

ScaMaha:

A Tool for Parsing, Analyzing, and Visualizing Object-Oriented Software Systems

Dr. [Ra'Fat Al-Msie'deen](#)

Department of Software Engineering, Faculty of IT, Mutah University,
Mutah 61710, Karak, Jordan

rafatalmsiede@mutah.edu.jo

<https://rafat66.github.io/Al-Msie-Deen/>



➤ To cite this version:

R. Al-Msie'deen, "*ScaMaha: A Tool for Parsing, Analyzing, and Visualizing Object-Oriented Software Systems*," International Journal of Computing and Digital Systems, vol. 17, no. 1, pp. 1-20, 2025.

- URL:

https://www.researchgate.net/publication/386985118_ScaMaha_A_Tool_for_Parsing_Analyzing_and_Visualizing_Object-Oriented_Software_Systems



ScaMaha: A Tool for Parsing, Analyzing, and Visualizing Object-Oriented Software Systems

- ❖ **ScaMaha** *is a tool for parsing, analyzing, and visualizing OO software systems.*
- ❖ **Keywords**: Software engineering, Reverse engineering, Software re-engineering, Object-Oriented source code, Static code analysis, Software visualization, Software metrics, ScaMaha tool.
- ❖ **ScaMaha tutorial**:
 - https://drive.google.com/drive/folders/11_CltJ2pPq_1CAWswcjRsAZnhmQO6OQ8
- ❖ **ScaMaha: @github**
 - <https://github.com/rafat66/ScaMaha>

Abstract

- **Reverse engineering tools** are required to handle the complexity of software products and the unique requirements of many different tasks, like software analysis and visualization. Thus, reverse engineering tools should adapt to a variety of cases.
- **Static Code Analysis (SCA)** is a technique for analyzing and exploring software source code without running it.
- **Manual review** of software source code puts additional effort on software developers and is a tedious, error-prone, and costly job.
- This paper **proposes an original approach** (called **ScaMaha**) for Object-Oriented (OO) source code analysis and visualization based on SCA.
- **ScaMaha** is a modular, flexible, and extensible reverse engineering tool. ScaMaha revolves around a new meta-model and a new code parser, analyzer, and visualizer.

Abstract ...

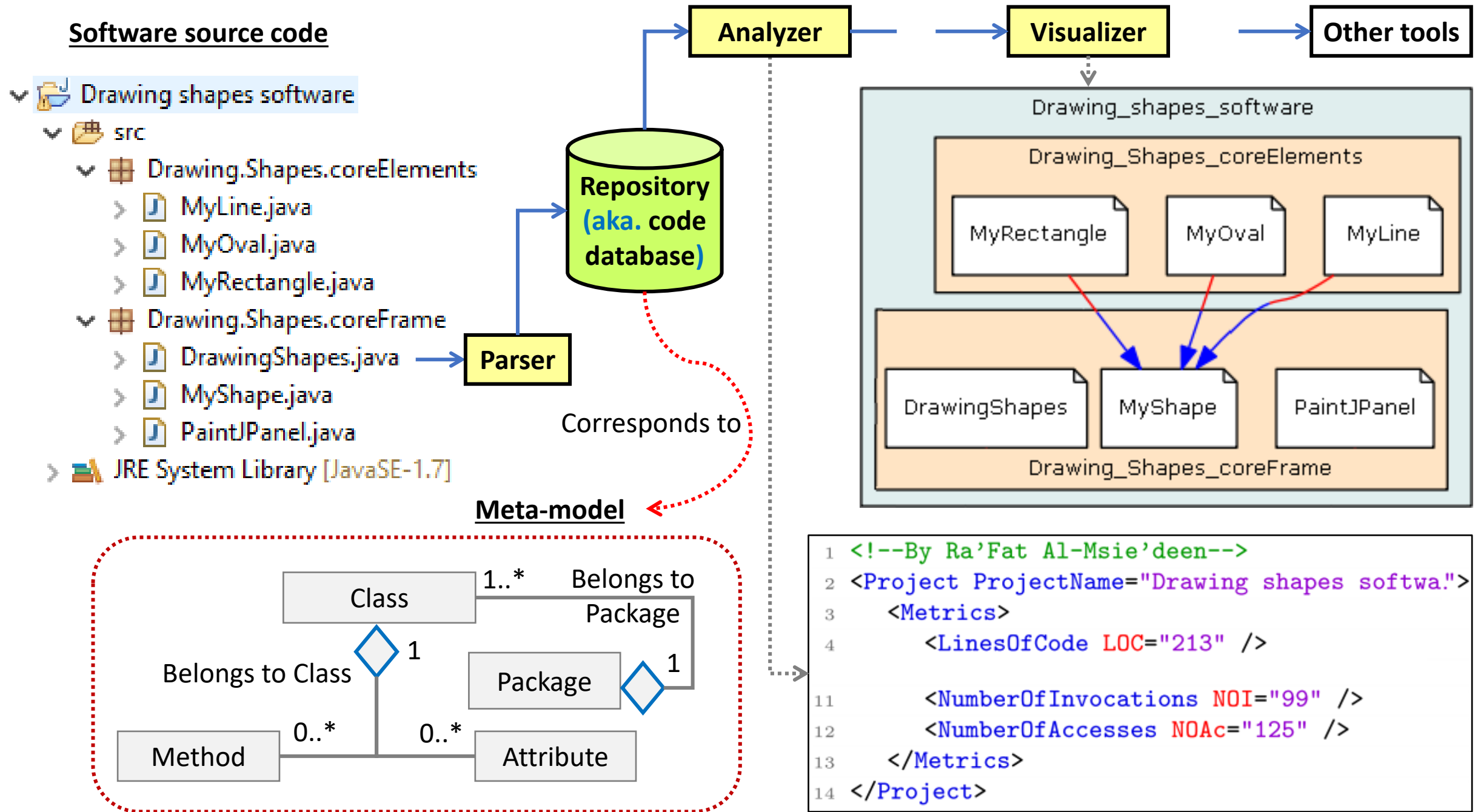
- ScaMaha parser extracts software source code based on the Abstract Syntax Tree (AST) and stores this code as a code file.
 - The code file includes all software code identifiers, relations, and structural information.
- ScaMaha analyzer studies and exploits the code files to generate useful information regarding software source code.
 - The software metrics file gives unique metrics regarding software systems, such as the number of method access relations.
- Software source code visualization plays an important role in software comprehension.
 - Thus, ScaMaha visualizer exploits code files to visualize different aspects of software source code.
 - The visualizer generates unique graphs about software source code, like the visualization of inheritance relations.

Abstract ...

- **ScaMaha tool** was applied to several case studies from small to large software systems, such as drawing shapes, mobile photo, health watcher, rhino, and ArgoUML.
- **Results** show the scalability, performance, soundness, and accuracy of ScaMaha tool.
- **Evaluation metrics**, like precision and recall, demonstrate the accuracy of ScaMaha in parsing, analyzing, and visualizing software source code, as all code artifacts (i.e., code file, software metrics file, and code visualizations) were **correctly extracted**.

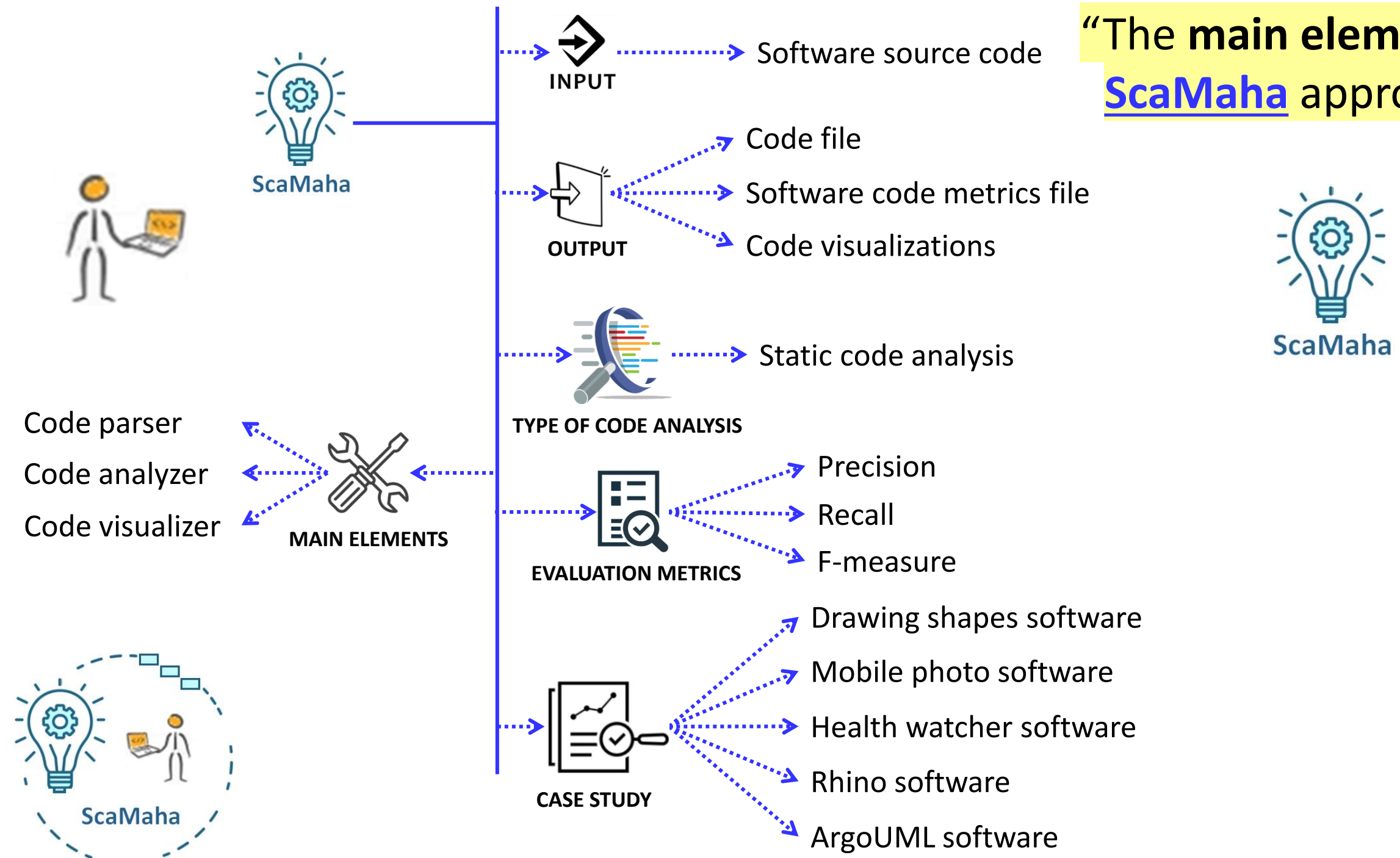
Introduction and Overview

- **Reverse engineering tools** are necessary to cope with the complexity of software systems.
- Also, such tools should cope with the specific requirements of the various reverse engineering tasks, like software comprehension and visualization. Thus, these tools should adapt to a wide range of cases.
- To analyze software code, tools need to represent it. Such a representation should be comprehensive. Software comprehension is still a manual activity. Thus, advanced tools will help developers fully understand complex systems.
- This paper presents **ScaMaha tool**, which is a **reverse engineering tool** for performing software analysis and visualization.
- The core of ScaMaha tool revolves around the meta-model, code parser, code analyzer, and code visualizer.
- With ScaMaha, tool developers can analyze and visualize software code.
- Also, developers can develop new, specific, and dedicated reverse engineering tools based on the infrastructure of ScaMaha tool.

