

Source Code Analysis



MTU

Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

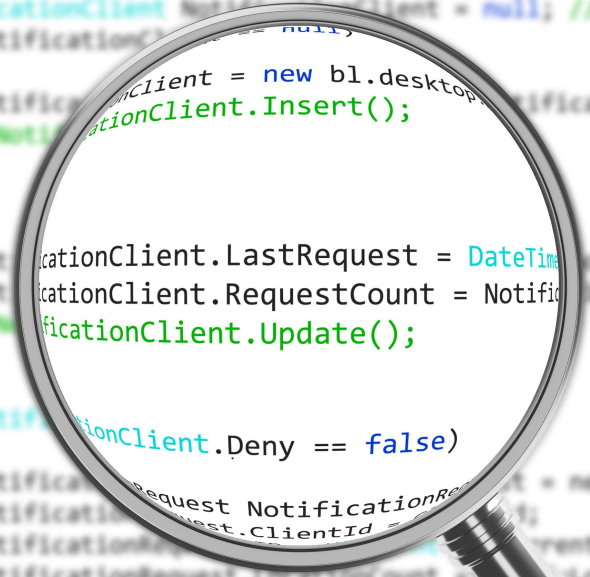
Module Description

- 5 credit module
- CA: Continuous assessment
- Two projects
 - ❑ Project 1: Around week 7
 - ❑ Project 2: End of semester
- Java and Eclipse
- Maven

About me:

- Lecturer at MTU in CS department.
- PhD in Computer Science from University of Limerick.
- Research Experience: Source Code Analysis, Feature Location, Source Code Manipulation, Data Analytics, Graph Visualization, Genetic Algorithms, Neural Networks.
- Email: farshad.toosi@mtu.ie

This Module is ALL about Source Code



```
//Load values from database store with channel select
NotificationClient NotificationClient = null; // = NotificationClient.Ge
if (NotificationClient == null)
{
    NotificationClient = new bl.desktop.NotificationClient() { Deny = fa
    //NotificationClient.Insert();
}
else
{
    NotificationClient.LastRequest = DateTime.Now;
    NotificationClient.RequestCount = NotificationClient.RequestCount + 1;
    //NotificationClient.Update();
}
if (NotificationClient.Deny == false)
{
    NotificationRequest NotificationRequest = new bl.desktop.Notification
    NotificationRequest.ClientId = NotificationClient.ClientId;
    NotificationRequest.CurrentRequest.UserHostAddress = NotificationClient.CurrentRequest.UserHostAddress;
    NotificationRequest.LocationCount = NotificationClient.Locations.Count;
    NotificationRequest.Timestamp = DateTime.Now;
    //Redirect message
    for (int i = 0; i <= NotificationClient.Locations.Count; i++)
```



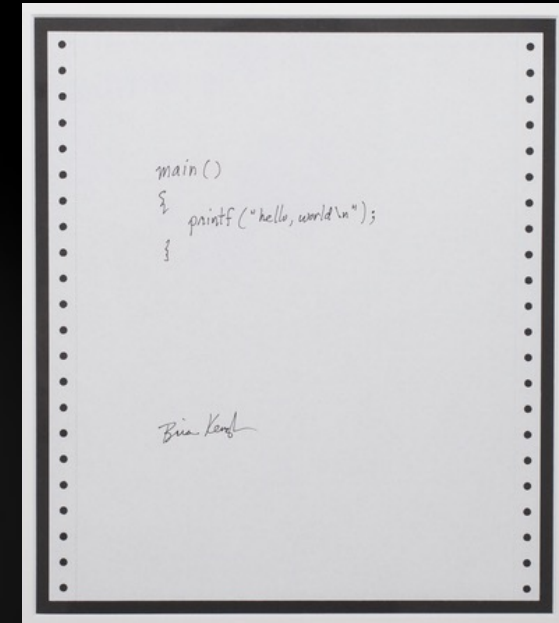
What will I learn from this module?

1. Clean Practices in Coding mostly related to Objected Oriented languages.
2. What is standard practice and what is Code Smell.
3. Source Code parsers.
4. How to detect bad practices or Code Smell and some security vulnerability assessment (E.g., SQL Injections).
5. Feature Location A.K.A Bug location.

Although the base language is Java, you can apply the techniques on any language in future.

Software Program

- A software program is a collection of instructions that performs a particular task.
- The human-readable form of a software system is called source code.
- Source code is any static, textual, human readable, that can potentially be compiled or interpreted automatically into an executable form.
- Source code is written by programmers and requires a high amount of precision and accuracy for the required task.
- Although software systems can perform different tasks, a lack of attention in designing (coding) a good software system might potentially lead to a faulty system.



"hello, world" computer program by Brian Kernighan (1978)

Software programs are like wools. Wools have tendency to attract dust as they have static electricity. Programs tend to attract defects and bugs.

Christof Ebert Static Code Analysis



What is happening in the Software World?

- A tremendous increase in the amount of software in use.
- **Software needs to be constantly maintained. Why?**
 - Successful software requires constant change that is triggered by evolving requirements, technologies, and stakeholder knowledge.
(Václav Rajlich 2014)
- A large amount of demand for programmers due to a large amount of software in use.



Developers generate more than 111 billion lines of new software code every year.

What is happening in the Software World?

❖ **Software needs to be constantly maintained. *Why.***

❖ **Active Software often requires change and update:**

- ❖ **Bug Fixing**
- ❖ **Efficiency improvement (compatibility with new hardware)**
- ❖ **Adding new functionalities**
- ❖ **Security updates**
- ❖ **and many more ..**

What is happening in the Software World?

That's what we are going to do this semester.

- Updating software generally involves three phases: (Robilard Coelho 2004)
 1. Understanding the existing software.
 2. Modifying the existing software. (The actual modification)
 3. Revalidating the modified software.
- *No efficient modification/update is possible before understanding the system.*

Software Modification

2. Modifying the existing software. (The actual modification):

- Adding a new functionality. Does the new functionality affect rest of the system?
- Changing the functionality/behavior of an existing component. Does this change make any change on the behavior of other components of the system?
- Removing an unnecessary functionality from the software system. What is the strategy to find an unnecessary functionality from a software system?
- Fixing a bug. How to locate the bug?
- Migrating to a new platform or a programming language; e.g., Procedural code to OO code. How to start the migration? How to create classes from a none-OO paradigm?
- Getting rid of redundant components in the software system that was stacked up over the time. How to detect redundant codes (code clones)?
- Fixing code bad smell. Detect irrelevant comments, none-expressive variable/method/class names and etc.

Software validation

3. Revalidating the modified software:

- **Software testing.** If the system works generally, e.g., no compilation error, no run-time error, no crashing etc.
- **Test case scenarios.** Some particular scenarios should be tested separately.
- **Security tests.** Any change might potentially jeopardize the security of the system; sufficient tests need to be performed to re-validate the security.

