ProAnalýza

PROPOSAL PRESENTATION

Project ID: 18-055

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

ProAnalýza

PROGRAM ANALYSIS TOOL

OUTLINE

INTRODUCTION

WHY IS WRITING QUALITY SOFTWARE & CODE IMPORTANT

Simply delivering functionality is not enough

- It’s crucial that developers pay attention to quality attributes
- Furious rate of product development
- Software updated multiple times
- Testing must be done every time system changes
- Otherwise high cost and effort to test and maintain system

- What cant be measured
- Cant be managed

?

WHY DO WE NEED TO ANALYSE CODE

- 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

ANALYSING CODE HELPS TO BRING DOWN

DEVELOPMENT EFFORT

DEVELOPMENT TIME

NUMBER OF PEOPLE INVOLVED

COST

OBJECTIVES

Develop a program analysis tool which :

- Allows its users to easily understand a given program
- Rate the software developers by evaluating their code quality
- Improve code quality
- Bring down the Cost of IT projects

LITERATURE SURVEY

PROGRAM ANALYSIS TECHNIQUES

Static Program 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.

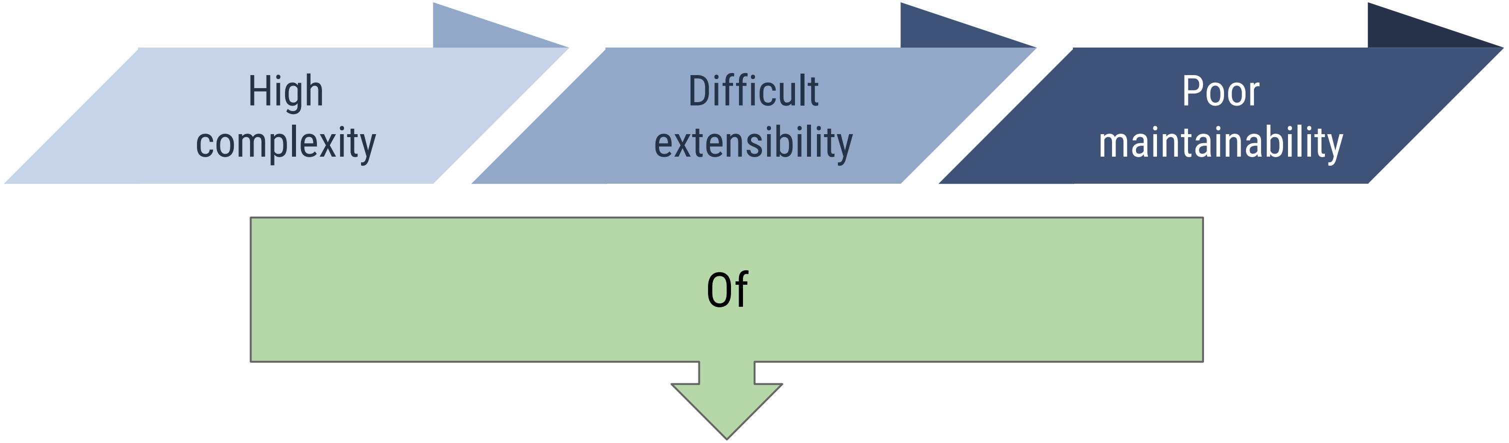
Dynamic Program Analysis

Can use runtime knowledge of the program to increase the precision of the analysis while also providing runtime protection but can only analyze a single execution of the problem and might degrade the program's performance due to runtime checks.

“ *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



Source code

RESEARCH PROBLEM

HYPOTHESIS

“Program analysis is a substantial process to understand the source code. This needs effective, reliable, and accurate program analysis tools, but these tools may mislead the software developers because they might provide inaccurate measures.”

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 DOMAIN

NATURAL LANGUAGE PROCESSING (NLP)

- The major software engineering artifact is source code.
- We aim to support NLP analysis in the software domain, which must be possible to process source code using standard NLP tools.
- Example: In order to analyze comments, identifiers, strings, and other NL components.