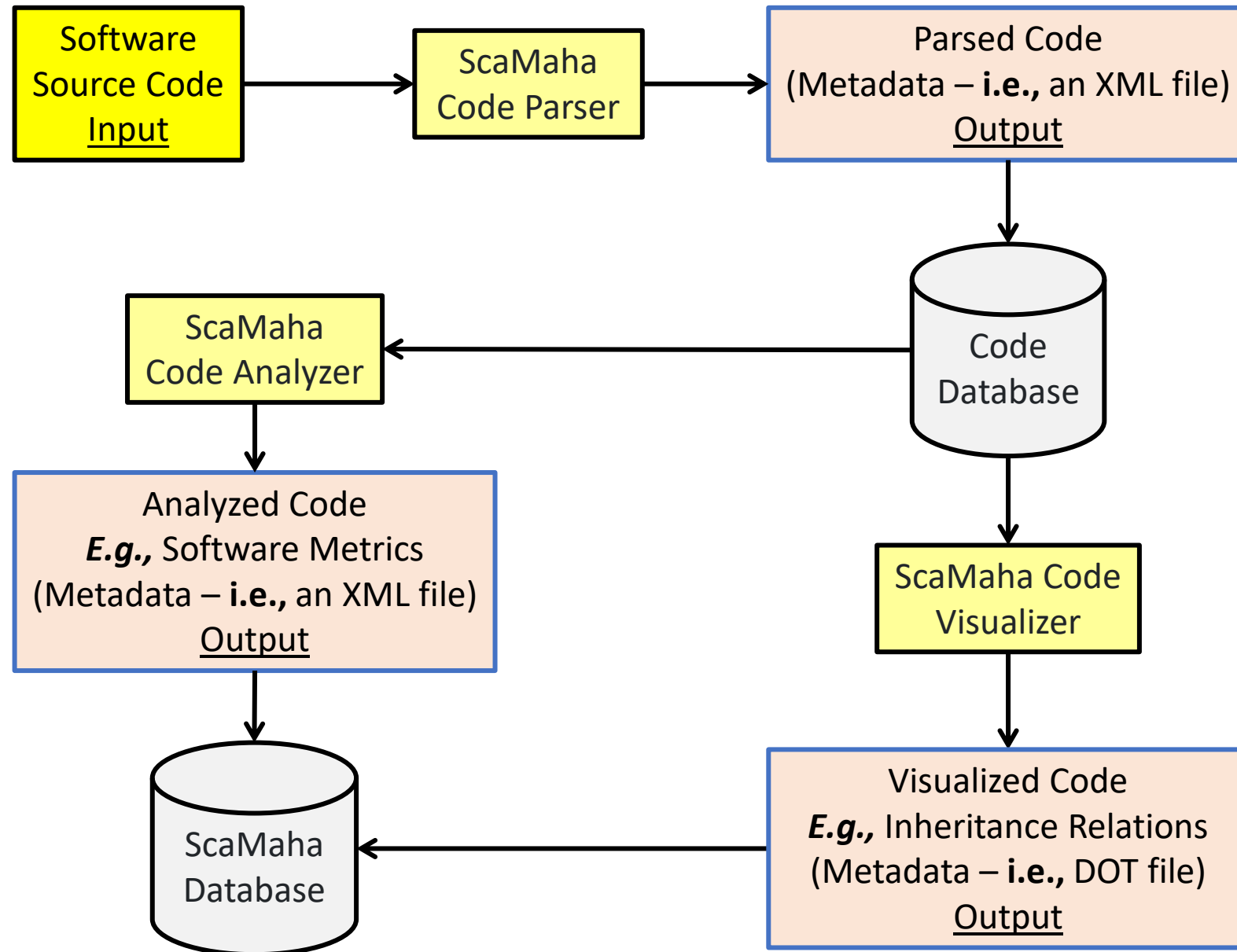


“Typical infrastructure for re-engineering tools”