What is happening in the Software World?

- **Updating software generally involves three phases:** (Robilard Coelho 2004)
 1. **Understanding the existing software.**
 2. **Modifying the existing software. (The actual modification)**
 3. **Revalidating the modified software.**
- *No efficient modification/update is possible before understanding the system.*

Understanding the existing software

In order to understand a software system, the **source code** needs to be **inspected/studied/analyzed**.

Software system is similar to a house, any new change or update requires some analysis or/and inspection.



Understanding the existing software

Imagine you are hanging a frame on the wall, first thing you need to do is to put a nail into the wall.

What do you need to know before putting the nail?

- Is there any electric wire passing by where the nail is being put? So, the structure of the house needs to be inspected.

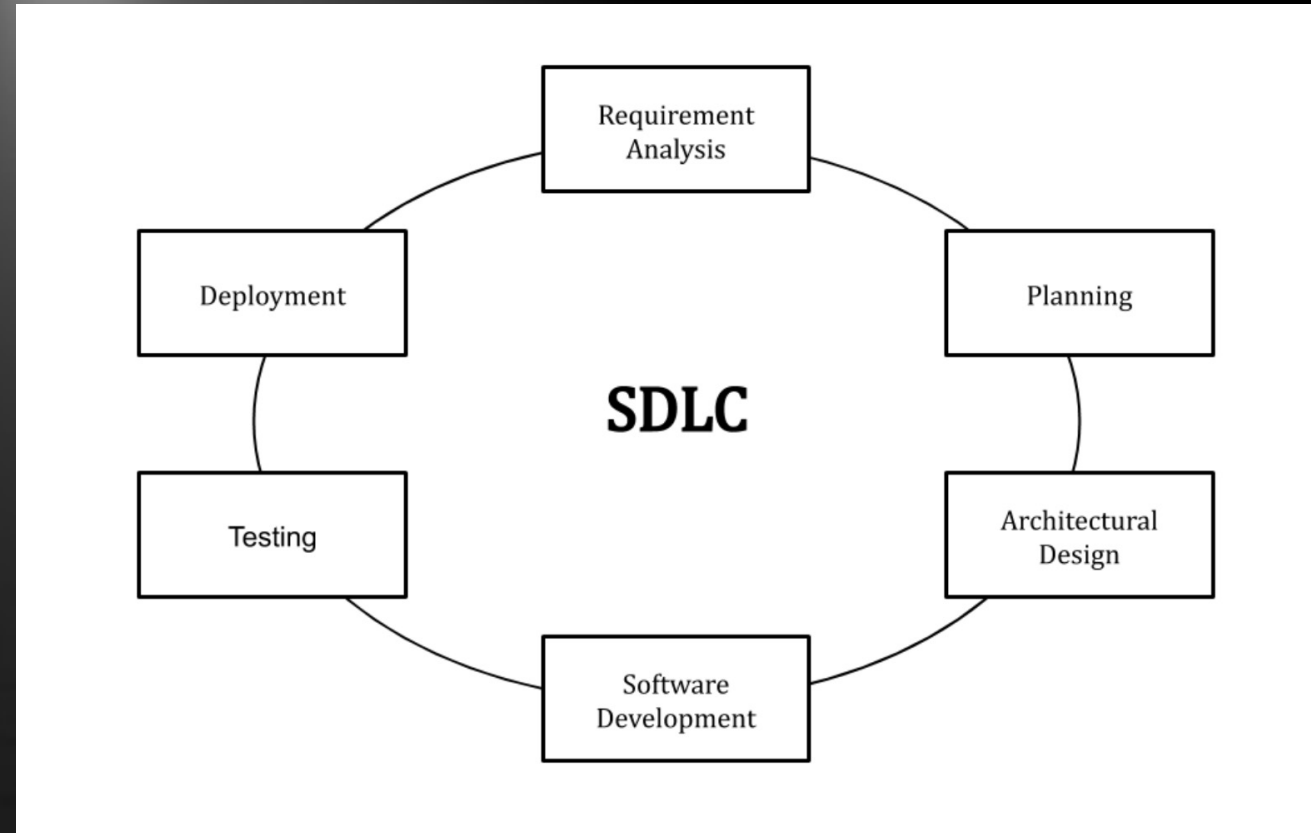


An other View:

Software Development Life Cycle (SDLC)

The Software Development Life Cycle (SDLC) refers to a methodology with clearly defined processes for creating high-quality software. In detail, the SDLC methodology focuses on the following phases of software development:

Late fixes/retrofits are
always expensive and risky

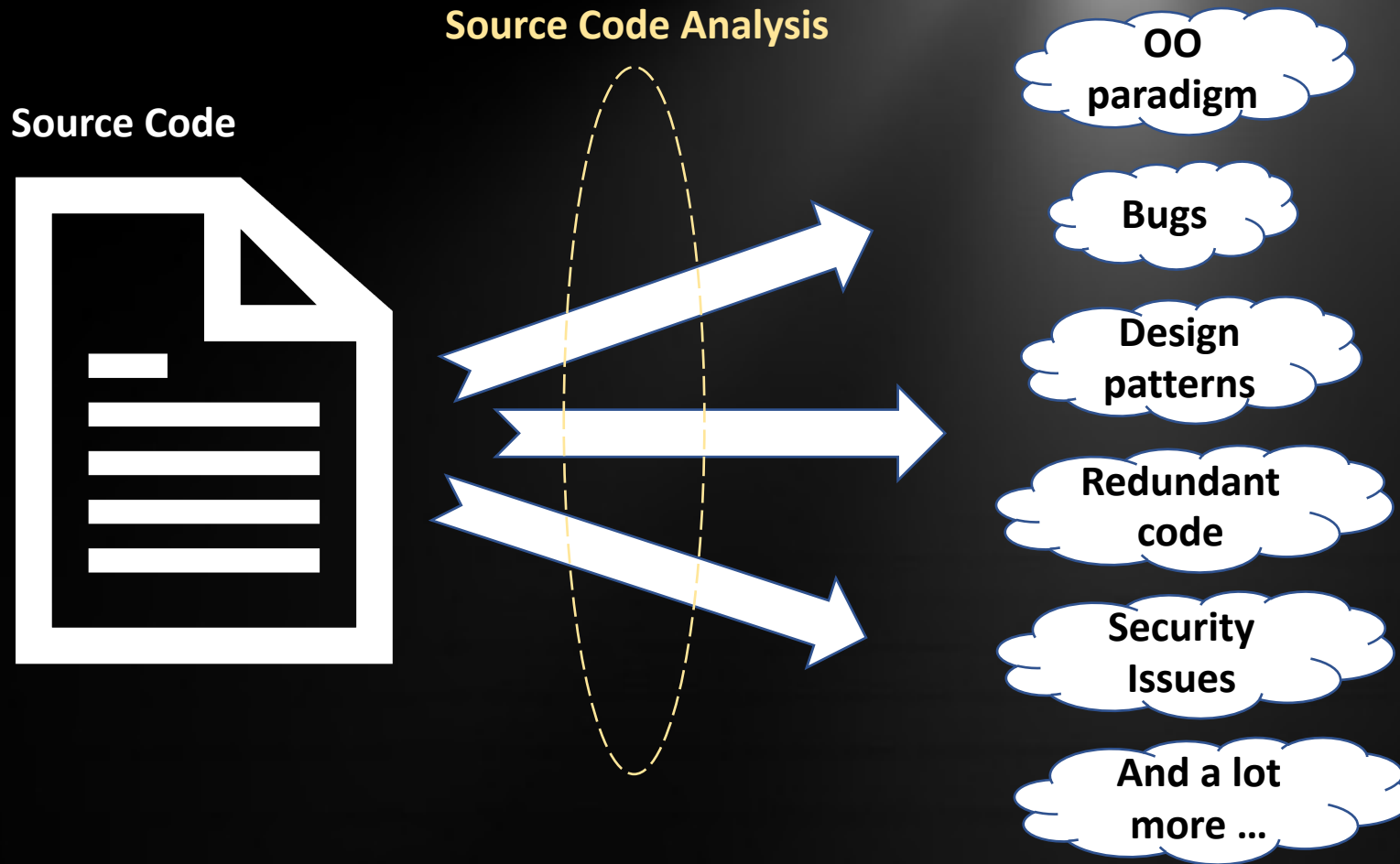


What is Source Code Analysis and its relation to SDLC?

Static code analysis (SCA) is one of the driving forces behind the secure SDLC philosophy after the requirements have been clearly defined and clarified to the developers.

One of the biggest advantages of using static code analysis throughout the SDLC is that testing can be fully automated, enabling developers to implement secure coding practices and sanitize the whole development process with minimal effort.

What does Source Code Analysis involve?



Source code analysis is the process of extracting information about a program from its source code.

Source code analysis is a set of **techniques** and **technologies** that are applied on program's source code with the purpose of understanding the underlying of the software system for some further tasks.

Usages of Source Code Analysis ?

- ❖ SCA techniques are used to identify functionalities/features/bugs from a based code.
- ❖ Developers can use this information for different purposes such as:
 - ❖ Modernizing a legacy system.
 - ❖ Debugging.
 - ❖ Preventing from code redundancy.
 - ❖ Separating different functionalities from each other.
 - ❖ dead code elimination.
 - ❖ Security vulnerability assessment.
 - ❖ Etc ...



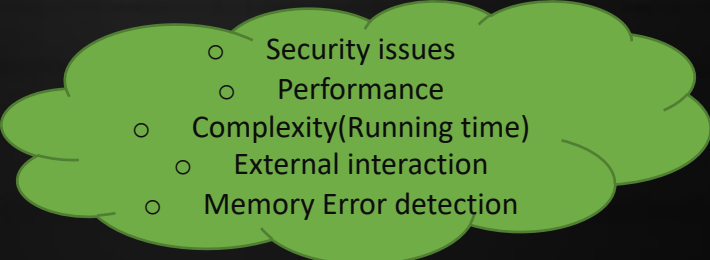
Analyzing the Software System

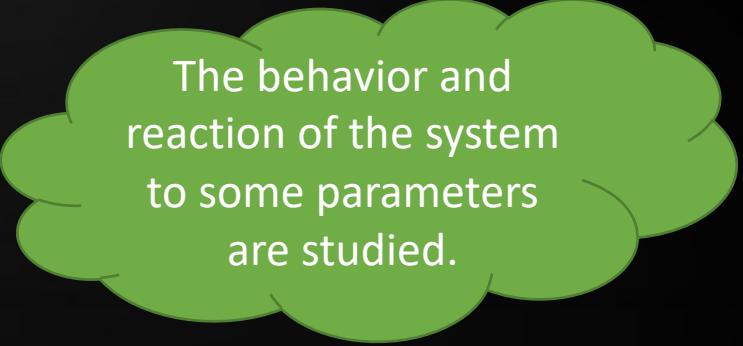
- **Static analyzing.** (Based on only source code)
- **Dynamic analyzing.** (Based on source code execution)
- **Hybrid analyzing.** (Based on both static and dynamic analysis)

Analyzing the Software System

Dynamic analyzing

- ❖ Dynamic code analysis is a number of methods of analyzing an application right during its execution.
- ❖ Dynamic code analysis is only possible when the source-code is compiled and it is error free.
- ❖ Dynamic code analysis has the following steps:
 - ❖ Preparing input data.
 - ❖ Designing and running a scenario (test run) launch.
 - ❖ Observing the targeted parameters.
 - ❖ Analyzing the output data.
 - ❖ Etc.

- 
- Security issues
 - Performance
 - Complexity(Running time)
 - External interaction
 - Memory Error detection



The behavior and reaction of the system to some parameters are studied.

Analyzing the Software System

Static analyzing

- ❖ Source code does not necessarily have to compile.
- ❖ The source code may have bugs.
- ❖ The source code is analyzed before the program is run.
- ❖ It's done by analyzing a set of code against a set (or multiple sets) of coding rules.

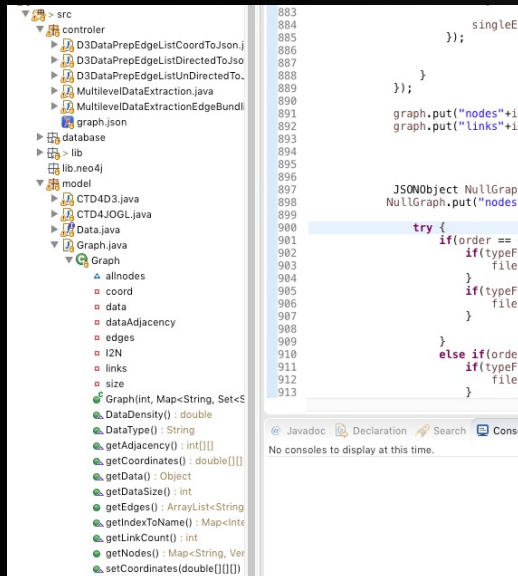
Static Analysis

From the structural point of view, a software system is similar to a building.

Both have a number of different components that are connected to each other in some fashion.

Failure in one section might lead to a defect in other sections.