SOLUTION

“DEVELOP AN AFFORDABLE & RELIABLE PROGRAM ANALYSIS TOOL”

UNIQUENESS OVER OTHER EXISTING SYSTEMS

- Can be used to analyse Java code
- 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
- Code inspection

METHODOLOGY

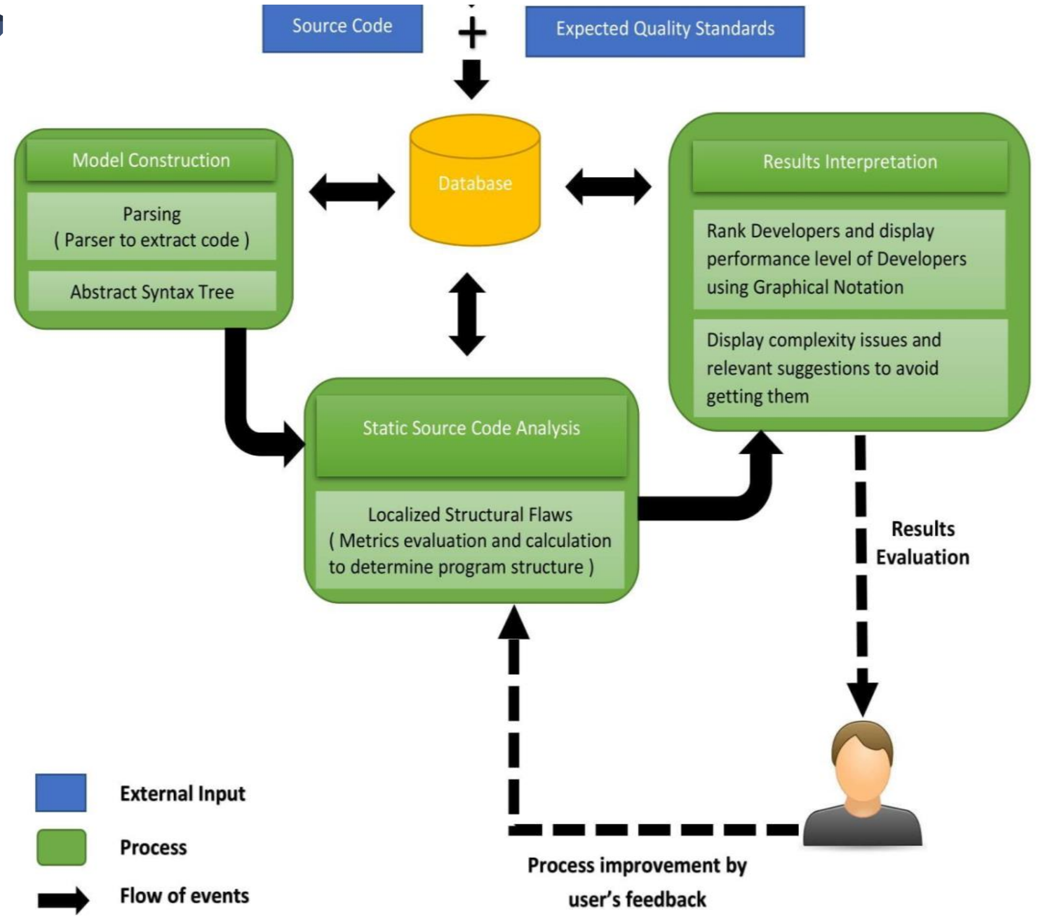
TOOLS & TECHNOLOGIES

Tools

- Eclipse IDE
- ANTLR
- Notepad++
- Xampp
- POSTMAN

Technologies

- MySQL
- Java
- HTML
- PHP5
- CSS
- AngularJS
- Node.js


Source Code
+
Expected Quality Standards
Database
Model Construction
Parsing
(Parser to extract code)
Abstract Syntax Tree
Results Interpretation
Rank Developers and display performance level of Developers using Graphical Notation
Display complexity issues and relevant suggestions to avoid getting them
Static Source Code Analysis
Localized Structural Flaws
(Metrics evaluation and calculation to determine program structure)
Results Evaluation
Process improvement by user's feedback
External Input
Process
Flow of events

High Level System Architecture

Code Parsing

$$\Downarrow$$

Abstract Syntax Tree

Example :

a = b + c;

$\Big\downarrow$ Lexical Analysis

Sequence of tokens
<id, 1> <=> <id, 2> <+> <id, 3>

$\Big\downarrow$ Syntax Analysis

Abstract Syntax Tree

CODE PARSING WITH NLP

Why we chose ANTLR ?

- Has a consistent syntax for specifying lexers, parsers, and tree parsers.
- Various plugins have been developed for the Eclipse development environment to support the ANLTR grammar
- Comes with complete source code unlike many other systems and has absolutely no restrictions on its use.
- Pretty flexible and decent error handling.
- ANTLR is well supported and has an active user community.