**"The main elements of
ScaMaha approach."**

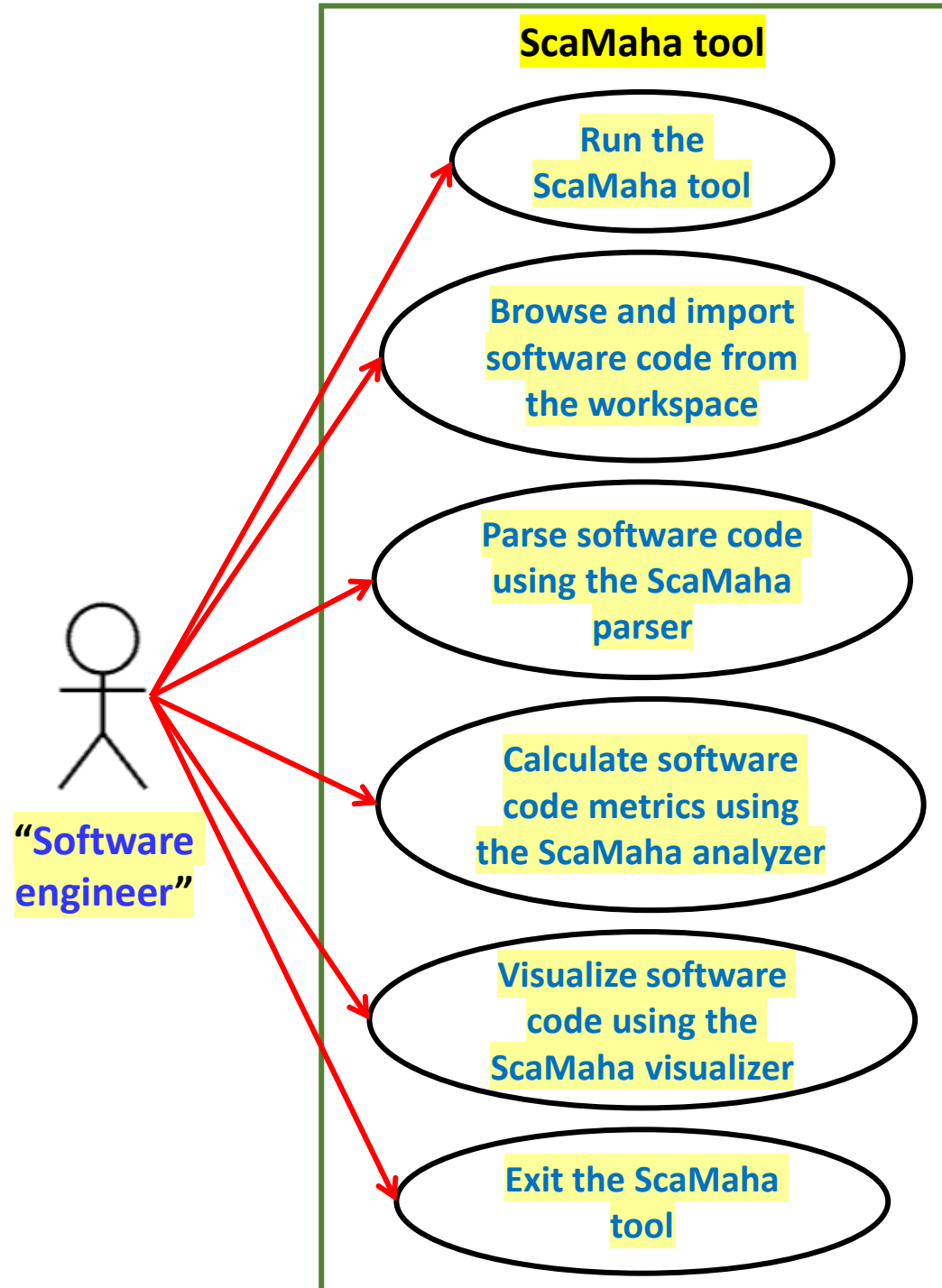


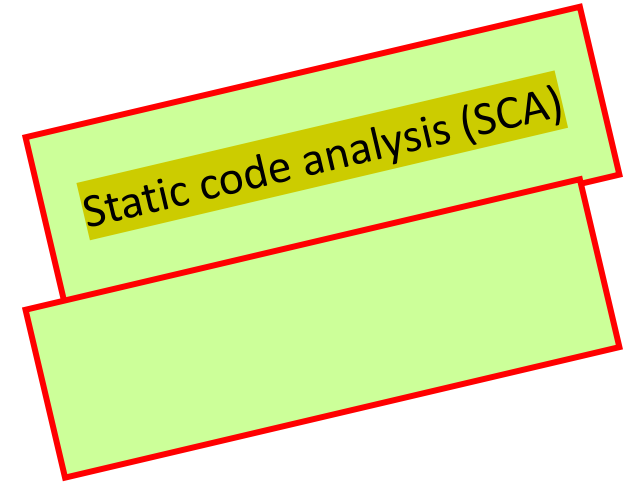
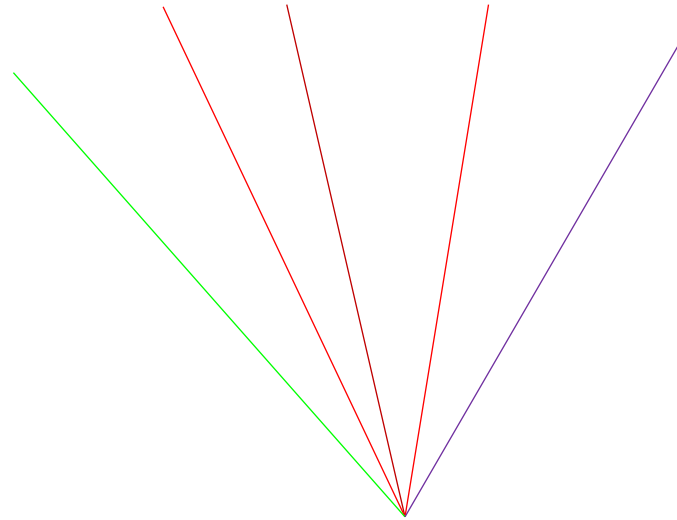
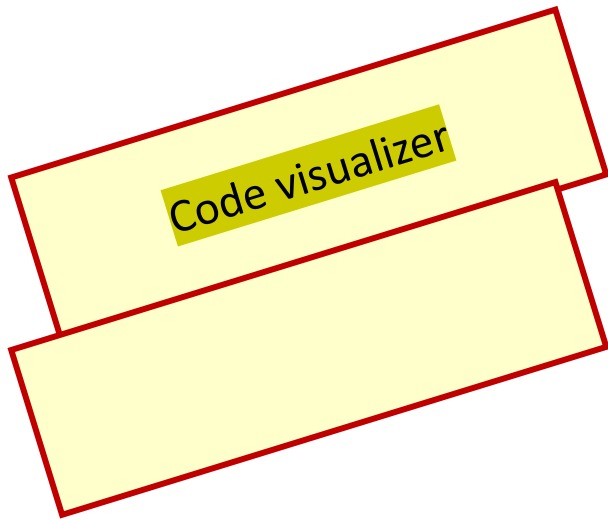
Use of Code Parser, Analyzer, and Visualizer in this Work



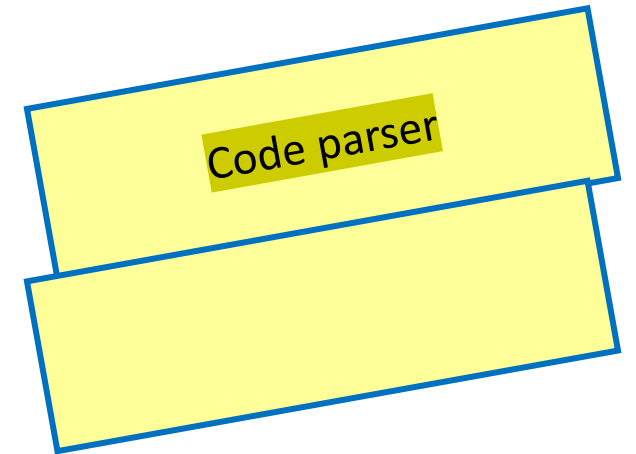
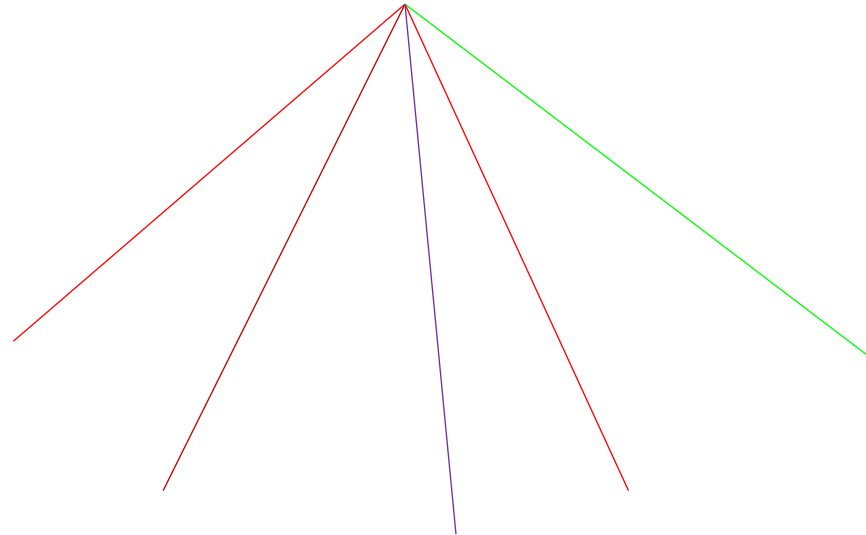
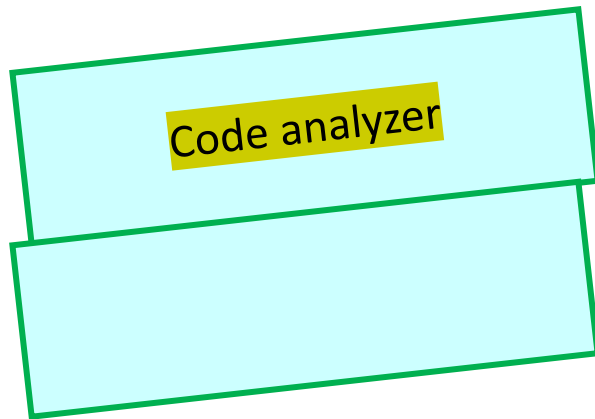
An **overview** of
ScaMaha tool

The use-case diagram of ScaMaha tool.





ScaMaha Tool



ScaMaha code parser

TABLE I. ScaMaha tool characteristics.

ScaMaha code parser									
Code entities (or identifiers)									
Package	Class			Attribute	Method				
	Comment	Super-class	Interface		Access level	Comment	Parameter list	Local variable	Exception
×	×	×	×	×	×	×	×	×	×
Code relations (or dependencies)									
Inheritance									✓
Attribute access									✓
Method invocation									✓

ScaMaha code analyzer

TABLE I. ScaMaha tool characteristics.

ScaMaha code analyzer	
Code metrics	
Lines of code	×
Number of packages	×
Number of classes	×
Number of attributes	×
Number of methods	×
Number of comments	×
Number of local variables	×
Number of inheritance relations	×
Number of attribute access relations	×
Number of method invocation relations	×

ScaMaha code visualizer

TABLE I. ScaMaha tool characteristics.

ScaMaha code visualizer	
Code visualizations	
Code organization	✓
Class inheritance relations	✓
Method invocation relations	✓
Polymetric view	✓
Tag cloud	✓

“Literature review”



Related Work

- **ScaMaha*** is a reverse engineering tool for parsing, analyzing, and visualizing software source code.
- **Moose** is a well-known reverse engineering tool [1]. It began as a research project around 24 years ago.
- **MODMOOSE** is the new version of Moose [2]. Tool developers can develop specialized reverse engineering tools with MODMOOSE. Moose was based on the **Famix** metamodel [3].

[ScaMaha] <https://github.com/rafat66/ScaMaha>

[1] O. Nierstrasz, S. Ducasse, and T. Gîrba, "The story of moose: an agile reengineering environment," in Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2005, Lisbon, Portugal, September 5-9, 2005, M. Wermelinger and H. C. Gall, Eds. ACM, 2005, pp. 1–10. [Online]. Available: <https://doi.org/10.1145/1081706.1081707>

[2] N. Anquetil, A. Etien, M. H. Houekpetodji, B. Verhaeghe, S. Ducasse, C. Toullec, F. Djareddir, J. Sudich, and M. Derras, "Modular moose: A new generation of software reverse engineering platform," in Reuse in Emerging Software Engineering Practices, S. Ben Sassi, S. Ducasse, and H. Mili, Eds. Cham: Springer International Publishing, 2020, pp. 119–134. https://link.springer.com/chapter/10.1007/978-3-030-64694-3_8

[3] S. Ducasse, N. Anquetil, M. U. Bhatti, A. C. Hora, J. Laval, and T. Gîrba, "MSE and FAMIX 3.0: an interexchange format and source code model family," 2011, [Online]. Available: <https://inria.hal.science/hal-00646884/document> (Accessed: Feb. 5, 2024).

Related Work ... [continued]

- Bruneliere *et al.* [4] suggested a semantic and syntax analysis-based parser for the creation of AST and metrics for multi-language software systems. Their meta-modeling tool [5] is utilized to analyze multi-language applications [6].

[4] H. Bruneliere, J. Cabot, F. Jouault, and F. Madiot, “Modisco: a generic and extensible framework for model driven reverse engineering,” in ASE 2010, 25th IEEE / ACM International Conference on Automated Software Engineering, Antwerp, Belgium, September 20-24, 2010, C. Pecheur, J. Andrews, and E. D. Nitto, Eds. ACM, 2010, pp. 173–174. [Online]. Available: <https://doi.org/10.1145/1858996.1859032>

[5] H. Bruneliere. (2010) MoDisco. [Online]. Available: <https://www.eclipse.org/MoDisco/> (Accessed: Feb. 5, 2024).

[6] H. Bruneliere, J. Cabot, G. Dupé, and F. Madiot, “Modisco: A model driven reverse engineering framework,” Inf. Softw. Technol., vol. 56, no. 8, pp. 1012–1032, 2014. [Online]. Available: <https://doi.org/10.1016/j.infsof.2014.04.007>

Related Work ... [continued]

- **VerveineJ** is a parser developed in Java that constructs an MSE file from software source code [7].
- Wettel and Lanza [8] have suggested an automatic tool called **CodeCity**. This tool visualizes software source code as a city metaphor. CodeCity is an interactive, three-dimensional software visualization tool [9].

[7] Moose VerveineJ. (2023) VerveineJ. [Online]. Available: <https://modularmoose.org/moose-wiki/Developers/Parsers/VerveineJ.html> (Accessed: Feb. 5, 2024).

