

# Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code

Denys Poshyvanyk, Adrian Marcus

*presented by  
Cihan Kaynak*

Software Engineering Research Group  
Department of Computer Science and System Analysis, Miami University

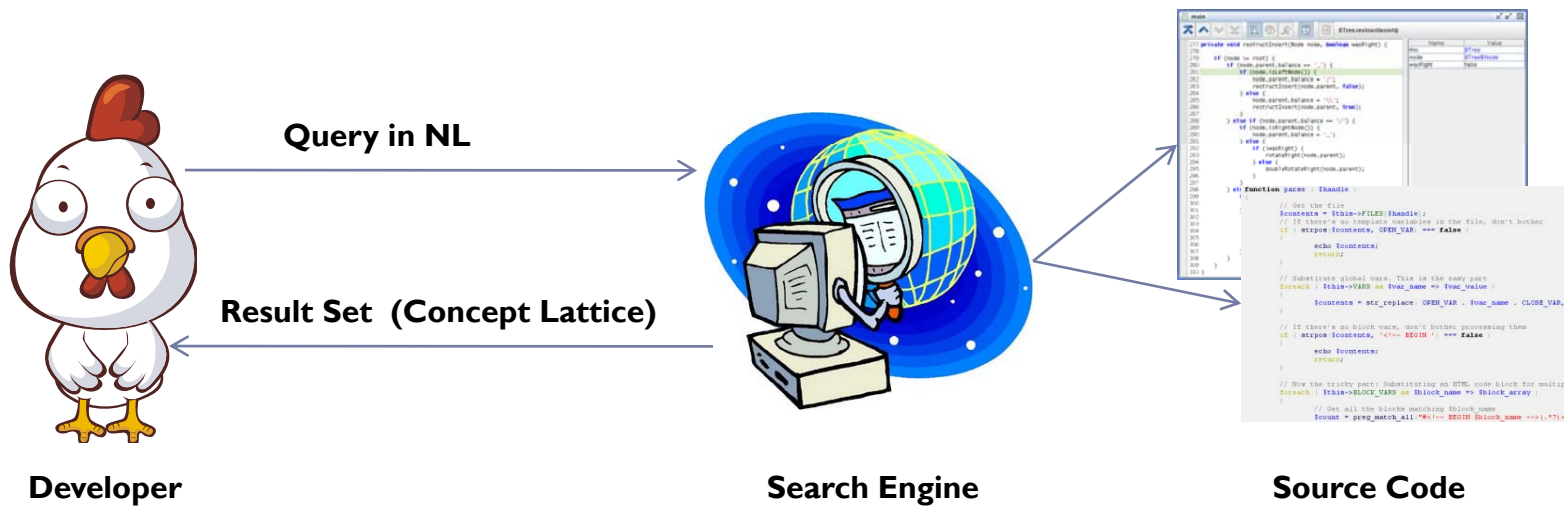
# Introduction

---

- ▶ *Concept (Feature) Location*
  - ▶ Determining the parts of the source code, which correspond to a specific functionality.
  - ▶ Commonly used in program comprehension
- ▶ **Source code text search** is the most commonly used technique
  - ▶ Developer -- Query -- Search Engine = Source code elements
  - ▶ Usually, the low recall values are obtained
    - ▶ Either work in the result set or refine the query
    - ▶ Needs series of query evaluation cycles

# Introduction (cont.)

- ▶ “Our work aims to reduce the searching effort by providing additional structure among the result set in a way that parts of the source code and documentation are grouped according to common topics.”
- ▶ The long sentence above, in fact, implies the integration of classical IR techniques to software concept identification by applying **FCA**.



# Background

---

## ▶ Formal Concept Analysis

- ▶ Mathematical lattice theory that groups *objects*, which share common *attributes*.

Sample Formal Context

O \ A	A1	A2	A3	A4
O1	+	+	+	+
O2		+	+	+
O3		+		
O4	+	+	+	+

$$C1 = ( \{ O1, O4 \}, \{ A1, A2, A3, A4 \} )$$

$$C2 = ( \{ O1, O2, O4 \}, \{ A3, A4 \} )$$

$$C3 = ( \{ O1, O2, O4 \}, \{ A2, A3, A4 \} )$$

$$X = \{ o \in O \mid \forall a \in Y: (o, a) \in R \}$$

$$Y = \{ a \in A \mid \forall o \in X: (o, a) \in R \}$$

# Background (cont.)

---

- ▶ Latent Semantic Indexing (LSI)
  - ▶ LSI is based on a Singular Value Decomposition (SVD) of the co-occurrence matrix of identifiers and comments in source code documents of a software system.
  - ▶ Authors refer us to read the paper “*Indexing by Latent Semantic Analysis*” for further formal details of SVD.
    - ▶ First, creates **term-document** matrix such that rows represents **terms**, whereas **columns** represent documents in the corpus.
    - ▶ Elements in the matrix are weighted by **tf-idf**.
    - ▶ Then, LSI finds a low-rank approximation to the term-document matrix.

