UI MAYNOOTH

Ollscoil na hÉireann Má Nuad

Erasmus Mundus MSc in Dependable Software Systems

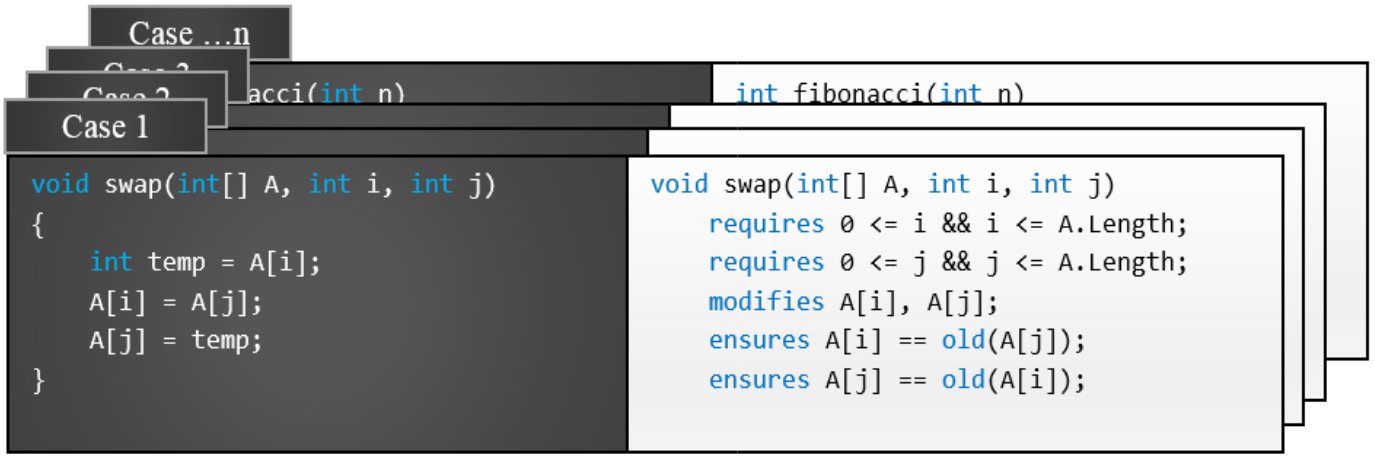
Source Code Retrieval using Case Based Reasoning

MIHAI PITU

SUPERVISORS: DR. DIARMUID O'DONOGHUE & DR. ROSEMARY MONAHAN

Introduction

- MSc Dissertation Project
- Automate some of the steps involved in writing formal specifications, by reusing existing verified programs
- Case-Based Reasoning approach, with a strong focus on source code retrieval



Motivation

Formal specification generation benefits:

- Reusability of implementations and specifications
- The software engineer does not need to learn the specification language, the transfer is done automatically
- Increase the number of verified implementation (Verified Software Repository)

Motivation

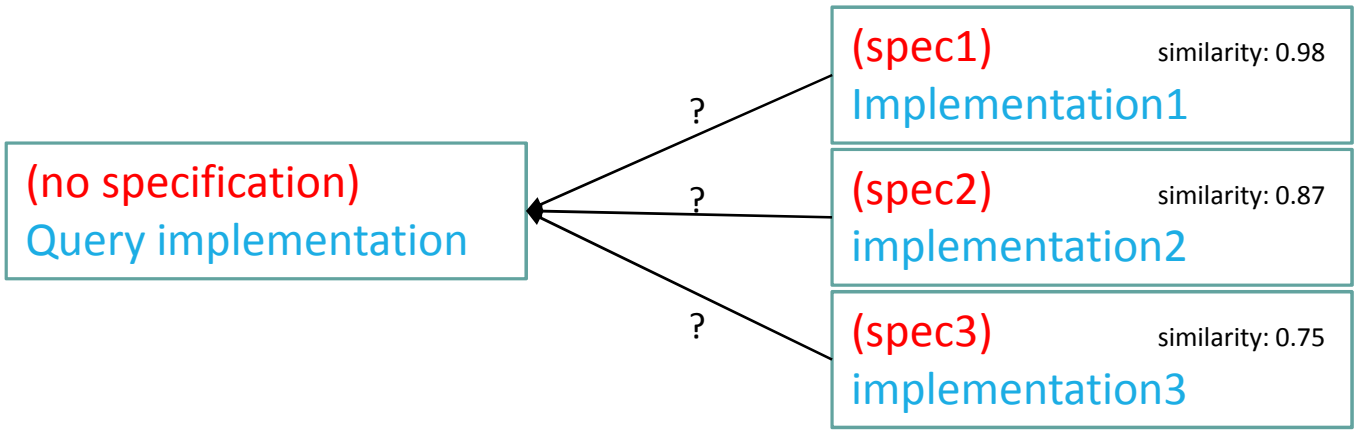
Source code retrieval benefits:

- Code theft detection
- Improve understanding of source code
- Knowledge acquisition by comparing different implementations
- Rapid prototyping
- Programming by example

Background

Case Based Reasoning and *Analogical Reasoning* methodology in *Arís*:

- **Retrieve** the most similar past solutions
- **Find the correspondences** that exists between new and old problems
- **Transfer** the old solution to the new problem
- **Retain** the solution for use in future problems



Related Work

Intensive research in **source code retrieval** in the past few years:

- "Component Retrieval Using Conversational Case Based Reasoning", M. Gu, A. Aamodt and X. Tong, 2004
- "Source Code Retrieval using Conceptual Similarity", G. Mishne, M. Rijke, 2004
- "Detecting similar software applications", C. McMillan, M. Grechanik, D. Poshyvanyk, 2012

A few publications related to **reusability of specifications**:

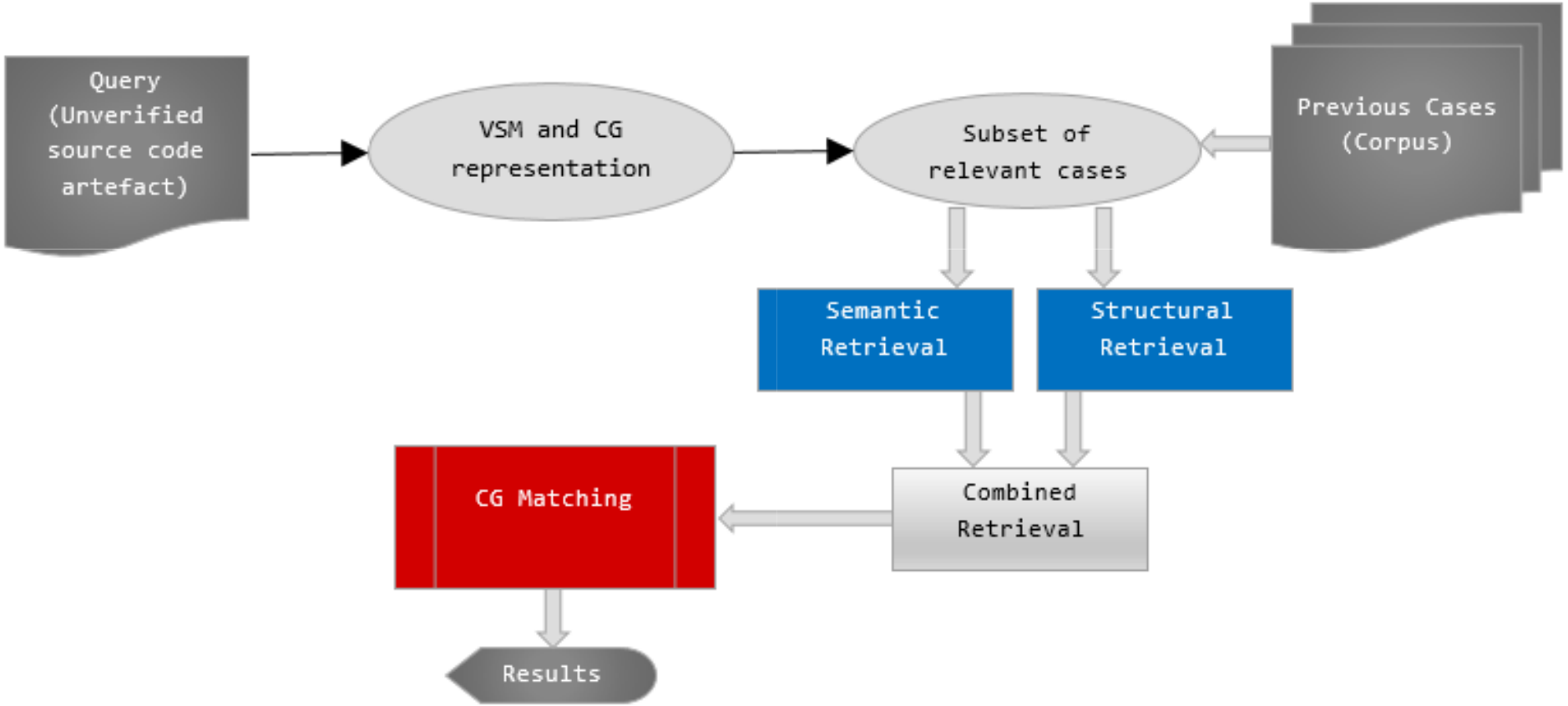
- "A two-stage framework for UML specification matching", W.J. Park, D.H. Bae, 2006
- "Towards an ontology-based retrieval of UML Class Diagrams", K. Robles, A. Fraga, J. Morato, J. Llorens, 2012