[8] R. Wettel and M. Lanza, "Visualizing software systems as cities," in Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, VISSOFT 2007. IEEE Computer Society, 2007, pp. 92–99. [Online]. Available: <https://doi.org/10.1109/VISSOF.2007.4290706>

[9] D. Moreno-Lumbreras, R. Minelli, A. Villaverde, J. M. González-Barahona, and M. Lanza, "Codecity: A comparison of on-screen and virtual reality," Inf. Softw. Technol., vol. 153, p. 107064, 2023. [Online]. Available: <https://doi.org/10.1016/j.infsof.2022.107064>

Related Work ... [continued]

- The Java language has a wide variety of parsers due to its long history of development, popularity, and huge number of applications nowadays.
- Numerous tools exist that turn software code into a tree-like structure, such as interpreters and compilers.
- There are several Java parsers for various contexts because there are so many different Java applications, such as **Spoon** [10], **SrcML** [11], and **SuperParser** [12].

[10] R. Pawlak, M. Monperrus, N. Petitprez, C. Noguera, and L. Seinturier, “SPOON: A library for implementing analyses and transformations of java source code,” *Softw. Pract. Exp.*, vol. 46, no. 9, pp. 1155–1179, 2016. [Online]. Available: <https://doi.org/10.1002/spe.2346>

[11] M. L. Collard, M. J. Decker, and J. I. Maletic, “SrcML: An infrastructure for the exploration, analysis, and manipulation of source code: A tool demonstration,” in 2013 IEEE International Conference on Software Maintenance, Eindhoven, The Netherlands, September 22-28, 2013. IEEE Computer Society, 2013, pp. 516–519. [Online]. Available: <https://doi.org/10.1109/ICSM.2013.85>

[12] I. Utkin, E. Spirin, E. Bogomolov, and T. Bryksin, “Evaluating the impact of source code parsers on ML4SE models,” *CoRR*, vol. abs / 2206.08713, 2022. [Online]. Available: <https://doi.org/10.48550/arXiv.2206.08713>

Related Work ... [continued]

- The methods of software source code visualization have become increasingly utilized to support software engineers in software understanding. In software visualization, some methods aim at displaying software source code in a recognized environment, like a **forest** [13] or a **city**.
- Another method is to generate what is called a **polymetric view** [14], described as a lightweight visualization technique supplemented with several metrics regarding software code.
- **ScaMaha** visualizes different aspects of software source code, such as code organization and relations.

[13] U. Erra and G. Scanniello, "Towards the visualization of software systems as 3D forests: the codeTrees environment," in Proceedings of the ACM Symposium on Applied Computing, SAC 2012, Riva, Trento, Italy, March 26-30, 2012, S. Ossowski and P. Lecca, Eds. ACM, 2012, pp. 981–988. [Online]. Available: <https://doi.org/10.1145/2245276.2245467>

[14] M. Lanza and S. Ducasse, "Polymetric views - A lightweight visual approach to reverse engineering," IEEE Trans. Software Eng., vol. 29, no. 9, pp. 782–795, 2003. [Online]. Available: <https://doi.org/10.1109/TSE.2003.1232284>

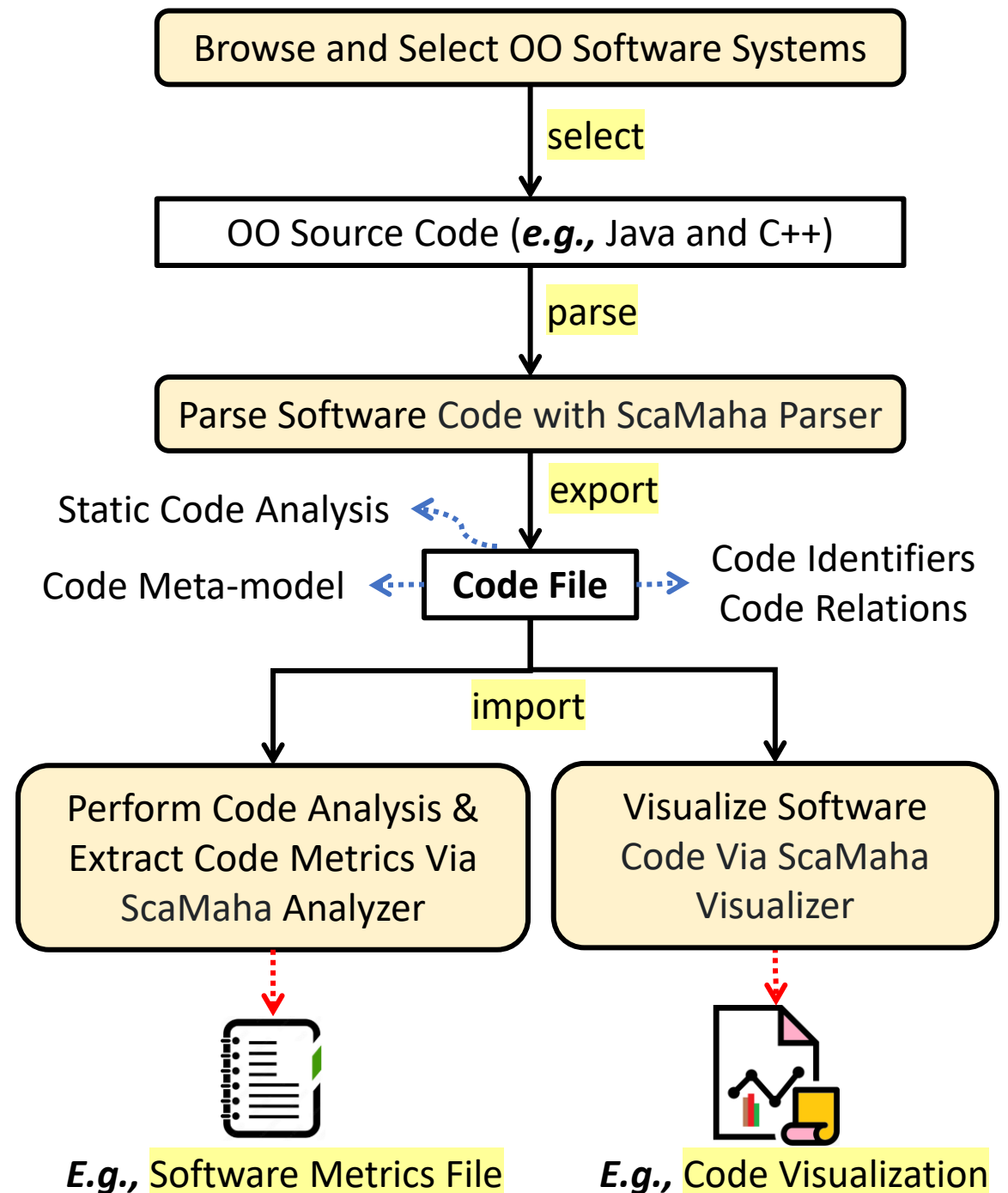
Related Work ... [continued]

- Lyons *et al.* [15] have suggested the **lightweight** multilingual software analysis method to analyze software systems. They use several code parsers, one for every programming language. The main goal of this work is to create a software engineering tool that addresses large and complex software systems in a lightweight and extensible style.
- The current version of the **ScaMaha** tool uses only one code parser for the Java language.

[15] D. M. Lyons, A. M. Bogar, and D. Baird, "Lightweight multilingual software analysis," in Proceedings of the 12th International Conference on Software Technologies, ICSOFT 2017, Madrid, Spain, July 24-26, 2017, J. Cardoso, L. A. Maciaszek, M. van Sinderen, and E. Cabello, Eds. SciTePress, 2017, pp. 201–207. [Online]. Available: <https://doi.org/10.5220/0006392502010207>

“ScaMaha tool”

Analyzing and visualizing OO source code with ScaMaha tool



The core of ScaMaha meta-model

- **ScaMaha** operates on a model of software source code, namely ScaMaha model.
- To analyze the OO software system, you must first create a model of it using ScaMaha.
- Figure “The core of ScaMaha meta-model” shows the core of ScaMaha meta-model.
- This meta model shows the **main OO software identifiers**, like software packages, classes, attributes, and methods.

The core of ScaMaha meta-model

- **Moreover**, this meta model displays the **main OO software relationships**, such as inheritance, method invocation, and attribute access.
- The core of ScaMaha involves a language-independent meta-model that can **show several OO languages in a uniform style**.
- In most cases, the developer gets sufficient information if he explores the basic types of entities that model an OO software system. These are code identifiers (e.g., **package**) and the relationships (e.g., **inheritance**) between them.

