:** is the percentage of the estimated affected features that are actually impacted to all estimated affected features

2. **Recall:** is the percentage of the estimated affected features that are really impacted to all actually affected features

3. **F-measure:** to find the best possible compromise between recall and precision

Experimental Evaluation [Cont.]

- » **Precision: [60% - 100%]**
- » **Recall: [75% - 100%]**
- » **F-measure: [67% - 100%]**

- CSC : change set of classes

- EIS : Estimated impacted set of features

CSC	|CSC|	|EIS|	Precision	Recall	F-measure	CAM
MobileMedia						
CSC1	5	5	60%	75%	67%	100%
CSC2	5	6	83%	100%	90%	83%
CSC1	8	6	67%	100%	80%	100%
AgroUML-SPL						
CSC1	9	5	80%	100%	88%	62%
CSC2	8	4	75%	100%	86%	50%
CSC1	18	5	80%	100%	88%	62%
BerkeleyDB-SPL						
CSC1	6	25	92%	100%	96%	92%
CSC2	5	25	100%	100%	100%	100%