Project ID : 18-055

PROGRESS PRESENTATION 02

GROUP MEMBERS

SUPERVISOR

Mr. Dilshan de Silva

P.K.H Palihakkara
IT15113900

Gamaarachchi G.A.C.Y
IT15111548

K.G.D.R Perera
IT15112538

M.A.N.S.U.K. Uvindasiri
IT15413802

PROGRAM ANALYSIS TOOL

ProAnalýza

OUTLINE

- BACKGROUND
- LITERATURE
- RESEARCH PROBLEM & OBJECTIVES
- POSSIBLE SOLUTION
- SYSTEM DIAGRAM
- METHODOLOGY
- DEVELOPED SOLUTION
- RESULTS & DISCUSSION
- CONCLUSION

BACKGROUND

IMPORTANCE OF MAINTAINING SOURCE CODE QUALITY

- Simply delivering functionality is not enough
- It’s crucial that developers pay attention to quality attributes
- Furious rate of product development
- Software updates at multiple times
- Testing must be done every time system changes
- Otherwise high cost and effort to test and maintain system

WHY DO WE NEED TO ANALYSE SOURCE CODE ?

DEVELOPMENT EFFORT

DEVELOPMENT TIME

NUMBER OF PEOPLE INVOLVED

COST

BY ANALYSING SOURCE CODE

Judge	Compare	Predict	Put
Judge how large your program is	Compare it with other programs	Predict the effort needed to maintain or rewrite your code	Put a price tag on your code

LITERATURE

PROGRAM ANALYSIS TECHNIQUES

STATICS ANALYSIS

DYNAMIC ANALYSIS

WHY WE CHOSE STATIC ANALYSIS ?

Can discover vulnerabilities during the development phase of the program.

These vulnerabilities are easier to correct than the ones found during the testing phase since static analysis leads to the root of the vulnerability.

“24% of all software projects fail, which means they are cancelled prior to completion or delivered and never used, while only 32% can be considered as successful”.

- The Standish Group Report CHAOS 2009

MAJOR REASONS FOR FAILURE

- High complexity
- Difficult extensibility
- Poor maintainability

of source code

EXISTING SYSTEMS & THEIR LIMITATIONS

Values of software metric, were calculated by the various program analysis tools, are different due to unclear definition of metrics, errors in calculation of metrics and different preprocessing steps used by them.

This difference among the results makes the program analysis tools unreliable. Therefore, the decision making under uncertainty.

	sonarqube	JArchitect	FindBugs	SourceMeter
Low Memory Usage	★		★	
Static and Dynamic code analysis	★	★		
Affordable			★	
Accurate				★
Comprehensive rules				★
Quality checking	★	★		★
View program structure		★		
Visual Representation	★	★		★
Rate software developers based on their code quality				

RESEARCH PROBLEM & OBJECTIVES

There’s no standard procedure to evaluate Software developers by measuring their source code quality.

The existing program analysis tools are unreliable & even unaffordable.

Hence there’s a requirement for a program analysis tool which evaluate & calculate metrics accurately. Thereby rank each developer working in a certain project.

Static program analysis is more precise in discovering vulnerabilities during the development phase of the program.

OBJECTIVES

- To develop a software metric to evaluate java source code.
- To develop a program analysis tool which allows to implement proposed software metric and rate the performance of software developers.