Retrieval system – Proposed approach

Source Code Retrieval: Our approach

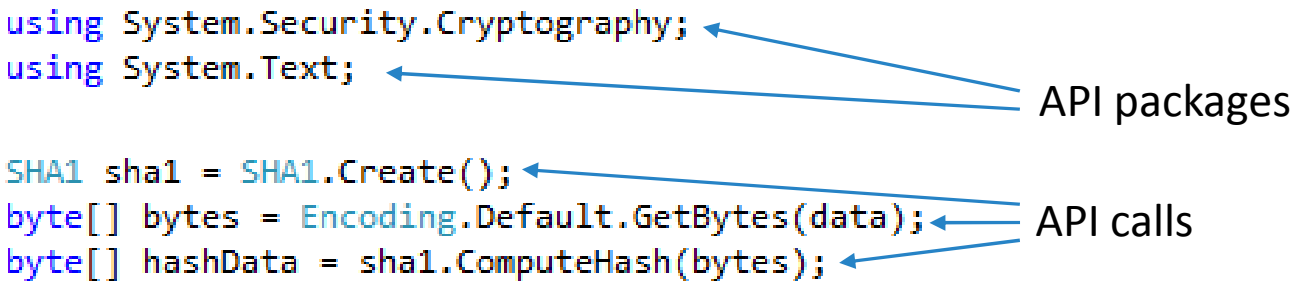
- Store the implementations and specifications as **cases** (various granularity: methods, classes, applications) – *proof of concept: C# implementations & Spec# specifications*
- **Retrieve** similar source code artefacts using semantic and structural characteristics:
  - *Semantic*: Use API calls to express the meaning of code
  - *Structural*: Explore graph representation of source code
- Select **most relevant** cases similar to the input implementation and **transfer** the specification
- **Retain** solution for future use

Source Code Retrieval: Our approach



Semantic Retrieval

- *API calls* from well-known and widely used libraries have **precisely defined semantics** unlike names of program variables, types and words that programmers use in comments
- Previous systems used API calls as semantic anchors and organized the repository in several ways (matrix, ontology)
- Our framework uses *Vector Space Model* as a main technique for representing documents, indexing API calls and computing similarity scores



Semantic Retrieval - VSM

In VSM, documents (source code files, text files, etc.) are represented as vectors of real numbers, where dimensions correspond to terms (words, API calls, etc.):

	$Object.ToString()$	$Integer.Parse()$	$List.Add()$	$List.Remove()$	$String.Replace()$	$String.IndexOf()$
$method_1 =$	0	2	1	1	1	0
$class_2 =$	4	1	0	0	0	1

