IMPROVING BUG LOCALISATION USING LEXICAL INFORMATION AND CALL RELATIONS

Tezcan Dilshener
Doctoral Research in
Software Evolution and Maintenance
Computing and Communications Department
The Open University, United Kingdom

Introduction

- Software applications require continued maintenance.
- Developers
  - need to comprehend an application prior to maintenance.
  - perform lexical search and navigate the application's structures.
- Many methods to decide which path to take to locate the target one.

Imagine…

1. Input JIRA Story ID and hit return
2. Automated search over the source code starts
3. Result lists all relevant classes for task at hand

Pillar-Two classes affected by CR # 2093	Lexical ranking
QuotaShareContractStrategy.java	2
EventAalLimitStrategy.java	31
LimitStrategyType.groovy	33
EventLimitStrategy.java	4
NoneLimitStrategy.java	38
ILimitStrategy.java	5

Current research

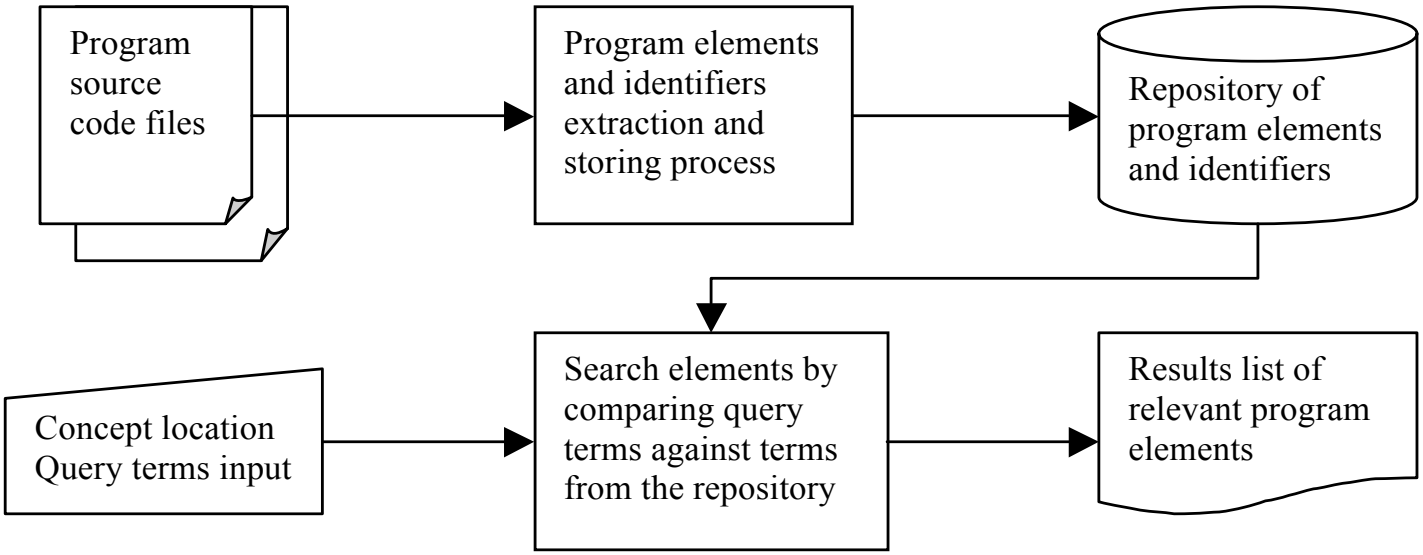
- Recognised the need for combining multiple analysis approaches to support program comprehension
- Combined approaches
  - first obtain a dynamic trace of involved classes
  - attempt to retrieve other relevant classes based on call relations
- Some of the challenges faced are,
  - CR documents tersely described or for non-existing feature.
  - complex class method call relations cause noise in results.

Our aim

- Identify opportunities of using CR vocabulary and application call relations to
  - address challenges faced by current combined approaches
  - reduce the program comprehension overhead.
- ***RQ1***: Does utilizing a combined approach based on lexical information and call relations improve the search performance with respect to a simple string search?
- ***RQ2***: How does the combined approach, implemented in our tool perform compared to another state-of-the-art tool?

