Semantic Code Clone Detection for Enterprise Applications

Jan Svacina and Jonathan Simmons

Motivation

- Enterprise software maintenance and costs
    - Corporations pay 20% - 25% of their total fees and costs for maintenance alone [2, 4]
    - Oracle reported in fiscal year spanning 2013 to 2014 to have earned 2.7 times license revenue in maintenance and support requests [4]
- Quality Assurance
    - Plagiarism (least of concerns)
    - Enterprise architecture design patterns / best practice [5]
        - Code readability
- Impact and Implication of code clones
    - 10% to 23% of large software projects are composed of duplicate code [3, 6]
    - Reducing clones can reduce code bloat / bug fix time [7]
    - Increased complexity has a proportional connection to increased maintenance costs [1]

Abstract Summary

Code clone detection is a popular problem within computer science, however, it is also a valuable tool for deriving metrics and **reducing code complexity** within applications.

It is evident that code clone detection tools as they are now do not handle the scope of enterprise solutions well at all, thus, we propose a method of finding code clones within enterprise applications focusing on **semantic meanings within enterprise frameworks.**

We used inter-procedure control flow graphs to model an enterprise application and demonstrated a solution to this problem with **a time complexity of $O(n^2)$.**

Hypothesis: Detect semantic code clones in enterprise application using semantic meaning of the program.

Terminology - Program analysis

Program analysis:

1. get information from the code and executions
2. reason about the code and executions
3. determine program properties (what facts and constraints it holds)

Program analysis tools: software analysis other software (like compiler)

Terminology - Program analysis cont.

Types of program analysis:

- **Static Analysis: source code**
- Dynamic Analysis: executions (bytecode analysis, isolated execution, etc.)

Types of Static Analysis

- Control flow: execution order of the statements (e.g. parse tree)
- Dataflow: program variable usage pattern
- **Interprocedural control flow: Get information through all procedures**

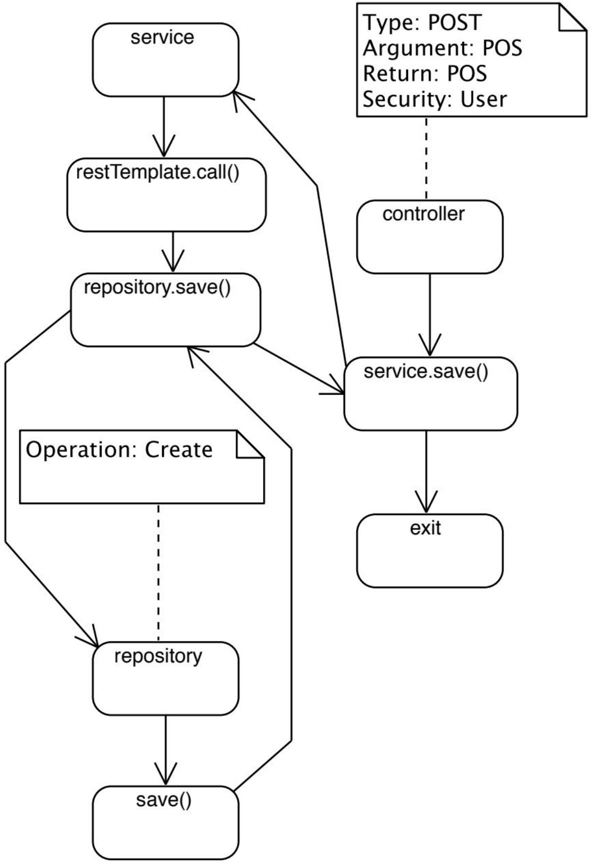
Static analysis - Interprocedural Control Flow Graph

Program Representation

A control flow graph (CFG) of a procedure is a graph G = (N, E), where the nodes in N represent statements of the procedure and the edges in E represent the transfer of the control between two statements.