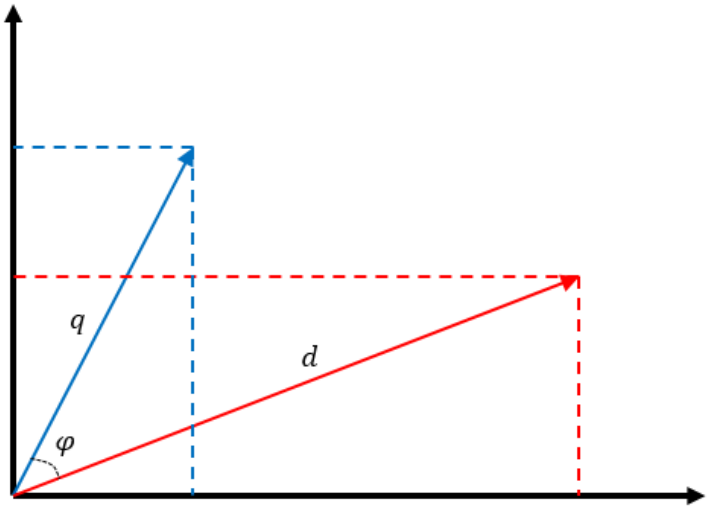
- In practice, storing these sparse vectors might be unfeasible, other techniques can be used (e.g. inverted indexing, where each API calls has a list of usage documents)

- Each element represents the weight of the term in that document (term frequency, tf-idf)
  - $tf(t,d) =$ number of occurences of t in d
  - $idf(t,D) = \log \frac{\#D}{1+\#\{d \in D : t \in d\}}$
  - $tf - idf(t,d,D) = tf(t,d) \times idf(t,D)$