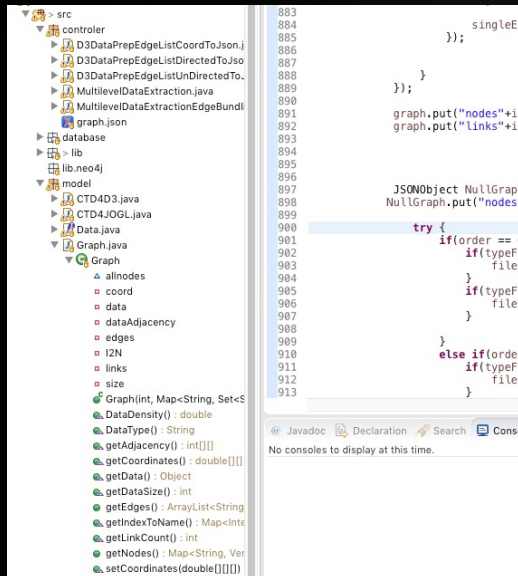
Therefore, components are dependent to each other.



Static Analysis

The first step in analyzing a building or a software system is to know the components and their relations to each other.

Components of a software system are comprised of a series of syntax and symbol.



Variable

Method/function

Class

Method Call

Data flow



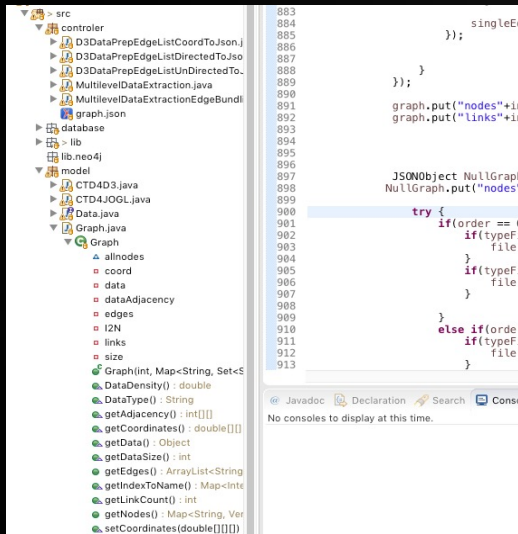
Static Analysis

One of the very early stage of a source-code analysis task is to analysis the syntax of the code.

Syntax analysis or **Parsing** is the process of analyzing a set of syntax or symbols in a programming language.

Each programming language has its own parser as each programming language has a different grammar.

E.g., some languages are typed languages some are un-typed.



Variable

Method/function

Class

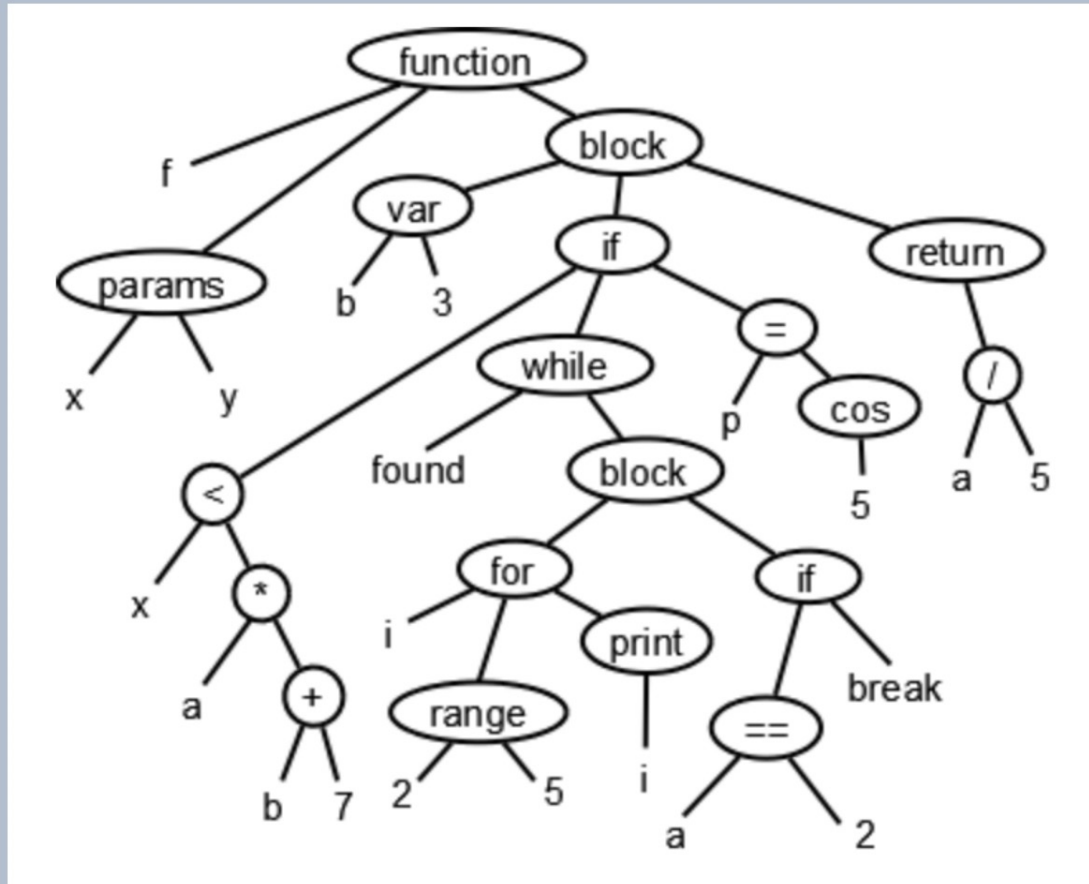
Method Call

Data flow

Static/Source Code Analysis

Language Parser

A **programming language parser** is a program itself that works out the components of the software and their relations to each other.



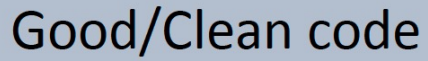
Static Analysis

There are several different defects that can be found by analysis a source code such as:

- Lack of OO paradigms; (e.g., lack of encapsulation)
- Inappropriate inheritance (e.g., inheritance smell)
- Irrelevant Information in the comments
- Too many input arguments in functions
- Obsolete Comment
- Existing of Dead-code
- Redundant Comment
- Security issues (e.g., SQL Injections)
- Meaningless and/or misleading variable names
- Poorly Written Comment
- Code duplication (Don't Repeat Yourself, DRY)
- Commented-Out Code

Some aspects of what we will be doing...

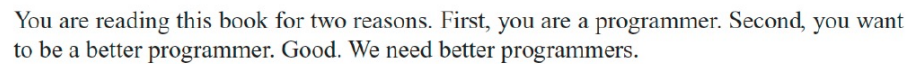
- Source code components and relationship identification. E.g., Call graph visualization.
- Redundant code identification. Code Clone detection.
- Functionality/Bug location. Feature Location.
- Object Oriented inspection in source-code.
- Code smell detection. E.g., comments, names etc. Security issues detections.



Source Code Quality

Clean Code

A Handbook of Agile Software Craftsmanship



Dr Farshad Ghassemi Toosi

A spotlight effect is centered at the top of the slide, casting a bright beam of light downwards onto the title text.