# Concept Location using Concept Lattices

---

## ▶ Idea:

- ▶ Apply FCA to organize the results of a search performed by a developer by issuing a query to LSI based source code engine.
  - ▶ Queries are in natural language.
  - ▶ Results are sorted according to the similarity value to the query.

## ▶ Steps:

- ▶ Creating a corpus of a software system
- ▶ Indexing (flexible granularity: methods or classes)
- ▶ **Formulating a query**
- ▶ Ranking documents (classes or methods)
- ▶ **Selecting descriptive attributes (from identifiers and comments)**
- ▶ **Applying Formal Concept Analysis**
- ▶ Examining Results

# Selecting descriptive attributes

---

- ▶  $D$  is a set of documents that forms the corpus such that  $D = \{ d_1, d_2, d_3, d_4, \dots, d_s \}$ .
- ▶  $D_n$  is a set of documents, which is returned and ranked by the search engine after issuing the query.
- ▶  $D^l$  is a set of documents such that  $D^l = D - D_n$ .
- ▶  $T_D$  is a set of unique terms in  $D$ .
- ▶  $T_{D_n}$  is a set of unique terms in  $D_n$ .
- ▶ In order to rank (update its similarity value) every  $t_i \in T_{D_n}$ :

$$sim(t_i, D_n) = sim(t_i, D) - \frac{1}{|D^l|} \times \sum sim(t_i, D^l)$$

# Applying formal concept analysis

- ▶ Locating feature *print page* in the source code of Eclipse

| # of top objects | = 6

| # attributes | = 7

	printer	print	page	job	device	paper	rendering
startJob		X		X			
endJob		X		X			
cancelJob		X		X			
startPage			X			X	X
endPage			X			X	X
getBounds	X				X	X	