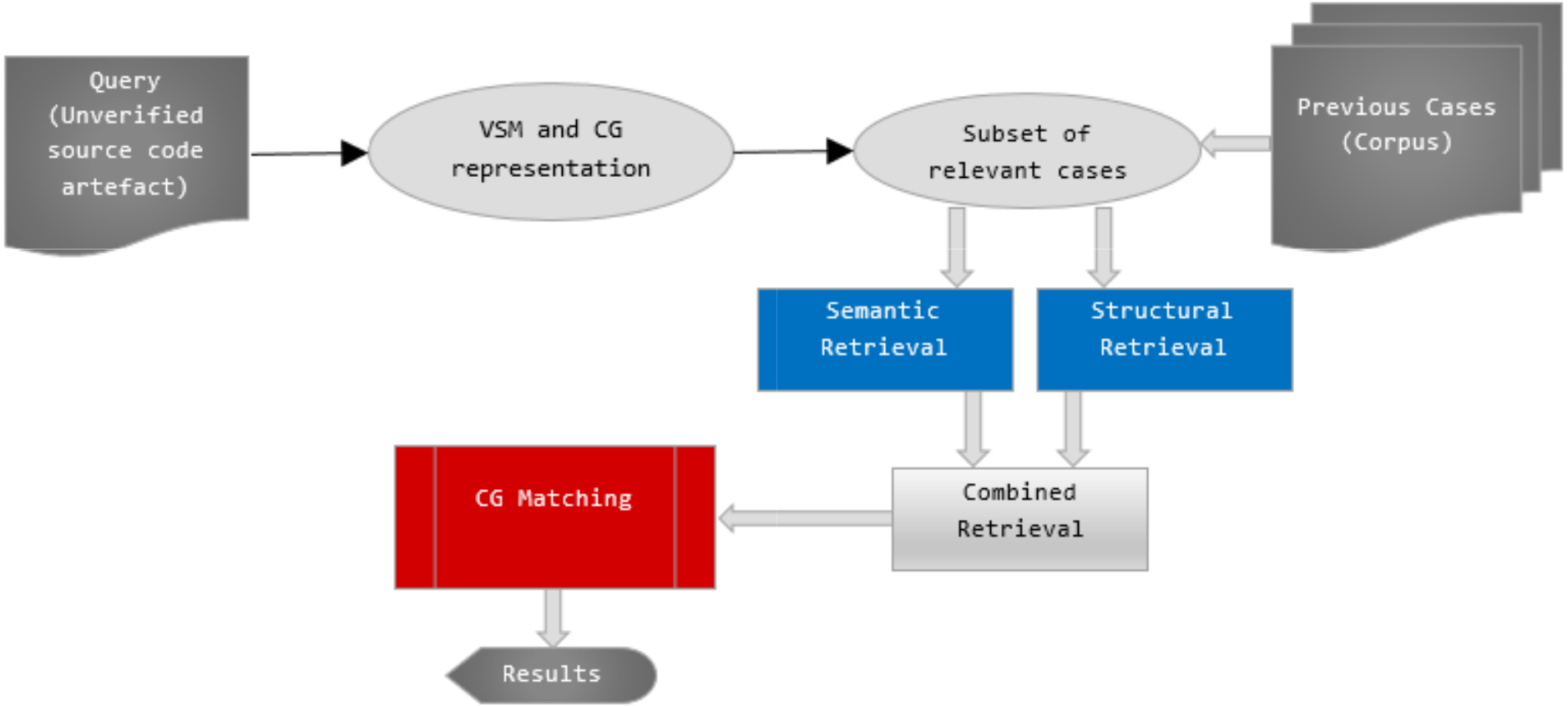
Semantic Retrieval - VSM

- The angle between vector space representations can be used to measure the similarity between documents
  - $simCos(q,d) = \cos(\varphi) = \frac{q*d}{||q||\,||d||}$

- Order of API calls encoded with Levenshtein distance



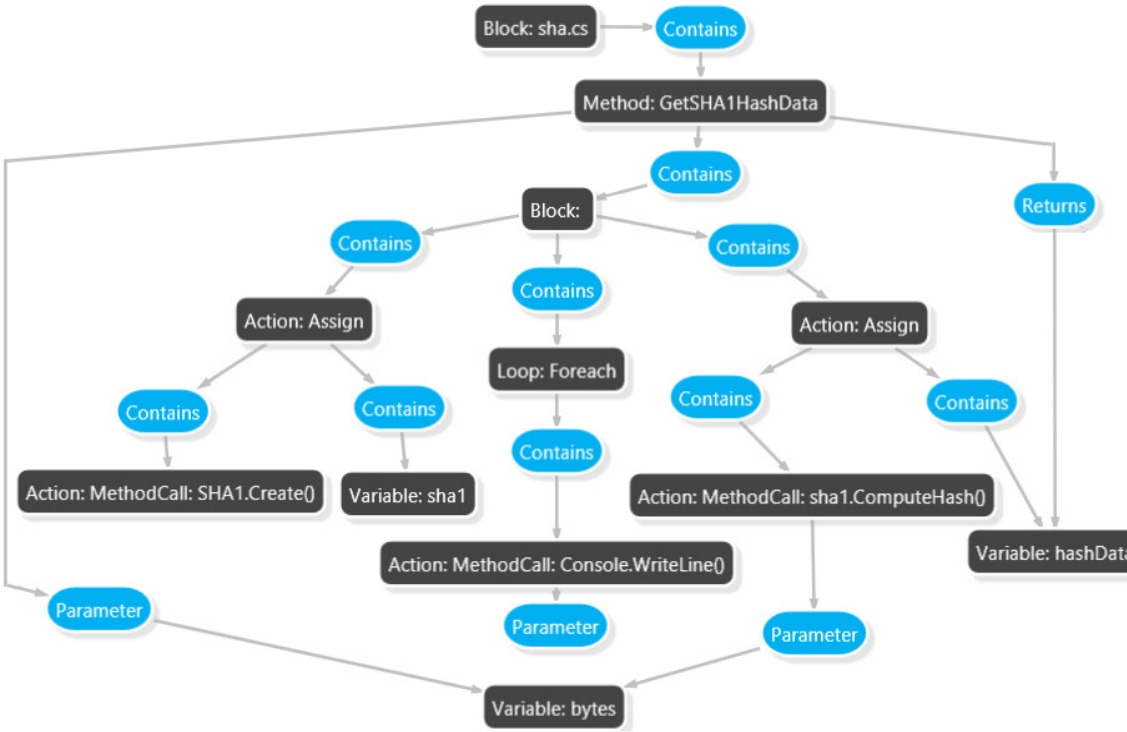
Source Code Retrieval: Our approach



Structural Retrieval

- Analyze structural and topological characteristics of source code

```
public byte[] GetSHA1HashData(byte[] bytes)
{
    var sha1 = SHA1.Create();
    byte[] hashData = sha1.ComputeHash(bytes);
    foreach (byte b in hashData)
        Console.WriteLine(b);
    return hashData;
}
```