Source Code Quality


What metrics do
you consider to
measure the source
code quality?

Source Code Quality

There are several different defects that can be found by analysing the source code such as:

- Inappropriate Information in the comments. AI and Machine learning techniques can be used as well e.g., NLP.

```
boolean changed = false;
try {
    PrintWriter out = new PrintWriter(file);
    if (mesg.contains("sh") || mesg.endsWith("ing")) {
        mesg = mesg.replace("sh", "ch");
        changed = true;
        out.println(mesg);
    }
} catch (FileNotFoundException e) {
    // File does not exist
}
```



Inappropriate comment and inappropriate location and design.

```
// This is a stupid class that I wrote under duress. I apologize to
// all affected.
```



Inappropriate comment by a developer in a team of developers.

Source Code Quality

There are several different defects that can be found by analysis a source code such as:

```
//this method takes two integer number and calculates their average.  
public static double average (double [] numbers) {  
  
    double ave = 0;  
    for(int i =0;i<numbers.length;i++) {  
        ave += numbers[i];  
    }  
    return ave/numbers.length;  
}
```



Obsolete
Comment

Source Code Quality

There are several different defects that can be found by analysing a source code such as:


- Redundant Comment
- Poorly Written Comment
- Commented-Out Code

```
//creating a new array list for house sizes  
ArrayList<Integer> houseSize = new ArrayList<Integer>();
```


Source Code Quality

Source code structure:

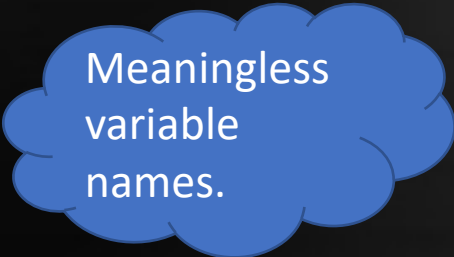
- Too many arguments for functions
 - Code duplication
 - Dead code
 - Variable names



Too many arguments.

```
public static void contentCheck (int size, double rate, String message, File file, boolean checked) {
```

```
int c = -1;  
double w = 1.5;
```



Meaningless variable names.

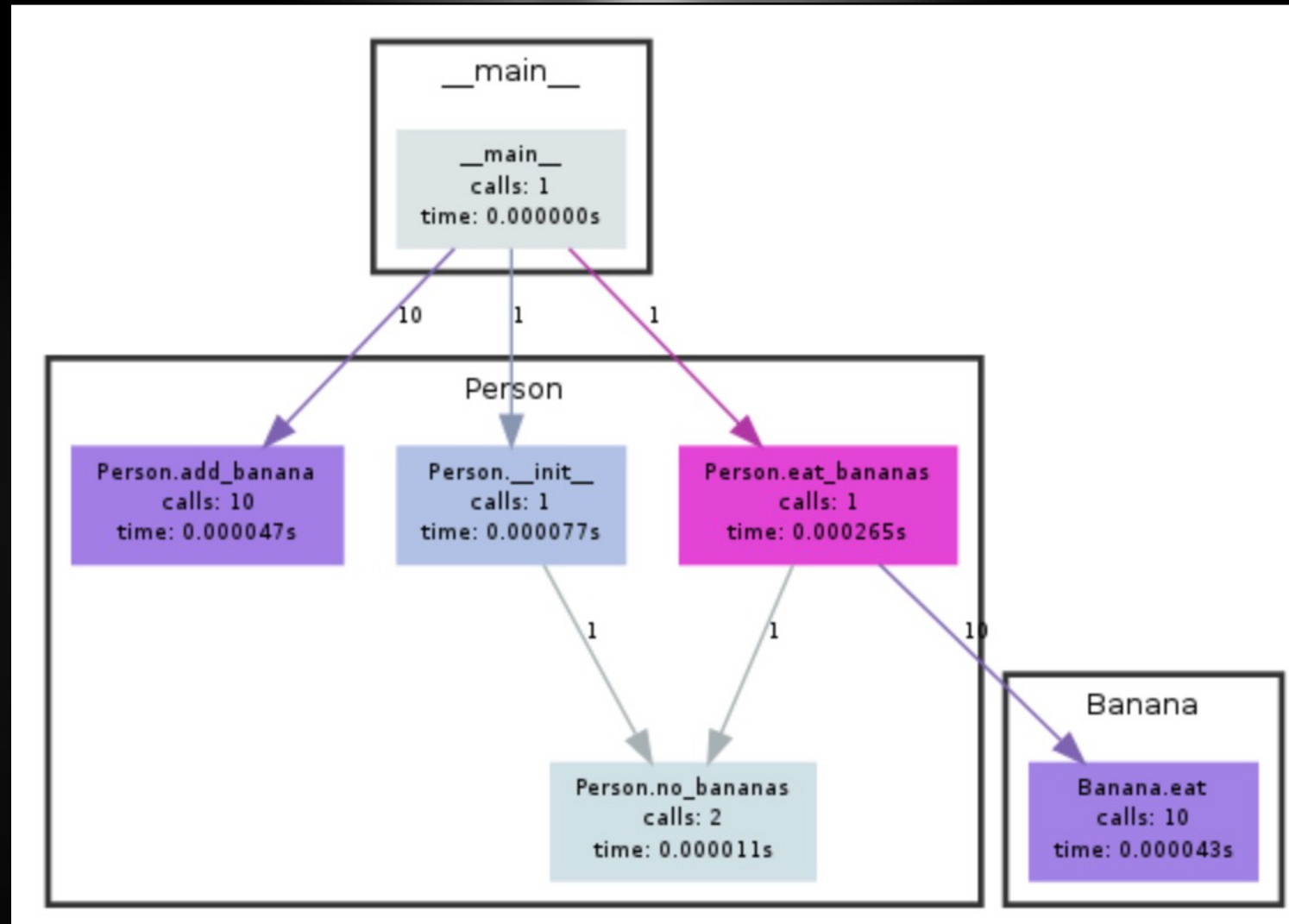
Source Code Quality

Error Handling

```
public void doNotIgnoreExceptions() {  
    try {  
        // do something  
    } catch (NumberFormatException e) {  
        // this will never happen  
    }  
}
```