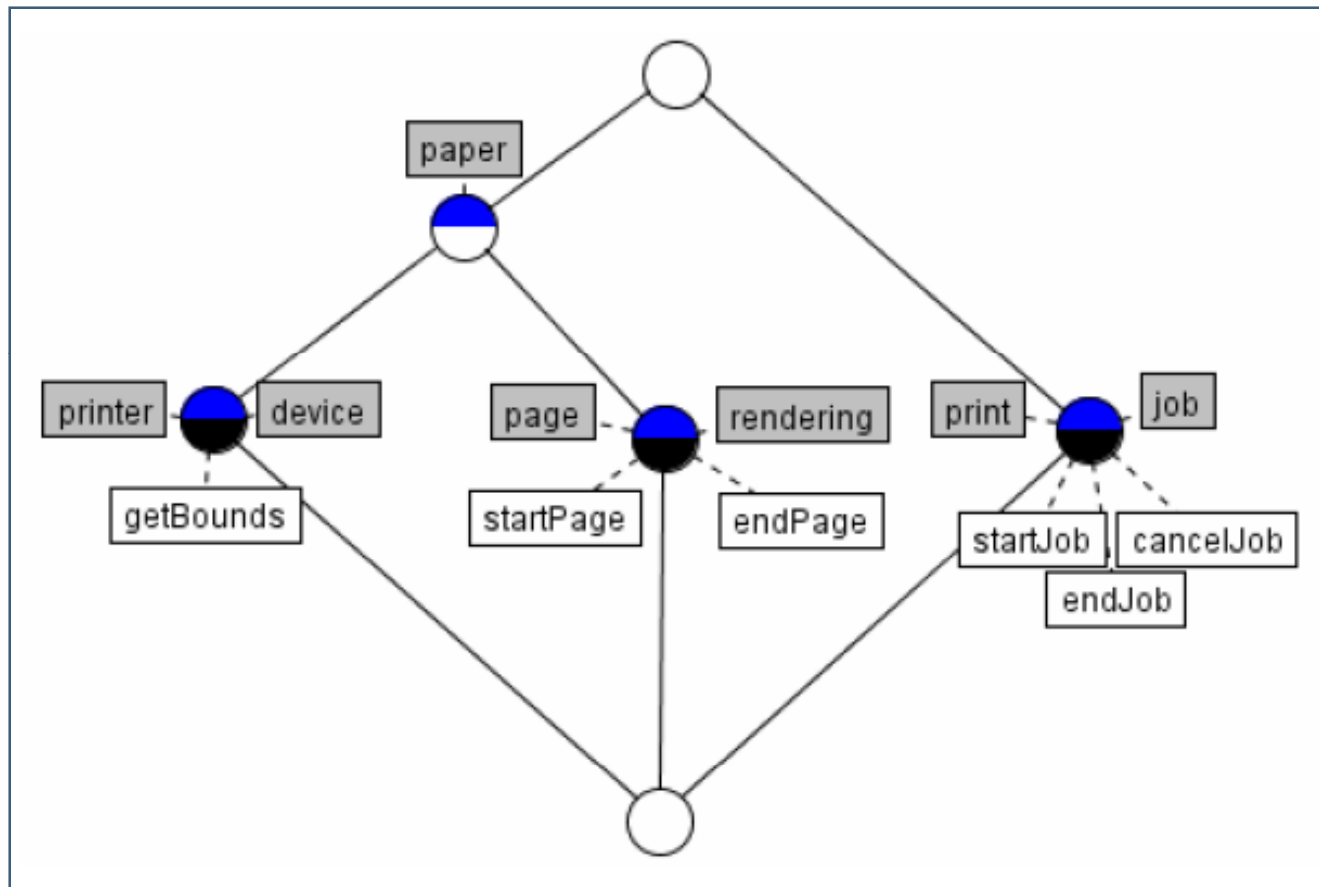
$C_1 = (\{\}, \{\text{paper}\})$

$C_2 = (\{\text{getBounds}\}, \{\text{printer}, \text{device}\})$

$C_3 = (\{\text{startPage}, \text{endPage}\}, \{\text{page}, \text{rendering}\})$

$C_4 = (\{\text{startJob}, \text{endJob}, \text{cancelJob}\}, \{\text{print}, \text{job}\})$

# Applying formal concept analysis



Concept Lattice for the 'print page' feature

# Evaluation of the proposed approach

---

- ▶ They performed a case study to evaluate their approach and better understand the effects selecting  $n$  (# of top objects) and  $k$  (# of most similar terms) to those  $n$  objects.
- ▶ The verification of the results
  - ▶ They focus on **concepts** that are associated with **bugs**
  - ▶ So, they can verify the correctness of their approach by checking final patches done for bugs.

# Evaluation of the proposed approach

---

- ▶ Test environment: Eclipse (version 3.1)

- ▶ It has well documented bug reports.

Item	Count
MLOC	2.9
Vocabulary	56,863
Number of parsed documents	86,208

- ▶ Evaluation criteria and measures:

- ▶ Check the correctness of their new approach
  - ▶ Compare the results of their new technique with the sorted list of results (obtained from LSI)
  - ▶ Measure whether the concept lattice structure effectively groups relevant documents.

# Evaluation of the proposed approach

---

## ▶ Lattice Distillation Factor (LDF)

$$\text{LDF}(C) = \frac{P_{MBA} - P_{RL}}{P_{RL}} \times 100 \%$$

MBA = the minimal part of the lattice that a user should explore to reach the first object, which is relevant to the interested feature

$P_{MBA}$  = precision of minimal browsing area in a concept lattice

$P_{RL}$  = precision of the ranked list

Consider the example from Figure 2 and let us assume that the developer is locating the method which cancels printing operation and the method of interest occurs in position 6, having  $P_{RL}=0.16$ . However, in the concept lattice it is in position 3, thus  $P_{MBA}=0.33$ . Eventually,  $\text{LDF}(C) = (0.33-0.16)/0.16 = 106\%$ , meaning that the concept lattice can distillate related information approximately two times more effectively than the simple ranked list in this particular case.