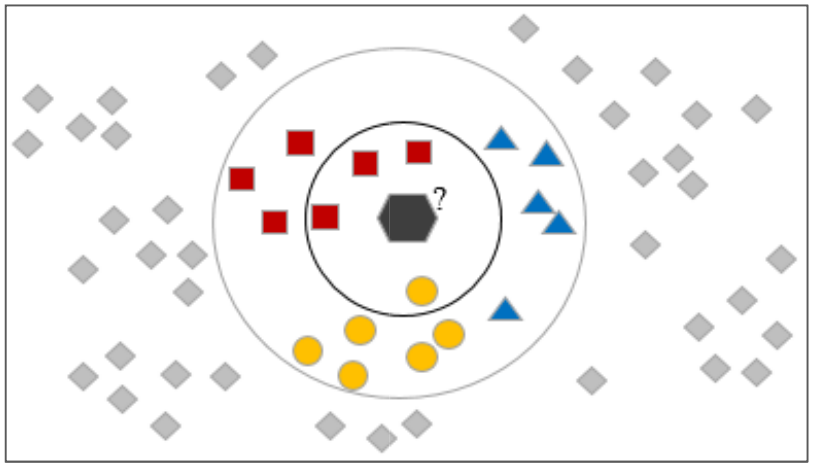
Structural Retrieval

- We use *Content Vectors* as representational structures, which encode various information extracted from *Conceptual Graphs* G(V, E):
- *Graph-Based* metrics:
  - *Vertex count(#V)*
  - *Edge count(#E)*
  - *Average degree*
  - *Graph diameter*
  - *Maximum out degree*
  - *Maximum in degree*
  - *Average node rank*

Concept Type	Summary
AssignOp	An assignment of a value to a field or variable
Block	Set of concepts that are logically grouped together
Class	Declaration or definition of a class
CompareOp	Binary comparison (e.g. >=, ==, !=, etc.)
Enum	Declaration of an enumerated set of values
Field	Declaration of a variable in a class
If	A conditional branching statement
LogicalOp	Binary logical operation (e.g. &&, ||, etc.)
Loop	Iterative process that depends on a condition
MathOp	Mathematical operation (e.g. *,+,-, etc.)
Method	Declaration or definition of a function part of a class
Method-Call	Method invocation
Null	Null reference
String	Constant textual entity
Switch	Conditional statement that has multiple branches
Try-Catch	Try block followed by catch clauses
Variable	Entity declared in an implementation that holds values during execution

Structural Retrieval

- The Case-Base can be very big, therefore we need to compare the *query* only with relevant source code artefacts
- Solution: Content Vectors are clustered with *K-means* and perform *K-Nearest Neighbours* only on the closest sub-groups
- Similarity between vectors computed using *Radial Basis Function:*
  - $RBF(q,d) = sim_{structural}(q,d) = \exp(-\gamma||q-d||^2)$



Combined Retrieval

- Structural and semantic retrieval results are combined and the top ranked source code artefacts are further analysed by the *Conceptual Graph-Matching* module in *Arís*
  - $sim_{comb}(q,d) = w_{semantic} \times sim_{semantic}(q,d) + w_{structural} \times sim_{structural}(q,d)$

where $w_{semantic}$ and $w_{structural}$ are weights for semantic and structural retrieval, chosen experimentally

	Semantic retrieval results		Structural retrieval results		Combined retrieval results
method1	0.92	+	0.95	=	0.935
method2	0.89	+	0.86	=	0.875
method3	0.64	+	0.64	=	0.64
method4	0.39	+	0.0	=	0.195
method5	0.0	+	0.56	=	0.280