Traceability through IR

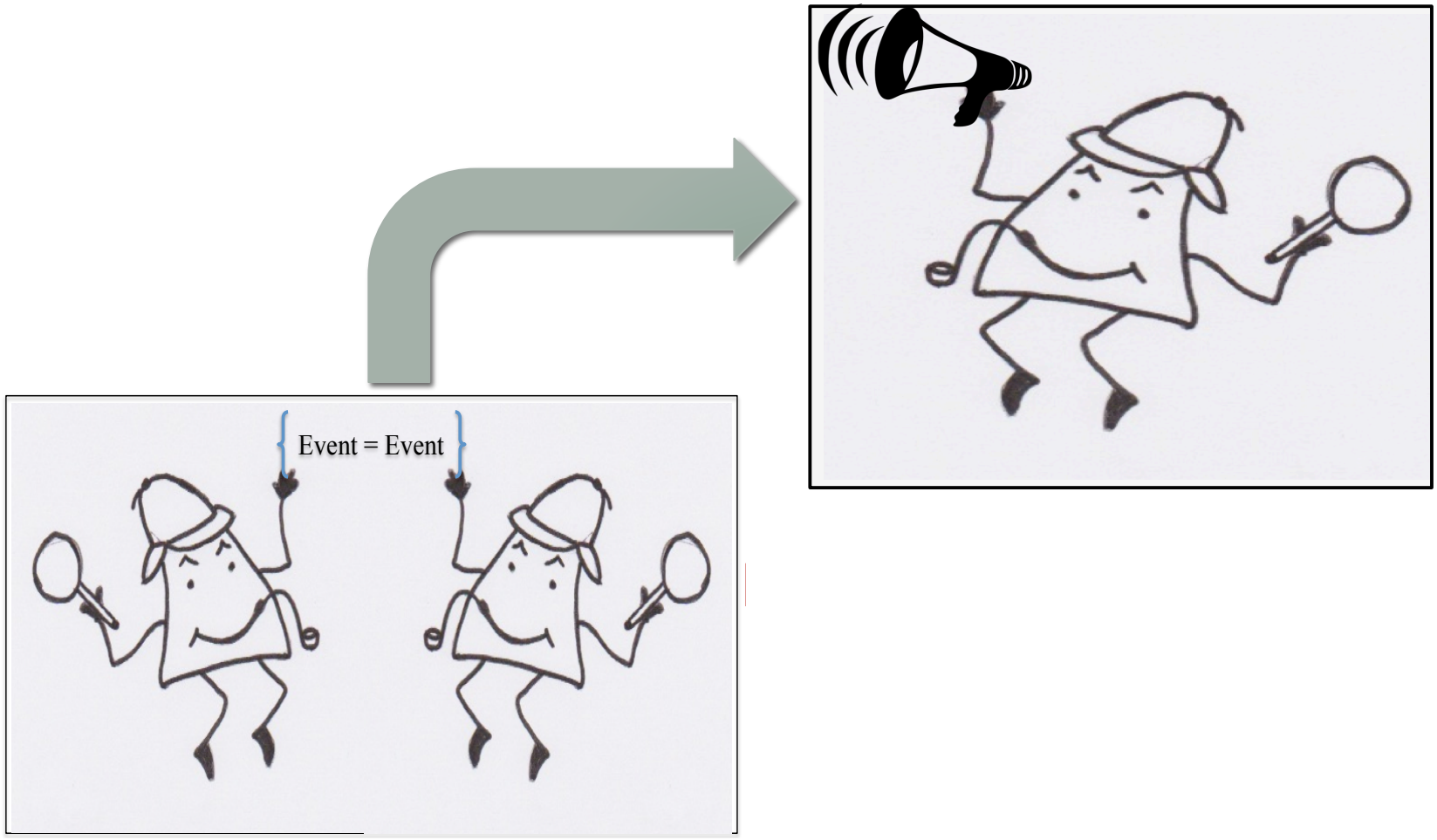
- Extract terms from class names, method signatures and identifiers
  - i.e. identifier *standAloneRisk* is split into *stand*, *alone* and *risk*.
- Store extracted terms in corpus and index repository.
- During a search compare query terms with those in corpus.
- Matching score is based on lexical similarity or probabilistic distance.
- Present a ranked list of resulting program elements i.e. classes



Our Approach

- Integrate lexical information retrieval with structural program dependency search.
- Present a ranked list of relevant classes for a change request (located in the top-*N* positions).
  1. Search for classes using the terms extracted from CR's.
  2. Assign a score to class based on where search terms occur i.e. on class name or in class body.
  3. For each class, adjust score based on the call-relations to the immediate neighbouring called classes (or calling).
  4. Sort result list using new score to group relevant classes together.