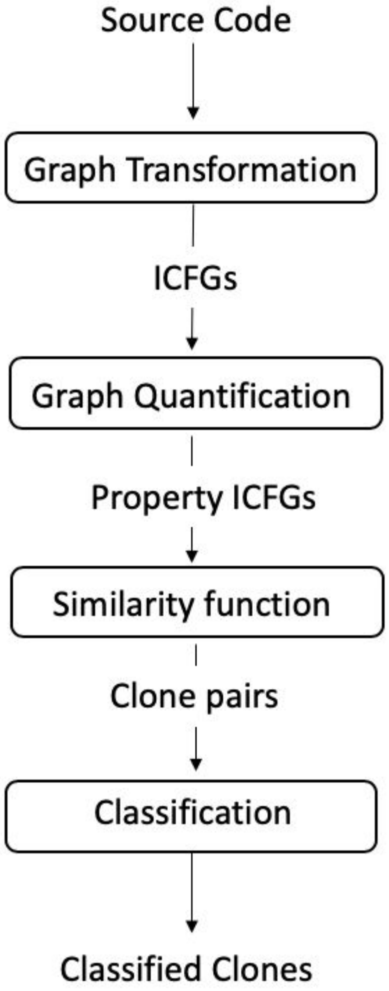
An interprocedural control flow graph (ICFG) of a program is a collection of GFGs such that each CFG represents a procedure in the program.

```
@Controller
public class PosController{
    @PreAuthorize("hasRole('USER')")
    @RequestMapping(value = "/pos", method = RequestMethod.POST)
    public POS create (@RequestBody Pos pos) {
        return service.save(pos);
    }
}


@Service
public class PosService {
    public Pos savePos(Pos pos){
        Properties p = restTemplate.postObject("/properties");
        pos.setProperties(p);
        return repository.save(pos);
    }
}


@Repository
public inteface PosRepository{
    public interface PosRepository{
        Pos save(Pos pos)
    }
}
```