Method Invocation

Attribute Access

Inheritance

The core of ScaMaha meta-model

Package

Name

Attribute

Name

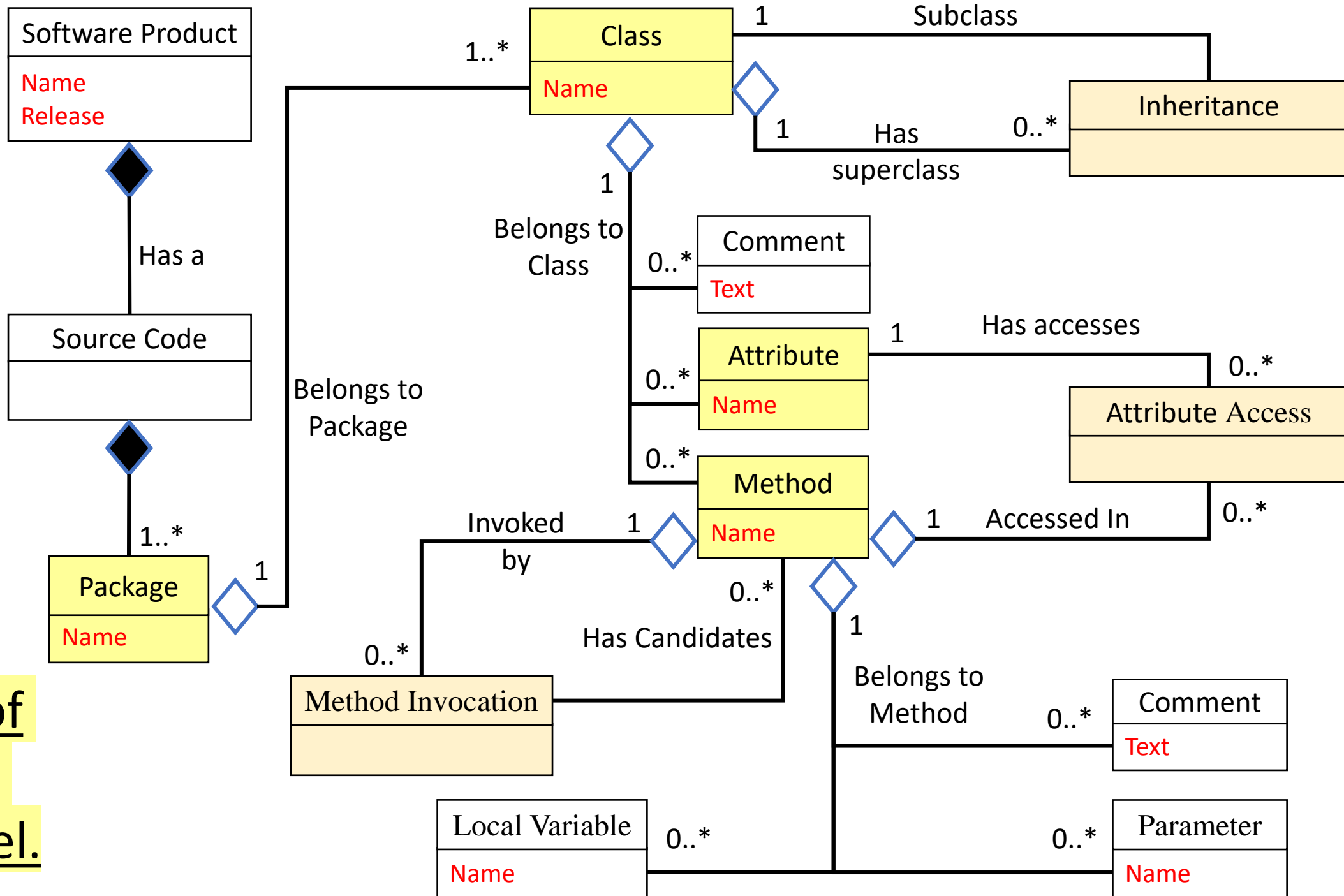
Class

Name

Method

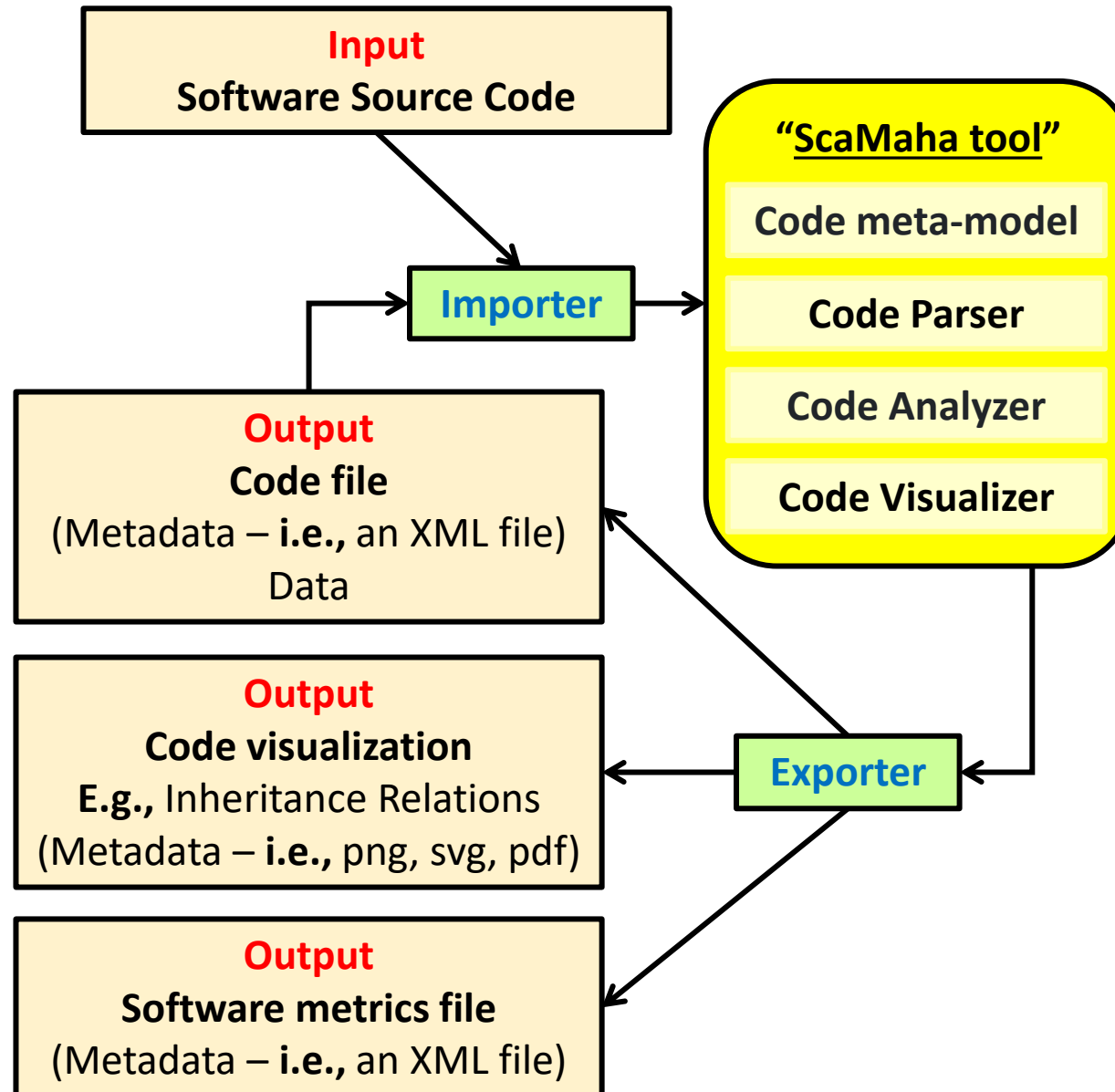
Name

The core of
ScaMaha
meta-model.

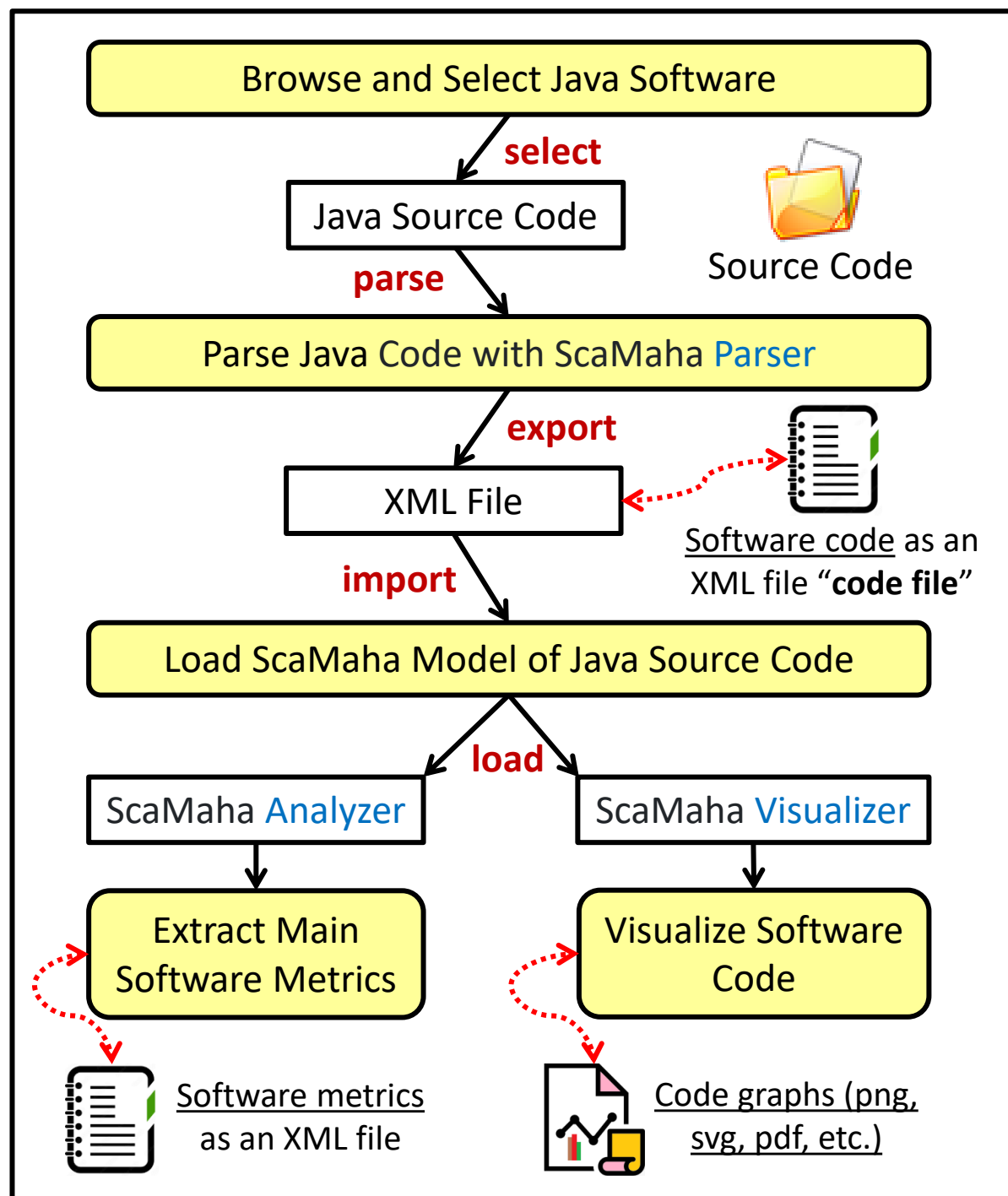


ScaMaha tool - An overview of the core parts of the ScaMaha tool

An overview of the
core parts of
ScaMaha tool.