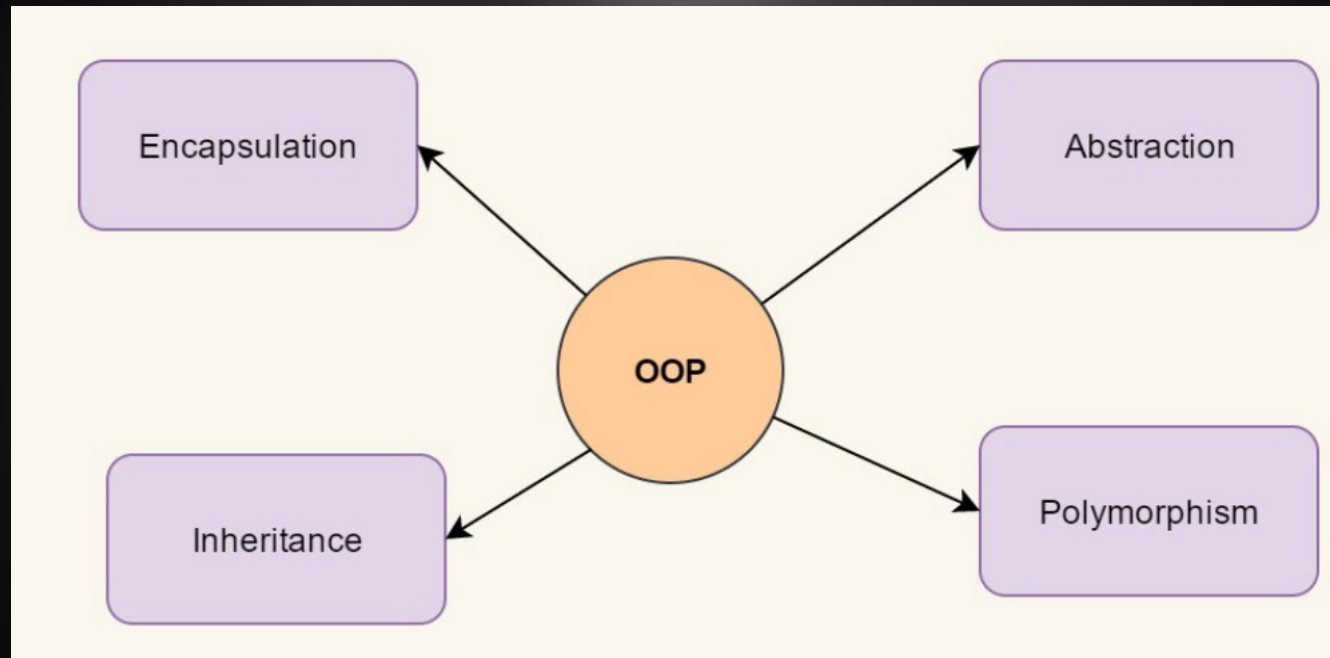
Source Code Quality

Call Graph:



Source Code Quality

OOPs



Source Code Quality

Dead Code:
Blocks of code that
are not used
anymore.



Source Code Quality

Identical/similar blocks of code

```
public static int myFunction() {  
    int n = 0;  
    int u = 0;  
  
    while(n<0) {  
        u += 0.5;  
    }  
    return u;  
}
```

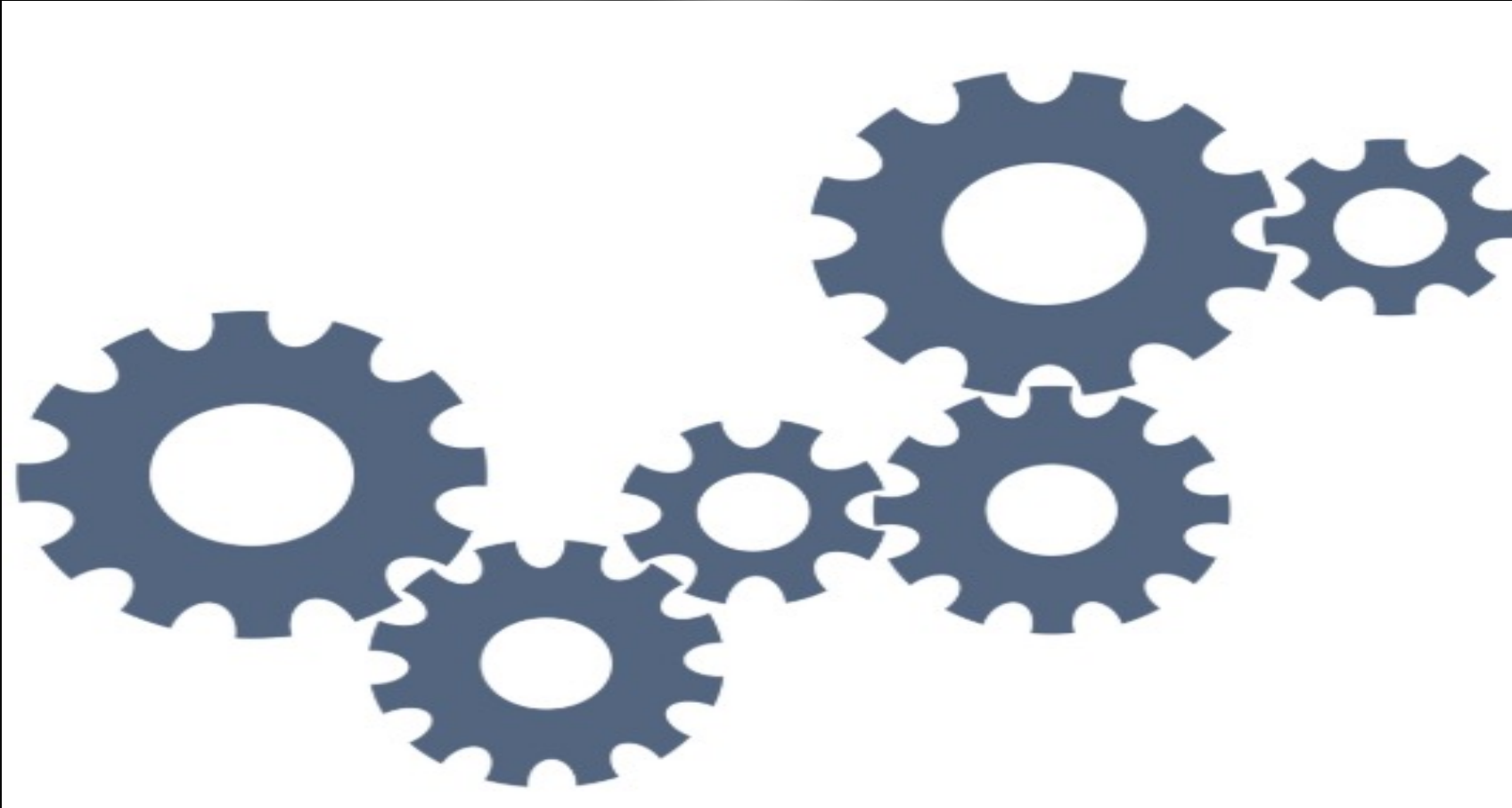
```
public static int myFunction() {  
    int n = 0;  
    int g = 0;  
  
    while(n<0) {  
        g -= 0.5;  
        g++;  
    }  
    return g;  
}
```

Feature Location

The process of locating/identifying a particular functionality (it can be a bug) in the source code.