Lexical Similarity + Call Relations

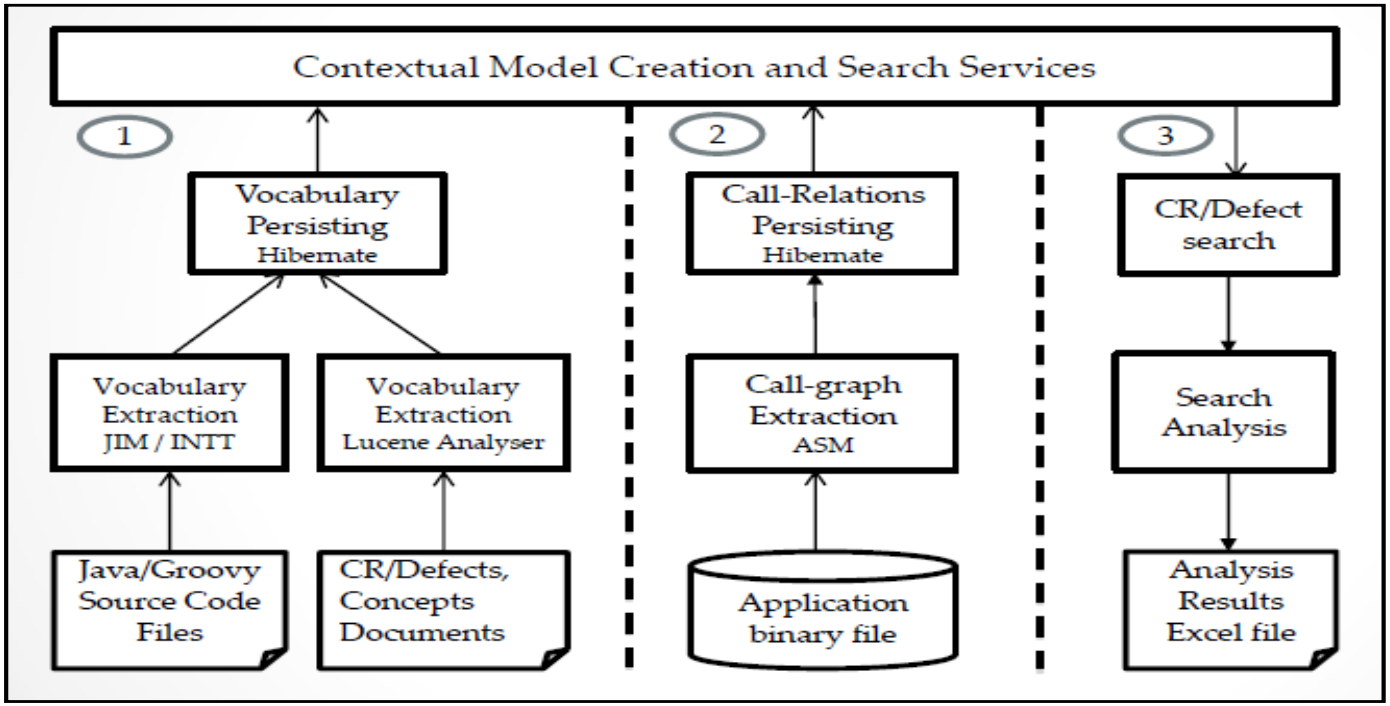


Example of Scoring

- Ranking with Lexical information and call-relations.

<buginformation>
<summary>Quota event limit needs to be selectable</summary>
<description>
By enhancing this feature we should allow the selection of specific events the limit is applied to.
</description>
</buginformation>

Data extraction, storage and search



Total Project Artefacts of our Case Study			
Applications	**Source files**	**CRs**	**Call relations**
6	24,915	3,517	94,864

AspectJ, Eclipse, Pillar-One, Pillar-Two, SWT, ZXing

Evaluation of Results

- RQ1: Lexical-only vs combined approach for *N* = 10
- Simple Lexical search approach
  - on average, less than 20% of CRs were located.
- CR descriptions
  - open source applications contain indications to class names
  - business oriented applications are very terse
- Our combined search approach
  - on average, more than 70% of CRs were located.

	Simple lexical match	**Lexical scoring + call relations**
Average	**19%**	**72%**

Evaluation of Results (cont.)

- RQ2: Performance of our tool compared to another tool.
  - ConCodeSe provides a 10 percentual point improvement over BugLocator

	Top-1		Top-10	
	Bug Locator	Con CodeSe	Bug Locator	Con CodeSe
Average	26%	35%	59%	69%

  - ConCodeSe locates more relevant classes in the top-N positions than BugLocator.

	Better		Same		Worse	
	top-1	top-10	top-1	top-10	top-1	top-10
Average	19%	30%	72%	54%	9%	16%

  - Achieved significant gains (i.e. AspectJ and Pillar-One applications)
    - MAP values improved almost 90% from 0.17 to 0.33 and from 0.18 to 0.26.