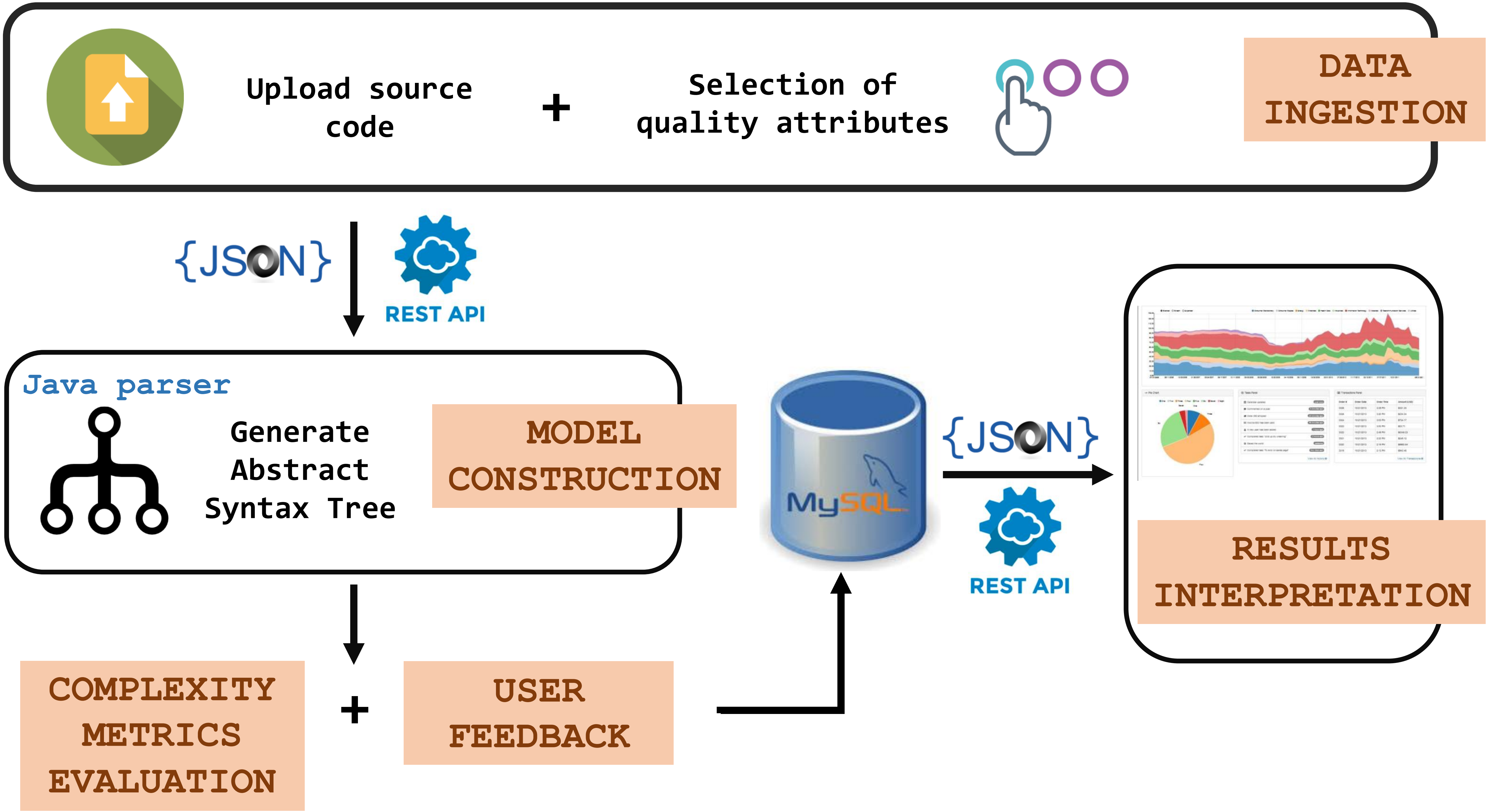
POSSIBLE SOLUTION

“AN AFFORDABLE & RELIABLE PROGRAM ANALYSIS TOOL”

UNIQUENESS OVER OTHER EXISTING SYSTEMS

- Can be used to analyze source code written in Java language
- Well defined software metrics evaluation & calculation
- Low memory usage
- Affordable
- High accuracy in determining code quality
- View program structure
- Visual interpretation of results
- Rate software developers based on their code quality

SYSTEM DIAGRAM


Upload source code
+
Selection of quality attributes
DATA INGESTION
{JSON}
REST API
Java parser
Generate Abstract Syntax Tree
MODEL CONSTRUCTION
MySQL
{JSON}
REST API
RESULTS INTERPRETATION
COMPLEXITY METRICS EVALUATION
+
USER FEEDBACK

METHODOLOGY

STEPS FOLLOWED

Data collection through surveys conducted undergraduate and industry level, literature review