Metrics Evaluation & Calculation

Measure complexity of source code with

McCabe's Complexity Measures (Cyclomatic complexity)

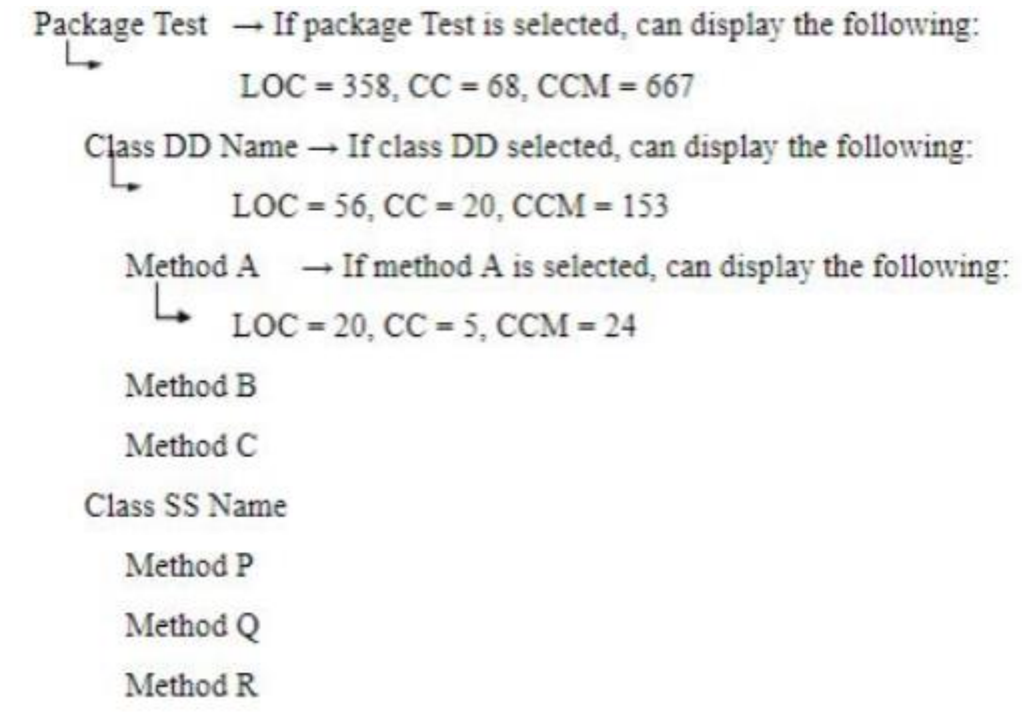
Halstead's Complexity Measures

Factors under consideration

1. Exception handling
2. Memory Consumption
3. Inheritance
4. Nesting Levels and Control Structures
5. Operators / Operands
6. LOC
7. Coupling