**The Java
implementation of
ScaMaha tool.**



XML format corresponding to ScaMaha meta-model

- In this work, **XML** is a generic file format that represents ScaMaha code model.
- Thus, ScaMaha parser converts OO source code to an XML file format.
- ScaMaha parser produces an XML file for each software product.
- This file includes all code entities (or identifiers) and relationships (or dependencies) between those entities.
- Listing “**XML format corresponding to ScaMaha meta-model**” shows the XML format corresponding to ScaMaha meta-model.

XML format corresponding to ScaMaha meta-model.

```
1 <!--By Ra'fat Al-Msie'deen-->
2 <Project Name="-----">
3   <Packages>
4     <Package Name="-----">
5       <Classes>
6         <Class Name="-----" AccessLevel="-----" isInterface="-----" Superclass="-----">
7           <SuperInterfaces>
8             <Interface InterfaceName="-----" />
9           </SuperInterfaces>
10          <Comments>
11            <Comment CommentText="-----" />
12          </Comments>
13          <Attributes>
14            <Attribute Name="-----" AccessLevel="-----" Type="-----" isStatic="-----" />
15          </Attributes>
16          <Methods>
17            <Method Name="-----" AccessLevel="-----" ReturnType="-----" isStatic="-----">
18              <Parameters NumberOfParameters="-----">
19                <Parameter Name="-----" Type="-----" />
20              </Parameters>
21              <Comments>
22                <Comment CommentText="-----" />
23              </Comments>
24              <LocalVariables>
25                <LocalVariable Name="-----" Type="-----" />
26              </LocalVariables>
27              <AttributeAccesses>
28                <Access Name="-----" Type="-----" HowIsItUsed="-----" />
29              </AttributeAccesses>
30              <MethodInvocations>
31                <MethodInvocation Name="-----" Arguments="-----" />
32              </MethodInvocations>
33              <MethodAssignments>
34                <Assignment LeftHandSide="-----" RightHandSide="-----" />
35              </MethodAssignments>
36              <MethodExceptions>
37                <Exception ExceptionType="-----" />
38              </MethodExceptions>
39            </Method>
40          </Methods>
41        </Class>
42      </Classes>
43    </Package>
44  </Packages>
45 </Project>
```


The **code file** of
mobile photo
software as an XML
file (**partial**) [16].

```
1 <!--By Ra'Fat Al-Msie'deen-->
2 <Project Name="Mobile photo software">
3   <Packages>
4     <Package Name="ubc">
5       <Classes/>
6     </Package>
7     <Package Name="ubc.midp.mobilephoto.core">
8       <Classes/>
9     </Package>
10    <Package Name="ubc.midp.mobilephoto.core.threads">
11      <Classes>
12        <Class Name="BaseThread" AccessLevel="public" isInterface="false" Superclass="Object">
13          <Comments>
14            <Comment CommentText="Start the thread running"/>
15          </Comments>
16          <Attributes />
17          <Methods>
18            <Method Name="BaseThread" AccessLevel="public" ReturnType="void" isStatic="false">
19              <Parameters NumberOfParameters="0"/>
20              <LocalVariables />
21              <AttributeAccesses />
22              <MethodInvocations>
23                <MethodInvocation Name="println" Arguments="[BaseThread:: 0 Param Constructor used ... ]"/>
24              </MethodInvocations>
25              <MethodAssignments />
26              <MethodExceptions />
27            </Method>
28            <Method Name="run" AccessLevel="public" ReturnType="void" isStatic="false">
29              <Parameters NumberOfParameters="0"/>
30              <LocalVariables/>
31              <AttributeAccesses/>
32              <MethodInvocations>
33                <MethodInvocation Name="println" Arguments="[Starting BaseThread::run()]" />
34              </MethodInvocations>
35              <MethodAssignments/>
36              <MethodExceptions/>
37            </Method>
38          </Methods>
39        </Class>
40      </Classes>
41    </Package>
42  </Packages>
43 </Project>
```

Software code metrics of ScaMaha code analyzer

TABLE II. Software code metrics of ScaMaha code analyzer.

Metric	Abbreviation
Lines of Code	LOC
Number of Packages	NOP
Number of Classes	NOC
Number of Attributes	NOA
Number of Methods	NOM
Number of Comments	NOC _o
Number of local variables	NOL _v
Number of inheritance relations	NOIn
Number of attribute access relations	NOAc
Number of method invocation relations	NOI

Software code metrics for Rhino software

```
1 <!--By Ra'Fat Al-Msie'deen-->
2 <Project ProjectName="Rhino software">
3   <Metrics>
4     <LinesOfCode LOC="139408" />
5     <NumberOfPackages NOP="11" />
6     <NumberOfClasses NOC="167" />
7     <NumberOfAttributes NOA="1854" />
8     <NumberOfMethods NOM="2301" />
9     <NumberOfComments NOCo="4247" />
10    <NumberOfLocalVariables NOLv="4447" />
11    <NumberOfInheritances NOIn="146" />
12    <NumberOfInvocations NOI="10763" />
13    <NumberOfAccesses NOAc="42227" />
14  </Metrics>
15 </Project>
```

Listing 8. Software code metrics for Rhino software.

ScaMaha tool in a nutshell

- ScaMaha is a tool for software code analysis and visualization.
- Also, it is a creative tool to navigate, analyze, and visualize OO software systems.
- Moreover, ScaMaha helps software developers to cheaply construct custom analyses of software code.
- ScaMaha implementation is based on Java, and it's an open-source tool.

ScaMaha tool in a nutshell

- ScaMaha can't be used to analyze the dynamic execution of software.
- So, it uses SCA, which is a method of exploring the software code without running it.
- The current version of ScaMaha is shipped with a Java code parser.
- Also, this tool uses the XML structure to represent software code.
- Thus, XML is a file format that describes ScaMaha meta-model.
- XML is generated by an internally provided code parser.
- In this work, ScaMaha meta-model is an abstract representation of software source code.

ScaMaha tool in a nutshell

- In general, it presents software code entities and relations.
- The code meta-model of ScaMaha is a generic model, and it may describe software systems written in Java and C ++ (i.e., OO languages).
- Also, ScaMaha tool includes a code analyzer and visualizer.
- The code analyzer (resp. visualizer) uses the generated XML file from the code parser.
- ScaMaha analyzer (resp. visualizer) generates software code metrics (resp. visualizations).

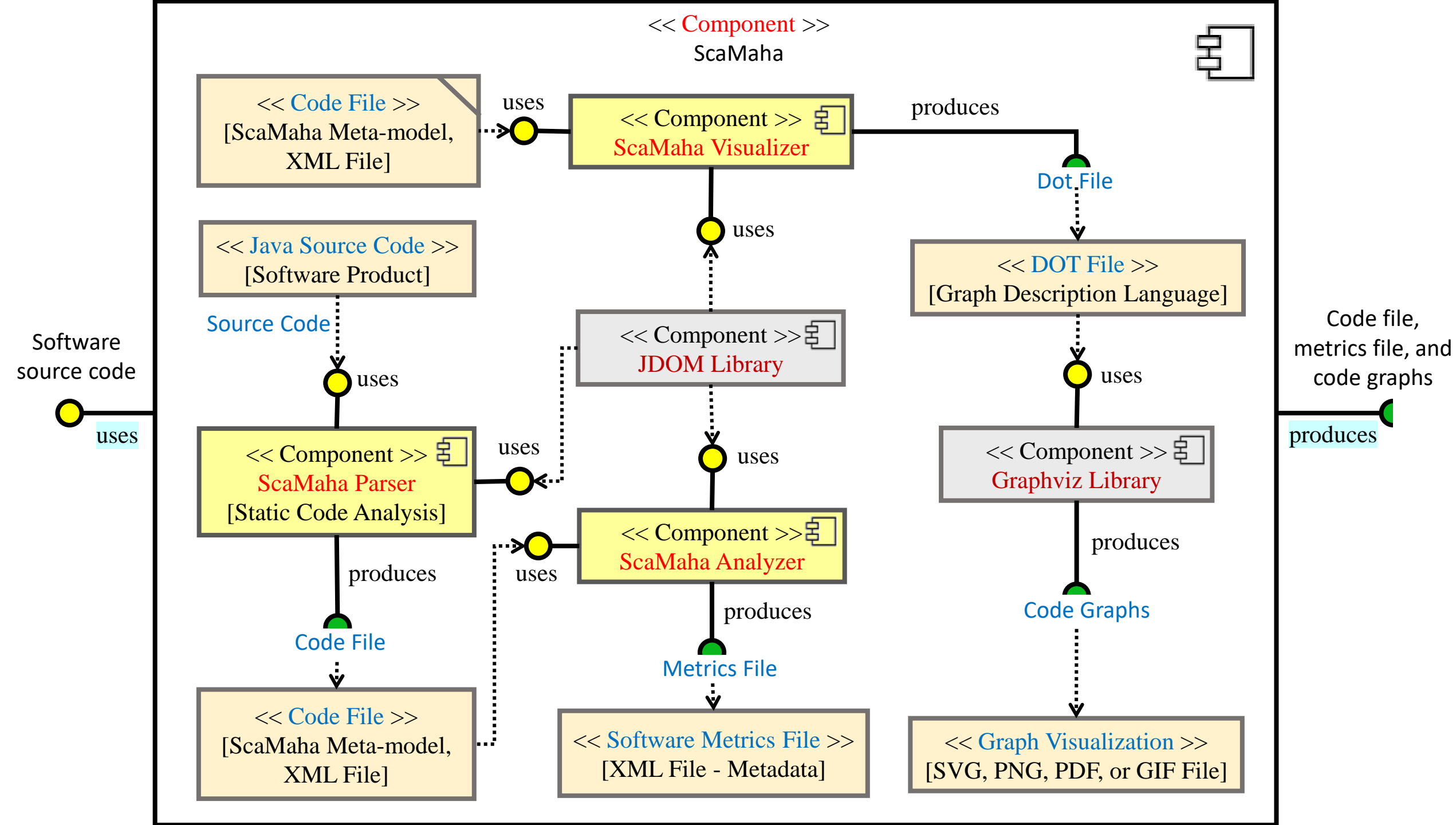
ScaMaha tool in a nutshell

- ScaMaha visualizer is an integral part of the tool that allows developers to visualize dependencies between classes and methods.
- In addition, ScaMaha visualizer gives a **polymetric view** of software based on its **packages**.
- Additionally, ScaMaha Visualizer generates **tag clouds** derived from software source code, specifically from **software identifiers**.
- Also, it reveals, in a unique visualization, the organization (or structure) of software code.
- Software engineers view and explore the generated code artifacts via ScaMaha tool to understand and analyze the chosen software systems.