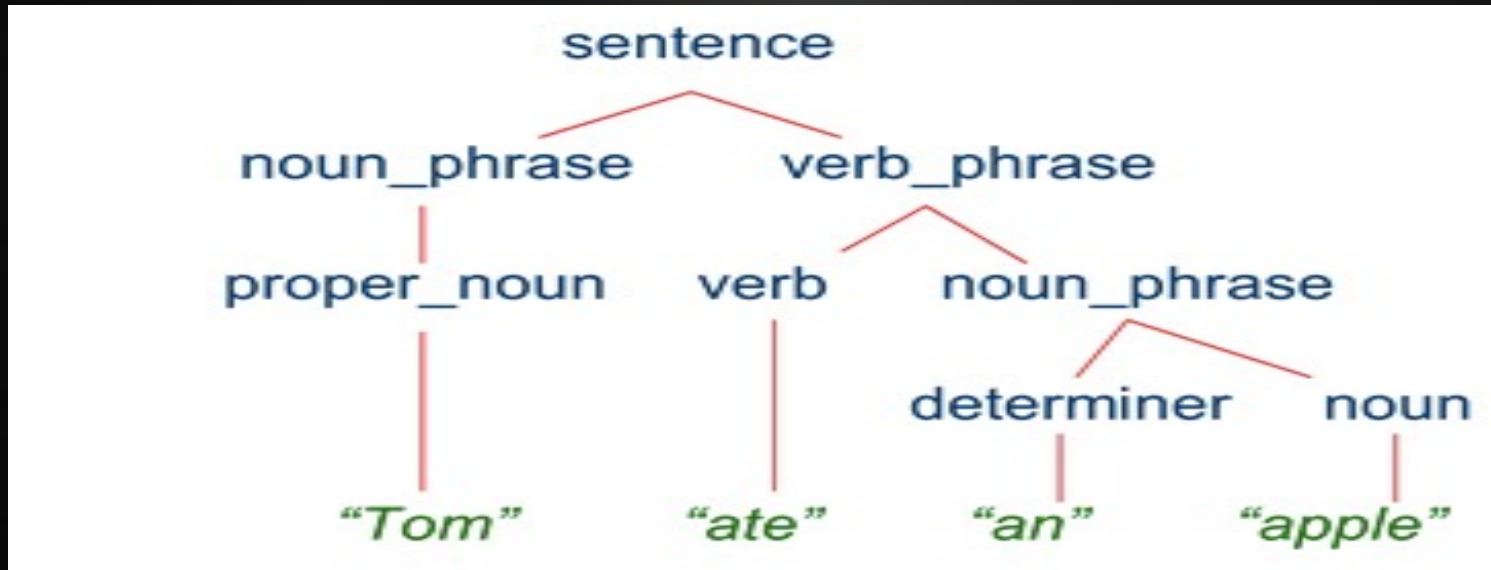
- Using textual search methods e.g., NLP
- Using call graph
- Using components dependencies
- ...

Parsing



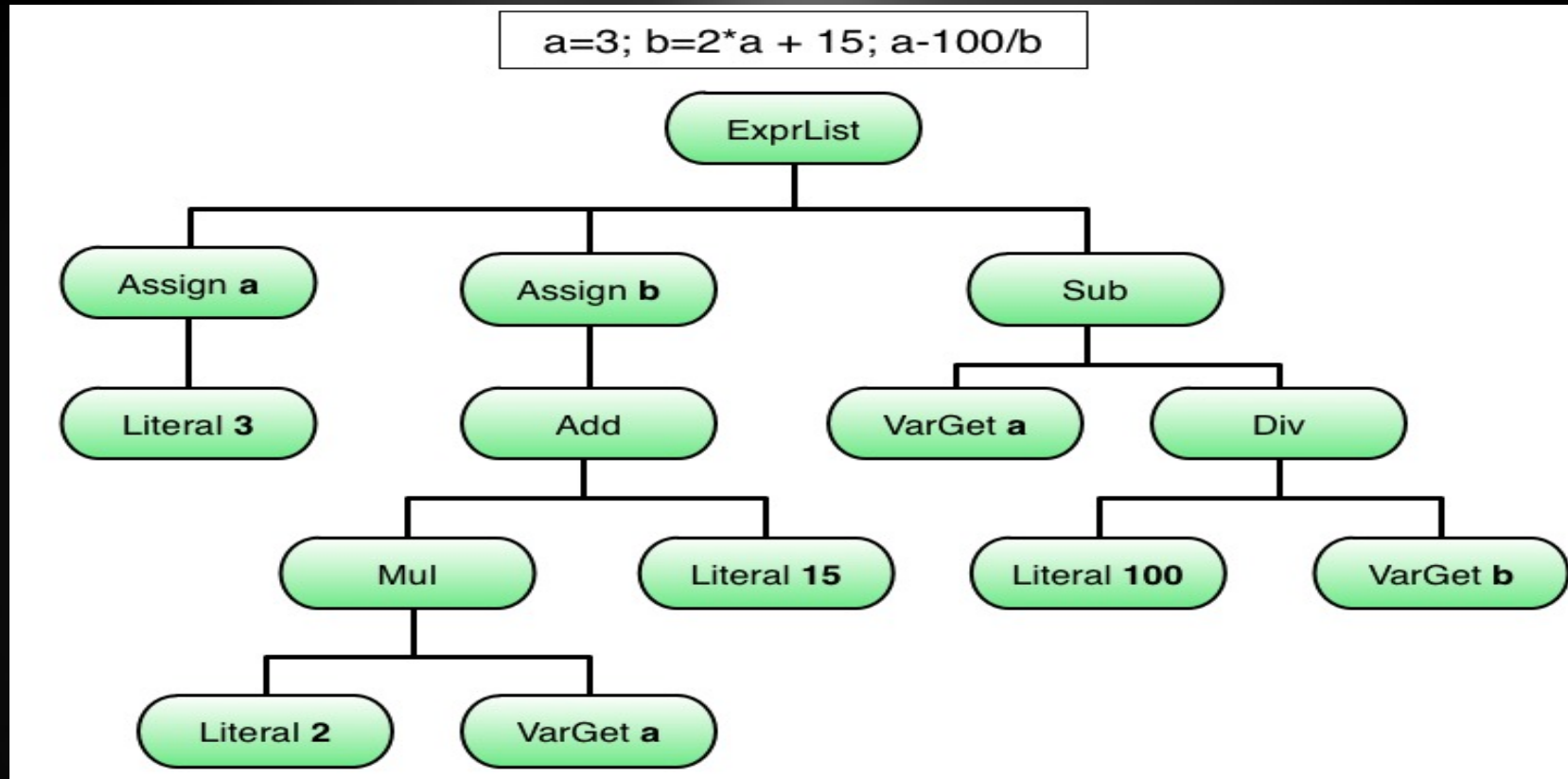
What is Parsing?