# Evaluation of the proposed approach

---

## ▶ Lattice Browsing Complexity (LBC)

$$\text{LBC}(C) = \frac{|C_{\text{VIEW}}|}{|C|} \times 100\%$$

$|C_{\text{VIEW}}|$  = number of concepts that are sub-concepts of each node in MBA  
 $|C|$  = total number of concepts in a lattice

Using the same example in Figure 2,  $C_{\text{VIEW}} = 3$ , which is the minimal number of nodes the user has to visit in order to locate the feature of interest. Thus  $\text{LBC}(C) = 3/6 \times 100 = 50\%$ , which means that the developer will need to explore at most half of the nodes in the lattice while locating the feature.

# Evaluation of the proposed approach

---

## ► Locating features in Eclipse

Bug #	Description	Query	Rank	Methods
34160 <sup>2</sup>	The task list, which uses the native table widget, cannot be <i>sorted</i> by clicking on the <i>table headers</i>	“table header sort”	71	org.eclipse.swt.widgets.Table.createHandle <b>org.eclipse.swt.widgets.Table.createWidget</b> org.eclipse.swt.widgets.Table.kEventMouseDown org.eclipse.swt.widgets.Table.itemNotificationProc
25457 <sup>3</sup>	<i>Renaming project</i> to the same name but with different case causes <i>source files</i> to be deleted if project's folder is locked by other application.	“rename project source”	89	<b>org.eclipse.core.internal.localstore.FileSystemStore.move</b>

# Results

---

- ▶ They studied how the **number** of top **documents** and **attributes** affects the size and quality of the concept lattice.
- ▶ # of attributes ranges between 10 and 25
  - ▶  $< 10 \rightarrow$  low clustering capacity
  - ▶  $> 25 \rightarrow$  too many concept nodes in the lattice
- ▶ # of top documents ranges between 80 and 100

# Experimental results

Bug	Docs	Terms	P <sub>MBA</sub>	LDF	C	C <sub>VIEW</sub>	LBC
34160	100	10	0.02	<b>42.8%</b>	17	8	<b>47%</b>
34160	100	15	0.025	<b>78.5%</b>	25	11	<b>44%</b>
34160	100	20	0.026	<b>85%</b>	33	11	<b>33%</b>
34160	100	25	0.033	<b>96.6%</b>	39	10	<b>26%</b>
34160	90	10	0.023	<b>62.3%</b>	17	8	<b>47%</b>
34160	90	15	0.027	<b>98.4%</b>	24	11	<b>46%</b>
34160	90	20	0.029	<b>104%</b>	31	11	<b>36%</b>
34160	90	25	0.039	<b>174%</b>	36	11	<b>31%</b>
34160	80	10	0.026	<b>87.9%</b>	15	7	<b>47%</b>
34160	80	15	0.033	<b>138%</b>	22	10	<b>45%</b>
34160	80	20	0.034	<b>146%</b>	28	10	<b>36%</b>
34160	80	25	0.043	<b>207%</b>	33	10	<b>30%</b>
25457	100	10	0.013	<b>15%</b>	13	8	<b>62%</b>
25457	100	15	0.013	<b>15%</b>	17	9	<b>53%</b>
25457	100	20	0.019	<b>72%</b>	25	10	<b>40%</b>
25457	100	25	0.021	<b>91%</b>	33	10	<b>30%</b>
25457	90	10	0.013	<b>15%</b>	13	8	<b>62%</b>
25457	90	15	0.013	<b>15%</b>	17	9	<b>53%</b>
25457	90	20	0.017	<b>55%</b>	25	11	<b>44%</b>
25457	90	25	0.046	<b>318%</b>	32	13	<b>40%</b>

- ▶ LDF increases when they increase  $k$ :
  - ▶ At the cost of larger lattice sizes (31-39)
  - ▶ LBC is decreasing
- ▶ They obtain best results when applying FCA over top 90 documents with 20-25 attributes.

# Future work

---

- ▶ They want to compare their approach with
  - ▶ Terminological weighting formula
  - ▶ Okapi
- ▶ They plan to propose a heuristic to select attributes.
  - ▶ Source code specific
- ▶ They want to use the rank information of a document to determine the most efficient direction while browsing a concept lattice.

# How can we be inspired from their work?

---

- ▶ Two main focuses in this work
  - ▶ Using formal concept analysis to group related objects
  - ▶ Using information retrieval to locate concepts
- ▶ **Formal Concept Analysis**
  - ▶ We can use that idea to abstract UML class diagrams
    - ▶ Abstraction = Reverse refinement
    - ▶ We can group highly-coupled, related classes located in a low-level design class diagram in order to capture the high-level class, which is, in fact, composed of those low-level concepts.

# Questions & Suggestions

---