Analysis of data and information

Prediction of a formula for complexity metrics based on predefined factors

Program Analysis Tool implementation

TOOLS

MySQL®

POSTMAN

eclipse

TECHNOLOGIES

node JS

Java

ANGULARJS

HTML

CSS

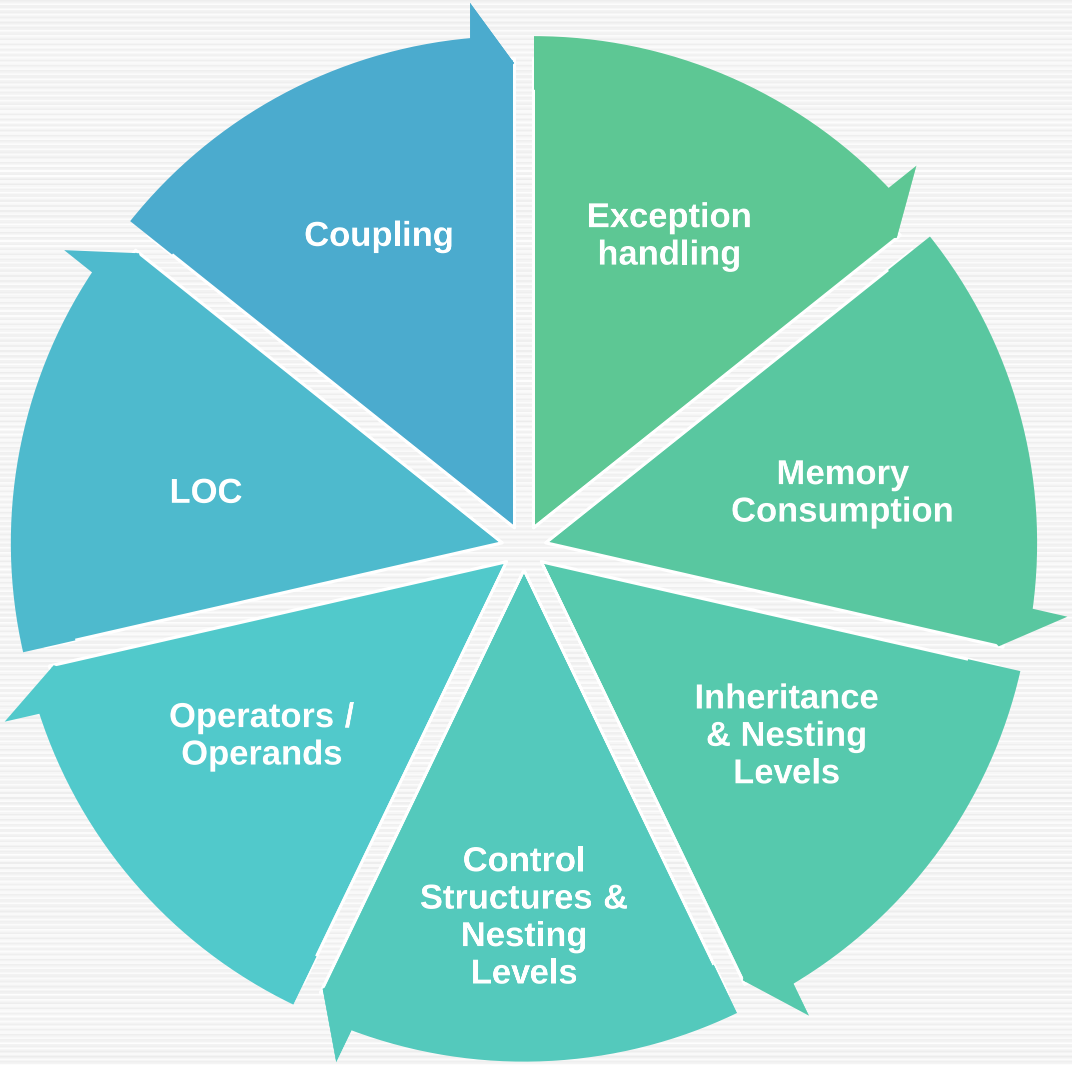
JS

SQL

DEVELOPED SOLUTION

COMPLEXITY METRIC EVALUATION

- FACTORS UNDER CONSIDERATION



PROPOSED METRIC

- IWCA = Weight of Inheritance in class A (Weight due to depth of class in inheritance hierarchy)
- CCA = Complexity of class A
- TW = Weight due to type of control structure
- NW = Weight d
- ue to nesting level of control structure
- OL = Weight due to method overloading
- NOOL = Number of overloading methods
- OR = Weight due to method overriding
- NOOR = Number of overriding methods
- VW = Weight due to variable types
- NOV = Number of variables
- CW = Weight due to types of coupling
- MCT = Weight due to type of method call
- BE = Weight due to handling built-in exceptions
- NOBE = Number of built-in exceptions
- UDE = Weight due to user defined exceptions
- NOUDE = Number of user defined exceptions

Complexity of class A (CA) =

TW class A + OL class A + OR class A + VW class A + CW class A + BE class A + UDE class A

Complexity of program (P) =

(CA * IWCA) + (CB * IWCB) + (CC * IWCC) + ………………… + (CZ * IWCZ)

```java
public class A {

	public int add(int x,int y){
		return x+y;
	}

}

public class B extends A {

	private void methodC(int property1, int property2){

		if(property1 == 3){
			System.out.println("Property1 is 3. ");
		}
		while(property2 !=0){
			property2++;
			System.out.println("Property2 value is incremented by 1");
		}

	}


	public static void main(String args[]){

	   B b = new B();
	   b.methodC(10,30);

	}

}
```

Complexity of class A (CA) = TW class A + OL class A + OR class A + VW class A + CW class A + BE class A + UDE class A

Sample weight for integer type variable = 2

Number of integer type variables = 2

Complexity of class A (CA) = 0 + 0 + 0 + (2*2) + 0 + 0 + 0

= 4

Complexity of class B (CB) = TW class B + OL class B + OR class B + VW class B + CW class B + BE class B + UDE class B

Sample weight for integer type variable = 2