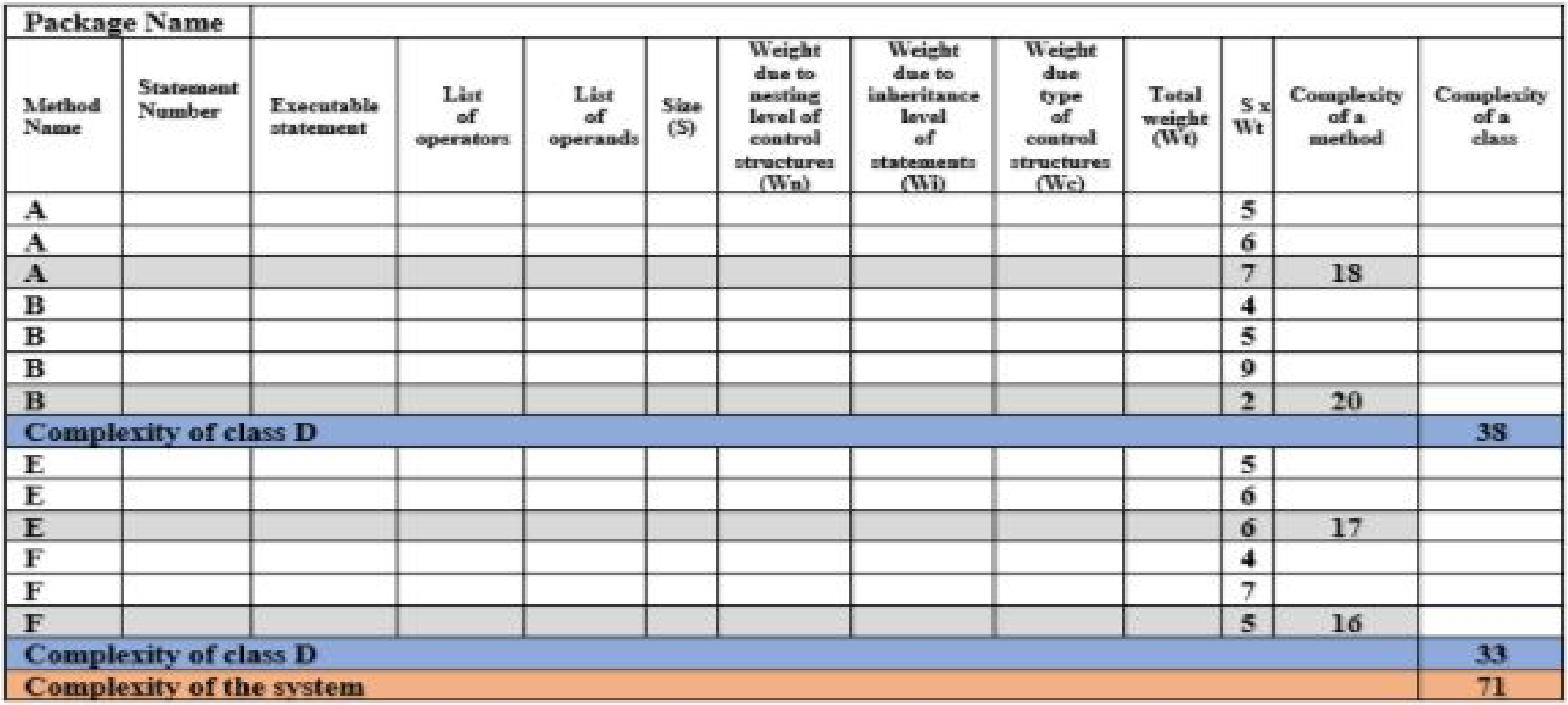
RESULTS INTERPRETATION

Example:



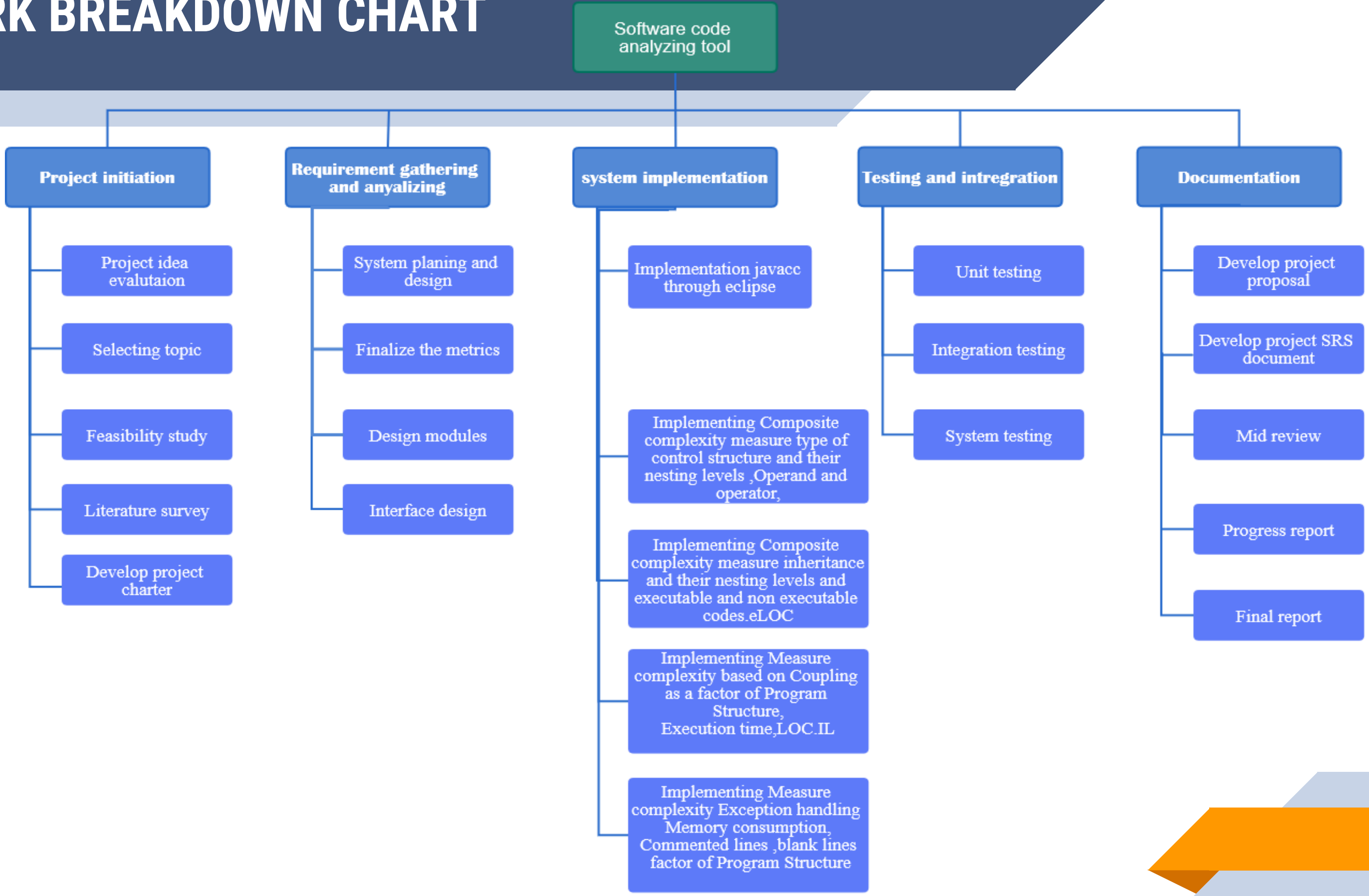
```
Package Test  → If package Test is selected, can display the following:
  ↳
                LOC = 358, CC = 68, CCM = 667
    Class DD Name → If class DD selected, can display the following:
      ↳
                LOC = 56, CC = 20, CCM = 153
       Method A    → If method A is selected, can display the following:
         ↳      LOC = 20, CC = 5, CCM = 24
       Method B
       Method C
    Class SS Name
       Method P
       Method Q
       Method R
```