Discussion

- Supplementing lexical searches with call relationship data noticeably improve search accuracy.
- Enhancing lexical scoring with call-relations leads to superior search results.
- Provides entry points to the relevant classes within the context of the CR.
- CRs revealed two characteristics:
  - (1) developer nature with technical details i.e. references to code.
  - (2) descriptive nature with business terminology i.e. use of concept.

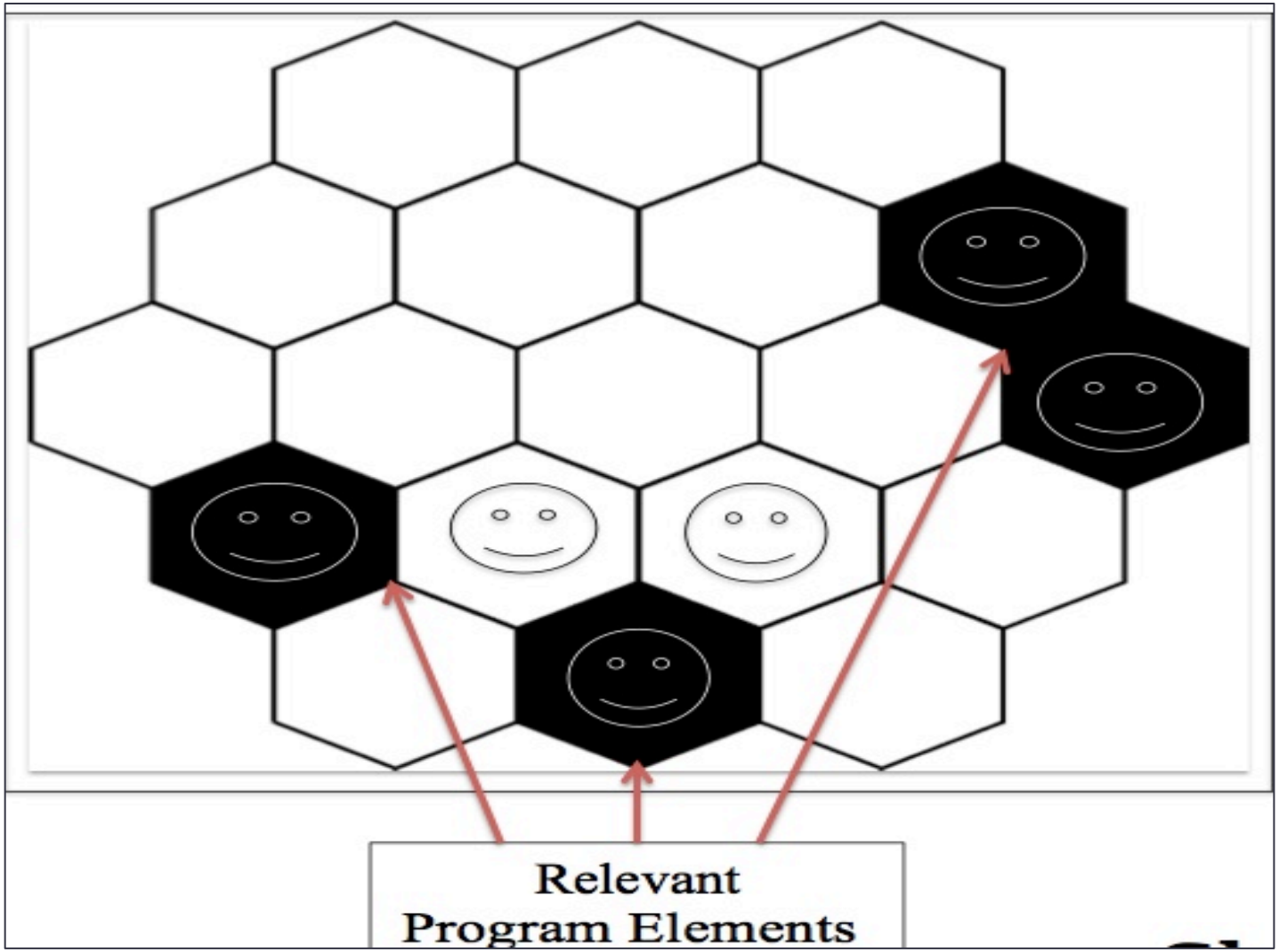
Discussion

- Multiple classes may implement a concept
  - on average only 3 classes listed as changed in a CR.
- Change Requests represent a unit-of-work
  - concept search in the context of CRs results in false positives.
- Conceptual responsibility v.s. technical functionality
  - mediate improved communication with business users.
  - finding relevant classes by considering the architecture.
- Domain vocabulary in change request descriptions
  - list of relevant domain vocabulary used in the code.
  - intelligently suggest the terms for selection when creating a CR.