Number of integer type variables = 2

Sample weight for control structure type (if) = 2

Sample weight due to nesting level of control structure (if) = 1

Sample weight for control structure type (while) = 4

Sample weight due to nesting level of control structure (while) = 2

Sample weight for method call to a regular method in the same class = 1

Number of method call to a regular method in the same class = 2

Complexity of class B (CB) = ((2*1) + (4*2)) + 0 + 0 + (2*2) + (1*2) + 0 + 0

= (2+8) + 4 + 2

= 10 + 4 + 2

= 16

Complexity of program (P) = (CA * IWCA) + (CB * IWCB)

= 4 + 16

= 20

RESULTS & DISCUSSION

- According to analysis of collected data and information, the most essential factors which need to be considered in measuring source code quality were identified.
- Depending on the factors that recognized, we found out the sample weight which can be allocate.
- As a result, we were able to propose a new metric to evaluate the source code quality of a program written in Java language.
- Thereby, we develop a program analysis tool which is capable of rating software developers by determining source code quality with the use of our proposed metric.

CONCLUSION

- Our proposed metric to measure complexity of a program may be accurate due to large sample size of data and information and clear definition.
- Hence the metric evaluation performed by the program analysis tool may be accurate.

REFERENCES

[1] Arnold, K., Gosling, J., Holmes, D., & Holmes, D. The Java programming language (Volume 2). Reading: Addison-wesley. 2000

[2] Sanchez, S. M., & Lucas, T. W. Exploring the world of agent-based simulations: simple models, complex analyses: exploring the world of agent-based simulations: simple models, complex analyses. In Proceedings of the 34th conference on Winter simulation: exploring new frontiers. 2002. Pages 116-126

[3] Shrivastava, S. V., & Shrivastava, V. Impact of metrics based refactoring on the software quality: A case study. In TENCON 2008- 2008 IEEE Region 10 Conference. 2008. Pages 1-6.

[4] Davis, J.S., & LeBlanc, R.J. A Study of the Applicability of Complexity Measures. IEEE Transactions on Software Engineering, Volume 14. Number 9. 1988