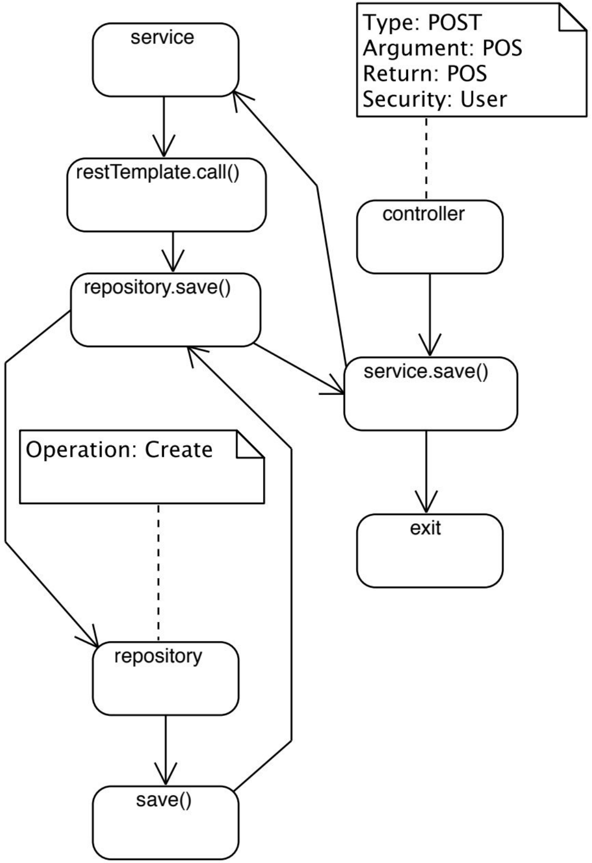


Proposed Method

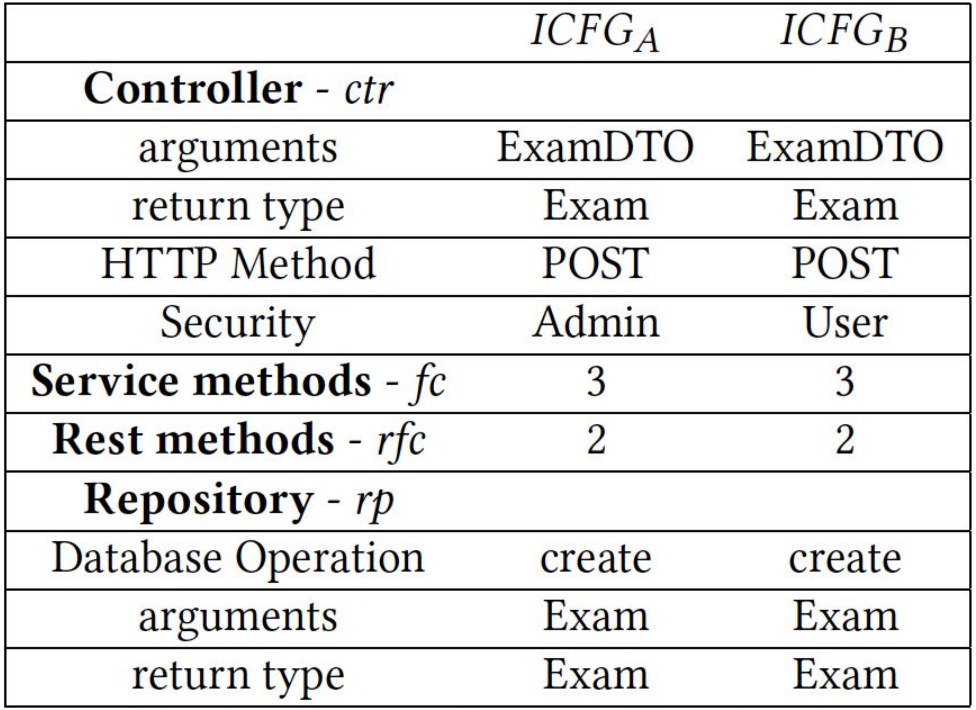
1. Source code to CFGs
2. Add properties to CFGs
3. Similarity function: global and local
4. Classification: strongly, weakly related



```
@Controller
public class PosController{
    @PreAuthorize("hasRole('USER')")
    @RequestMapping(value = "/pos", method = RequestMethod.POST)
    public POS create (@RequestBody Pos pos) {
        return service.save(pos);
    }
}


@Service
public class PosService {
    public Pos savePos(Pos pos){
        Properties p = restTemplate.postObject("/properties");
        pos.setProperties(p);
        return repository.save(pos);
    }
}


@Repository
public inteface PosRepository{
    public interface PosRepository{
        Pos save(Pos pos)
    }
}
```



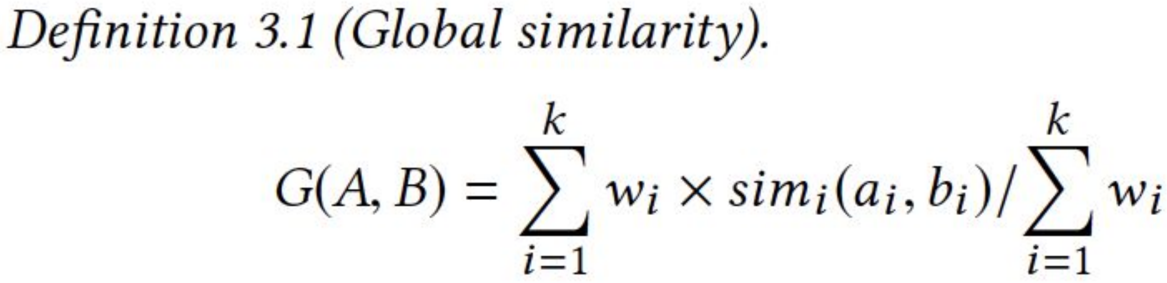
Semantic meaning of ICFG



	$ICFG_A$	$ICFG_B$
Controller - *ctr*		
arguments	ExamDTO	ExamDTO
return type	Exam	Exam
HTTP Method	POST	POST
Security	Admin	User
Service methods - *fc*	3	3
Rest methods - *rfc*	2	2
Repository - *rp*		
Database Operation	create	create
arguments	Exam	Exam
return type	Exam	Exam