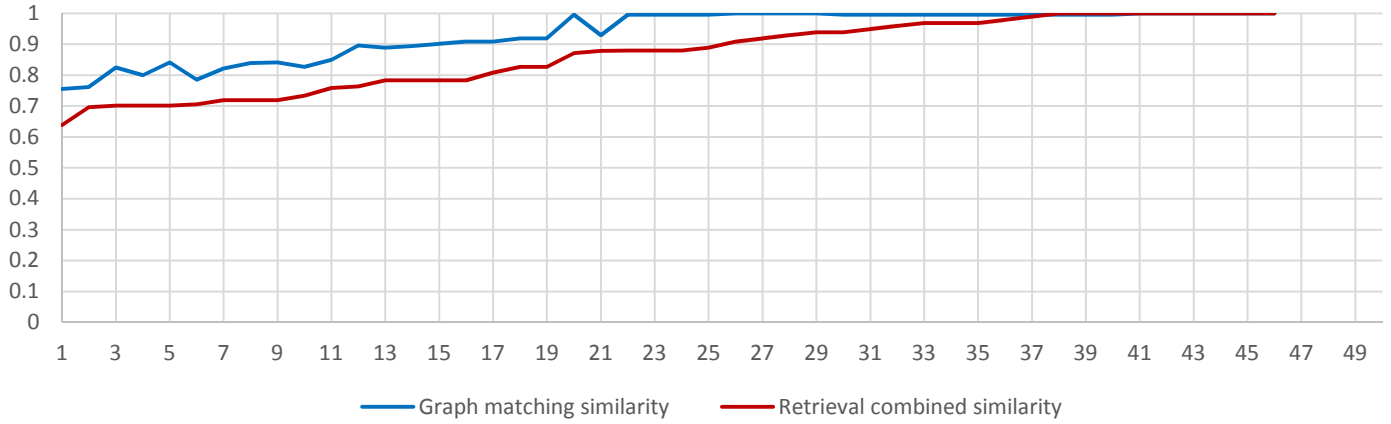
Evaluation

Evaluation – Case-Base

- For evaluation purposes, we selected real-world and Spec# test-suite software artefacts, containing 175,291 classes and 2,033,623 methods (a total of approx. 74 million lines of code) to aggregate the Case-base
- *Arís* retrieval response time is on average 2.89 seconds
- In our experiments, the query set contains 50 modified case-base examples, that do not change the original code functionality
  - *Lexical modifications*
  - *Method parameter reorder*
  - *Code style changes*
  - *Order of statements*
  - *Extraneous statements*
  - *Changing variable types*

Evaluation – Parameter selection

- We tested our retrieval system with a number of different configurations in order to select the best parameters
- The main conclusion of our parameter selection experiments was that using the Conceptual Graph-Matching similarity improves performance
- We proved retrieval quality using statistical significance tests (Mann-Whitney & Kruskal–Wallis)



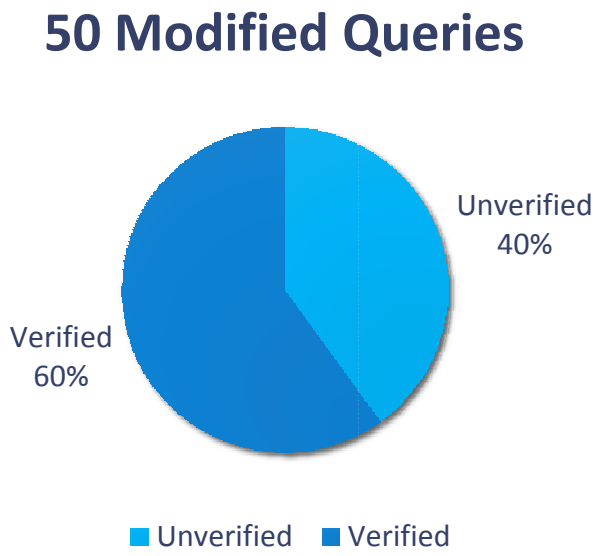
Evaluation – Retrieval measures

- Although on a different query set, we measured *Precision* and *Mean Reciprocal Rank* for comparison with existing source code retrieval systems
  - $Precision = \frac{|Relevant \cap Retrieved|}{|Retrieved|}$
  - $MRR(Q) = avg\left\{\frac{1}{rank(d(q_1))}, \ldots, \frac{1}{rank(d(q_n))}\right\}$

SC Retrieval Systems	Precision	MRR
SC Retrieval in Arís	0.442	0.908
CG Retrieval (Mishne & De Rijke, 2004)	0.352	0.813
CLAN (McMillan, et al., 2012)	0.45	(n/a)

Evaluation – Specification generation

- If the query is identical with an existing verified document in the Case-Base, mapping and verification in *Arís* is **guaranteed** to be successful
- In specification generation, *Arís* was able to successfully verify **60%** of the modified queries provided, using retrieved source code artefacts



Demo

Conclusions

- We presented a novel approach to the problem of writing/generating formal specifications using the Case-Based Reasoning methodology
- The source code retrieval module can be used independently from *Arís*
- This project represents promising work towards reuse of formal specifications