Interpretation of the composite complexity of a system based on user's requirement



Package Name												
Method Name	Statement Number	Executable statement	List of operators	List of operands	Size (S)	Weight due to nesting level of control structures (Wn)	Weight due to inheritance level of statements (Wi)	Weight due type of control structures (Wc)	Total weight (Wt)	S x Wt	Complexity of a method	Complexity of a class
A										5		
A										6		
A										7	18	
B										4		
B										5		
B										9		
B										2	20	
Complexity of class D												38
E										5		
E										6		
E										6	17	
F										4		
F										7		
F										5	16	
Complexity of class D												33
Complexity of the system												71

WORK BREAKDOWN CHART



Software code analyzing tool
Project initiation
Project idea evalutaion
Selecting topic
Feasibility study
Literature survey
Develop project charter
Requirement gathering and anyalizing
System planing and design
Finalize the metrics
Design modules
Interface design
system implementation
Implementation javacc through eclipse
Implementing Composite complexity measure type of control structure and their nesting levels ,Operand and operator,
Implementing Composite complexity measure inheritance and their nesting levels and executable and non executable codes.eLOC
Implementing Measure complexity based on Coupling as a factor of Program Structure, Execution time,LOC.IL
Implementing Measure complexity Exception handling Memory consumption, Commented lines ,blank lines factor of Program Structure
Testing and intregration
Unit testing
Integration testing
System testing
Documentation
Develop project proposal
Develop project SRS document
Mid review
Progress report
Final report

INDIVIDUAL COMPONENTS

Gamaarachchi G.A.C.Y **IT15111548**	**M.A.N.S.U.K. Uvindasiri** **IT15413802**	**K.G.D.R Perera** **IT15112538**	**P.K.H Palihakkara** **IT15113900**
Code parsing with NLP & analyze code in terms of exception handling & memory consumption	Code parsing with NLP & analyze code by measuring Composite complexity including control structures, their nesting levels, operands & operators	Code parsing with NLP & analyze code with respect to LOC & Coupling	Code parsing with NLP & analyze code by measuring Composite complexity including inheritance & their nesting levels

COMMERCIAL VALUE

TARGET MARKET & USERS

Schools/ Universities

Students

Lecturers

Teachers

IT Companies

Programmers

Testers

Maintainers

Project Managers

Research Forums

Researchers

Scholars

BENEFITS

- Allow to quickly & easily remove code bottlenecks
- Reduce testing cost
- Reduce maintenance cost
- Allow to improve the efficiency of programs
- Allows programmers to improve their coding skills
- Allows to differentiate the performance of each programmer
- Allows to increase the productivity of programmers involved in a project

CONCLUSIONS

- The existing program analysis tools are unreliable & even unaffordable.
- Hence there’s a requirement for a program analysis tool which evaluate & calculate metrics accurately.
- Static program analysis is more precise in discovering vulnerabilities during the development phase of the program.
- NLP can be used to develop the tool.

THANKS!

Any questions?