An architectural view of ScaMaha
tool in a simplified structure.

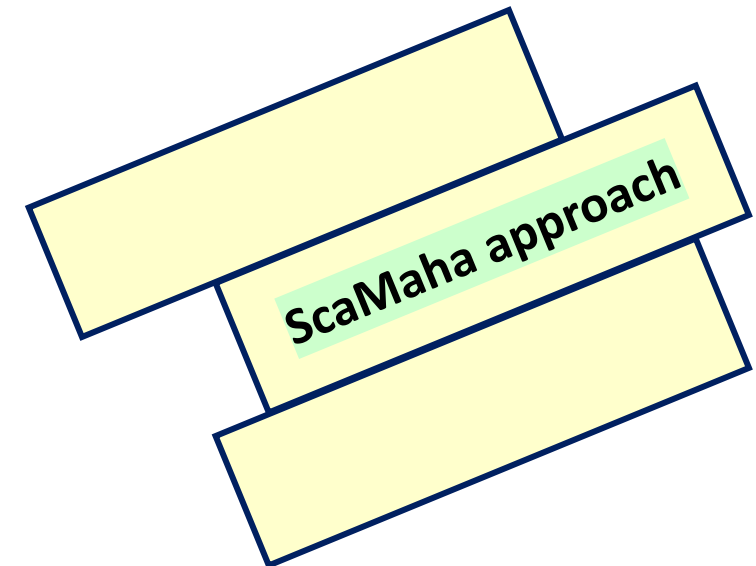
Software
source code

Code file,
metrics
file, and
code
graphs



Experimentation

ID	Case study	Software size
1	Drawing shapes	Small
2	Mobile photo	Medium
3	Health watcher	Medium
4	Rhino	Medium
5	ArgoUML	Large



Experimentation



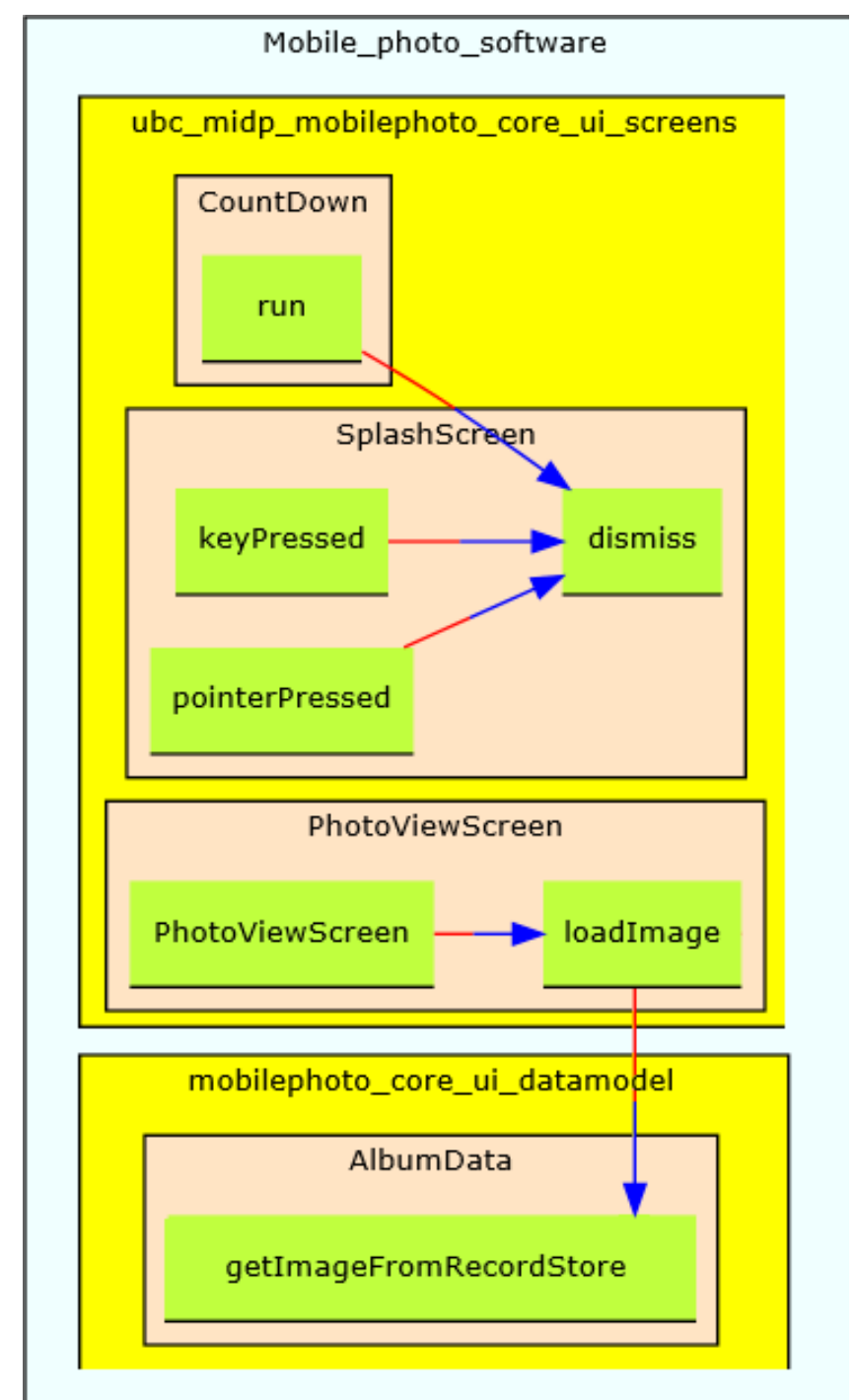
Ra'fat Al-Msie'deen: ScaMaha - A Tool for Parsing, Analyzing, and Visualizing OO Software Systems.

TABLE III. ScaMaha results for all experiments (*i.e.*, case studies).

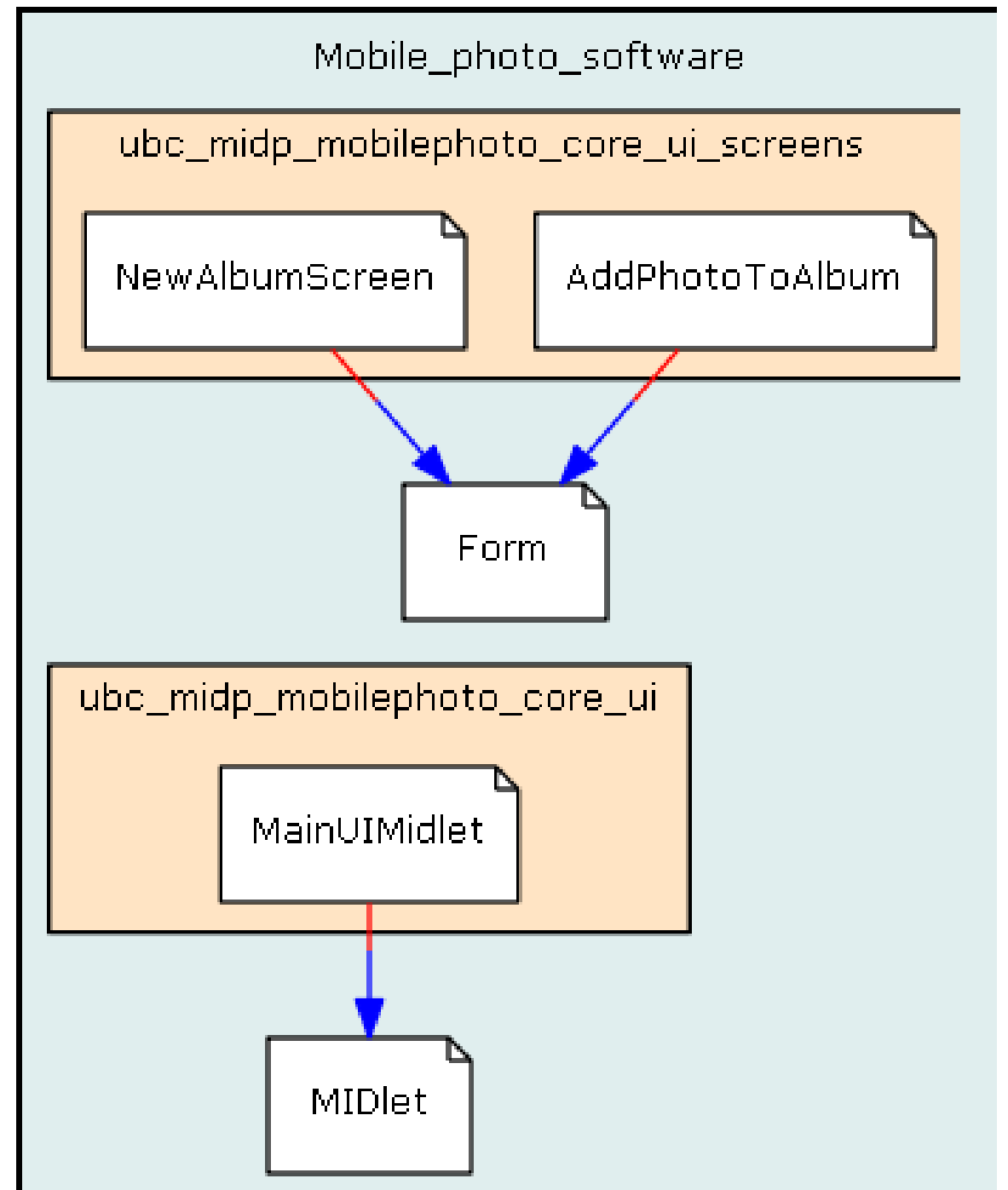
ID	Case study	Software size	Software artifacts		Execution time (in <i>ms</i>)	Visualization				
			Code	Metrics file		Code ^a	Class ^b	Method ^c	Polymetric ^d	Cloud ^e
1	Drawing shapes	Small	✓	✓	2095	×	×	×	×	×
2	Mobile photo	Medium	✓	✓	2850	×	×	×	×	×
3	Health watcher	Medium	✓	✓	5031	×	×	×	×	×
4	Rhino	Medium	✓	✓	7965	×	×	×	×	×
5	ArgoUML	Large	✓	✓	31698	×	×	×	×	×

^a Code organization, ^b Class inheritance relations, ^c Method invocation relations, ^d Polymetric view, ^e Tag cloud.