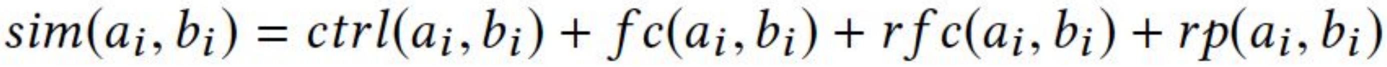
Table 3: Example of properties of 2 ICFGs

Similarity functions



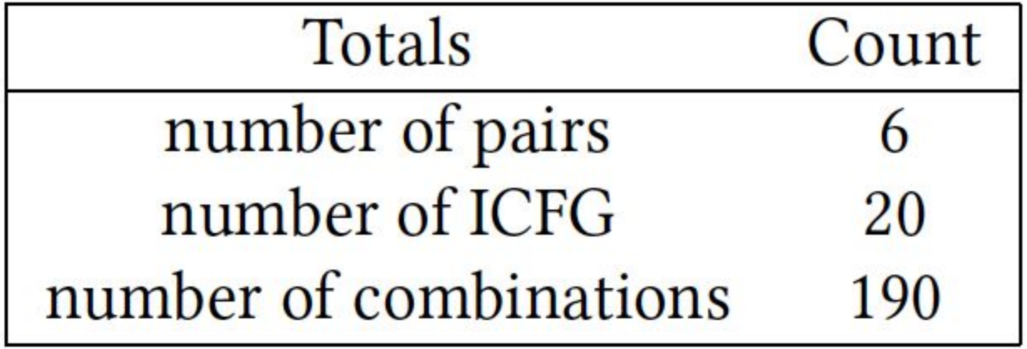
Definition 3.1 (Global similarity).

$$G(A, B) = \sum_{i=1}^{k} w_i \times sim_i(a_i, b_i) / \sum_{i=1}^{k} w_i$$



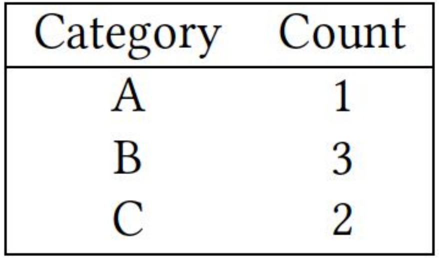
$$sim(a_i, b_i) = ctrl(a_i, b_i) + fc(a_i, b_i) + rfc(a_i, b_i) + rp(a_i, b_i)$$

Case study



Totals	Count
number of pairs	6
number of ICFG	20
number of combinations	190

Table 5: Results quantification



Category	Count
A	1
B	3
C	2

Table 6: Categorizing found clones

Future work

- Heuristic approach towards code preprocessing
- Which parts of a codebase are worth analyzing
- More metrics to report
    - E.g. procedural entropy via git commit progression
    - Class stereotype behavior index

Many possibilities thanks to our extensibility.

Presentation References

[1] Rajiv D. Banker, Srikant M. Datar, Chris F. Kemerer, and Dani Zweig. 1993. Software complexity and maintenance costs.

[2] Chris Doig. 2015. Calculating the total cost of ownership for enterprise software.

[3] Cory Kapser and Michael W. Godfrey. 2003. *Toward a Taxonomy of Clones in Source Code: A Case Study.*

[4] Michael Krigsman. [n. d.]. Danger zone: Enterprise maintenance and support.

[5] Matthew Foemmel Edward Hieatt Robert Mee Randy Stafford Martin Fowler, David Rice. 2002. *Patterns of Enterprise Application Architecture*. Addison Wesley.

[6] Chanchal K. Roy, James R. Cordy, and Rainer Koschke. 2009. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming* 74, 7 (May 2009), 470–495. https://doi.org/10.1016/j.scico.2009.02.007

Presentation References cont.

[7] Neha Saini, Sukhdip Singh, and Suman. 2018. Code Clones: Detection and Management. *Procedia Computer Science* 132 (Jan. 2018), 718–727. https://doi.org/10.1016/j.procs.2018.05.080