Related Work

- On the use of call relations to recover traceability links, Hill *et al.* [1] proposed similar approach to ours.
  - a query is created with search terms,
  - source code is searched for matching methods by using LSI method.
  - a term/document frequency (tf/idf) score is obtained.
  - call-graph is utilized to evaluate relevance of neighbouring methods.
- Petrenko *et al.* [2] presented a technique called DepIR that combines Dependency Search (DepS) with IR.
  - query to obtain a ranked list of methods retrieved by probabilistic LSI.
  - 10 methods with highest ranks are selected as entry points.
  - explore the shortest path on the call-graph to relevant method.

Contextual Model



Concluding Remarks

- An algorithm using lexical and structural information suggests, in a ranked order, classes to be changed.
- Vastly superior to simple string search, as performed by developers using an IDE.
- Compared to an existing approach,
  - improved ranking of affected classes.
  - increased percentage of CRs for which relevant classes are retrieved.
  - enhanced number of relevant classes suggested among the top-1 or -10.

Challenges

- (1) How do I determine the relevance of a class for a CR?
  - List of classes that has the search terms.
  - List of classes that does not have the search terms and still connected with those from above list.
  - Are they still relevant within the context of a CR?

- (2)How do I enrich the CR vocabulary to create a context?
  - Cluster of terms based on term frequency in source code and CR.
  - Use structural relations i.e. package names or inheritance.

Challenges

- (1) How to determine the relevance of a class for a CR?

<summary>Percentile/VaR/TVaR cannot be displayed second time</summary>
<description>How to reproduce: run any parametrization → open results → show e.g. 99.5% PROFIT VaR
→ hide column again → try to show 99.5% PROFIT VaR again → ERROR nothing happens
</description>
<file>org.pillarone.riskanalytics.application.dataaccess.function.AbstractFunction.java</file>
<file>org.pillarone.riskanalytics.application.dataaccess.function.AbstractQuantilePerspectiveBasedFunction.java</file>
<file>org.pillarone.riskanalytics.application.dataaccess.function.FunctionDescriptor.java</file>
<file>org.pillarone.riskanalytics.application.dataaccess.function.IFunction.java</file>
<file>org.pillarone.riskanalytics.application.dataaccess.function.ParametrizedFunctionDescriptor.java</file>
<file>org.pillarone.riskanalytics.application.dataaccess.function.QuantileBasedFunctionDescriptor.java</file>
<file>org.pillarone.riskanalytics.application.dataaccess.function.SingleIterationFunction.java</file>
<file>org.pillarone.riskanalytics.application.ui.result.action.keyfigure.ParametrizedKeyFigureAction.java</file>
<file>org.pillarone.riskanalytics.application.ui.result.action.keyfigure.QuantilePerspectiveBasedKeyFigureAction.java</file>
<file>org.pillarone.riskanalytics.application.ui.result.model.ResultViewModel.java</file>

TvarFunction	0.075	tvar
VarFunction	0.0375	var
PercentileFunction	0.0375	percentile
ProfitCommission	0.0375	profit

?

Challenges

- (2) How to enrich CR vocabulary to create a context?
  - Looking at some CR descriptions and source code of the affected classes revealed some n-term usage like
    - "poisson distribution"
    - "result template"
    - "simulation page"
    - "save dialog”
  - Also there seems to be a usage mapping between the terms like,
    - Show ---> perspective
    - Second time ---> iteration
    - CVS export ---> exportCVS
    - Percentile ---> quantile

Future work

- Challenging to find the classes referred by a CR:
  - in spite our improvements, over 40% of CRs may not be localised.
- Results emphasize combined tool support needed.
- Further enhance our algorithm by considering domain concept relations.
- An application base with well documented repository like Prepaid for further research by considering additional artefacts like git commits history.

Thank you

- Contact me per email:
  - tezcan@dilshener.de

- My Publications:
  - http://crc.open.ac.uk/People/Tezcan_Dilshener_41C5

- References

[1] Hill, E., Pollock, L., and Vijay-Shanker, K., "Exploring the Neighborhood with Dora to Expedite Software Maintenance", in Proc. 22nd Int'l Conf. on Automated Software Engineering, 2007, pp. 14-23

[3] M. Petrenko and V. Rajlich, "Concept location using program dependencies and information retrieval (DepIR)," Information and Software Technology, vol. 55, no. 4, pp. 651–659, 2013.