- ❖ In a simple word, *Parsing* is a process of analyzing some symbols (can be a string symbols) in a different language.
- ❖ The following shows parsing a human language sentence (*Tom ate an apple*) in a formal language that represents the grammatical roles of each word.



What is Parsing?

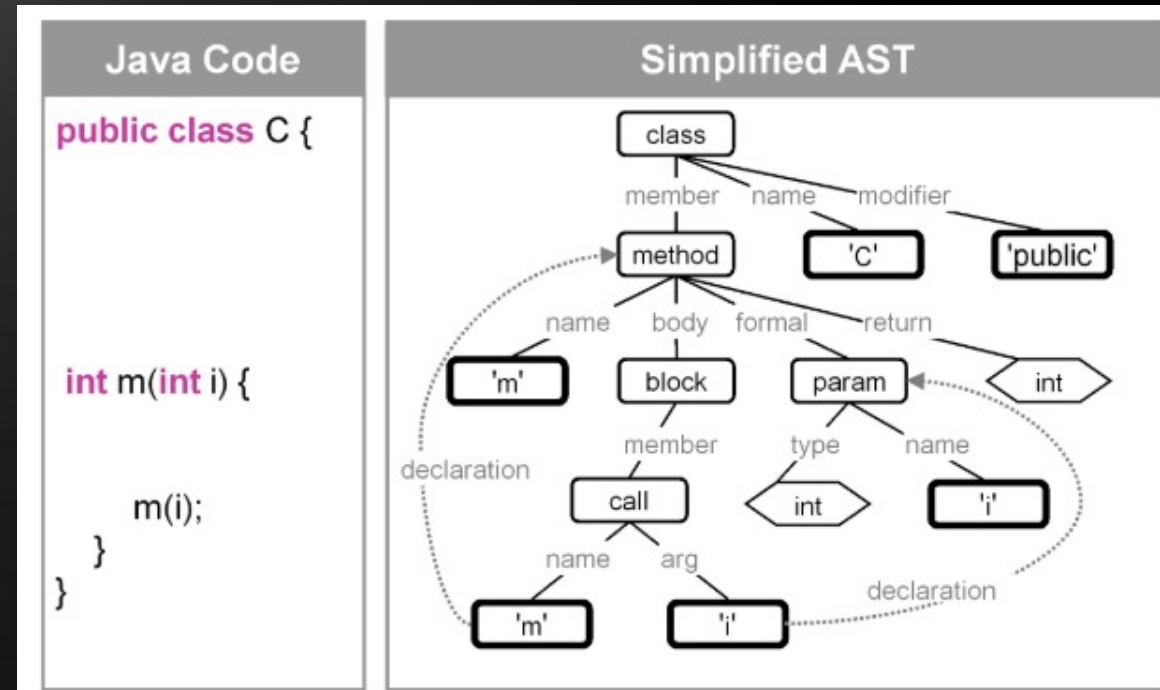
- ❖ All mathematical expression can also be parsed.



Parsing and Source Code

- ❖ In essence, source code is nothing but some text.
- ❖ The very first step to make sense out of source code is to parse it.
- ❖ In a simple word, a parser allows you to detect source code components and their relationships to each other.

```
45 ~ void printGraph(ArrayList<ArrayList<String>> mydata, int[][] matrix) {  
46 ~     mydata.forEach(l -> {  
47 ~         l.forEach(s -> {  
48 ~             System.out.println(s + "\t");  
49 ~         });  
50 ~     });  
51 ~     System.out.println();  
52 ~     System.out.println();  
53 ~     System.out.println();  
54 ~     System.out.println();  
55 ~     System.out.println();  
56 ~     for (int i = 0; i < matrix.length; i++) {  
57 ~         for (int j = 0; j < matrix.length; j++) {  
58 ~             System.out.print(matrix[i][j] + "\t");  
59 ~         }  
60 ~         System.out.println();  
61 ~     }  
62 ~     System.out.println();  
63 ~     System.out.println();  
64 ~ }  
65 ~  
66 ~ public GraphDatabaseService GraphNode4j() throws IOException {  
67 ~     //  
68 ~     GraphDatabaseFactory dbFactory = new GraphDatabaseFactory();  
69 ~     GraphDatabaseService db = dbFactory.newEmbeddedDatabase("DataVisualization/TPnew4jDB");  
70 ~     Transaction tx = db.beginTx();  
71 ~     tx.success();  
72 ~     Map<String, Long> allNodes = new HashMap<String, Long>();  
73 ~     try (InputStream input = DBDataPrepEdgeListDirectedToJson.class.getResourceAsStream("/database/gr.txt");  
74 ~         Scanner in = new Scanner(input)) {  
75 ~         while (in.hasNextLine()) {  
76 ~             String[] bits = in.nextLine().split("\t");  
77 ~             String nodeName = bits[0], replaceAll("\t", "");  
78 ~             if (allNodes.containsKey(nodeName)) {  
79 ~                 Node node = db.createNode();  
80 ~                 //System.out.println(node.getId());  
81 ~                 allNodes.put(nodeName, node.getId());  
82 ~             }  
83 ~             for (int i = 1; i < bits.length; i++) {  
84 ~                 if (allNodes.containsKey(bits[i])) {  
85 ~                     node.createRelationshipTo(allNodes.get(bits[i]), RelTypes.CC);  
86 ~                 }  
87 ~             }  
88 ~             //System.out.println(node.getProperty("Name") + "\t" + db.getNodeById(allNodes.get(bits[i])));  
89 ~         }  
90 ~     }  
91 ~ }
```



Abstract Syntax Tree (AST)

- ❖ AST is a tree representation of the abstract syntactic structure of source code written in a programming language. Each node of the tree denotes a construct occurring in the source code.
- ❖ The syntax is "abstract" in the sense that it does not represent every detail appearing in the real syntax, but rather just the structural or content-related details.
- ❖ The Abstract Syntax Tree maps plain Java source code in a tree form.
 - ❖ This tree shows containment (parent-child or membership) relationships between nodes.
- ❖ A tree is more convenient and reliable to analyze and modify programmatically than text-based source.
- ❖ A tree representing Java code will have a root representing a whole file, with nodes connected to the root for all the top elements of a file, like import or class declarations. From a single class declaration, we could reach multiple nodes, representing the fields or the methods of the class.

More details on what we will be doing for this module

1. Working with Maven projects.
2. Choosing an appropriate parser. Java in this module.
3. Working with the parser to:
 1. Identify components of the source code from high granularity to low granularity.
 2. Identify the relations between components of the code. E.g., method call, class inheritance, variable usage etc...
4. Source Code quality: Comments, functions, OO principles, Error handling and many other quality metrics.
5. Code clone and dead-code identification.
6. Feature Location.



Thank you

To contact me you can either email me or send a canvas message

Farshad.toosi@mtu.ie