Visualization of **method invocation relations** for **mobile photo software** (partial).

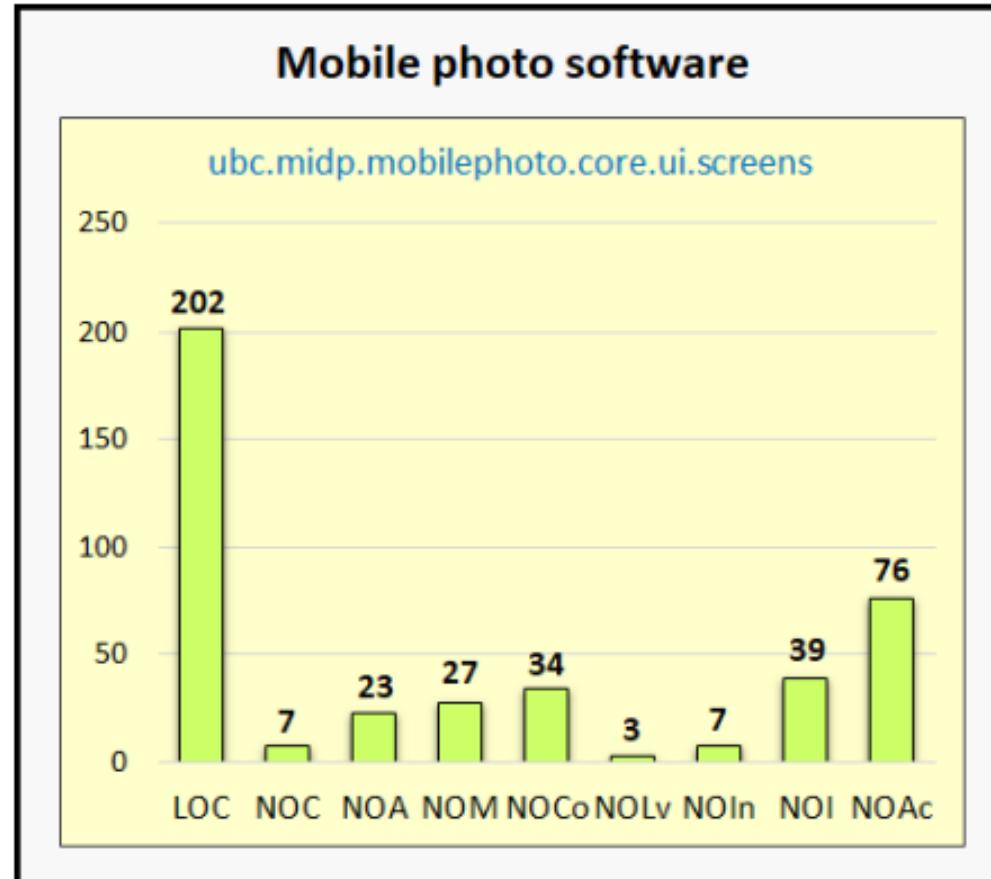


Visualization of **class inheritance relations** for **mobile photo software** (partial).

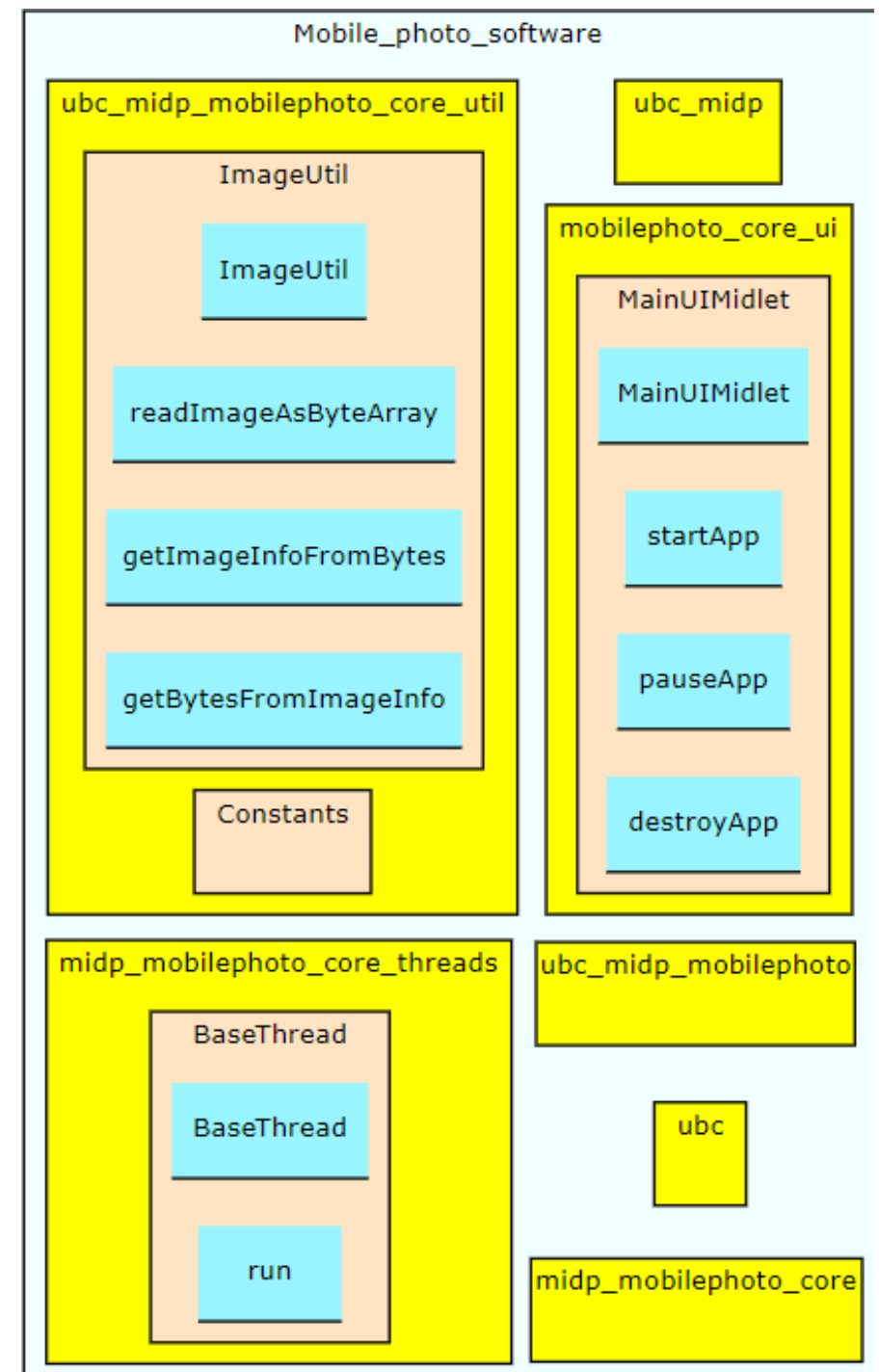


Polymetric view of mobile photo software based on its packages.

This view uses the following metrics for each package: LOC, NOC, NOA, NOM, NOCo, NOLv, NOIn, NOI, and NOAc.



Code organization visualization of mobile photo software (partial)



Software code metrics files

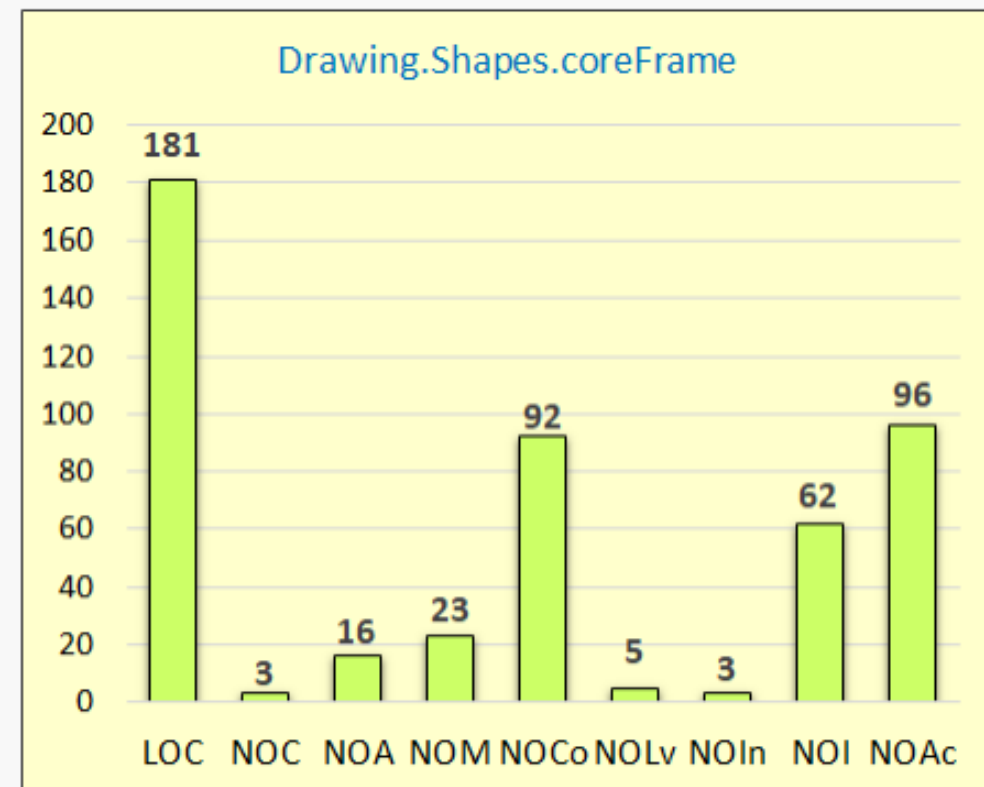
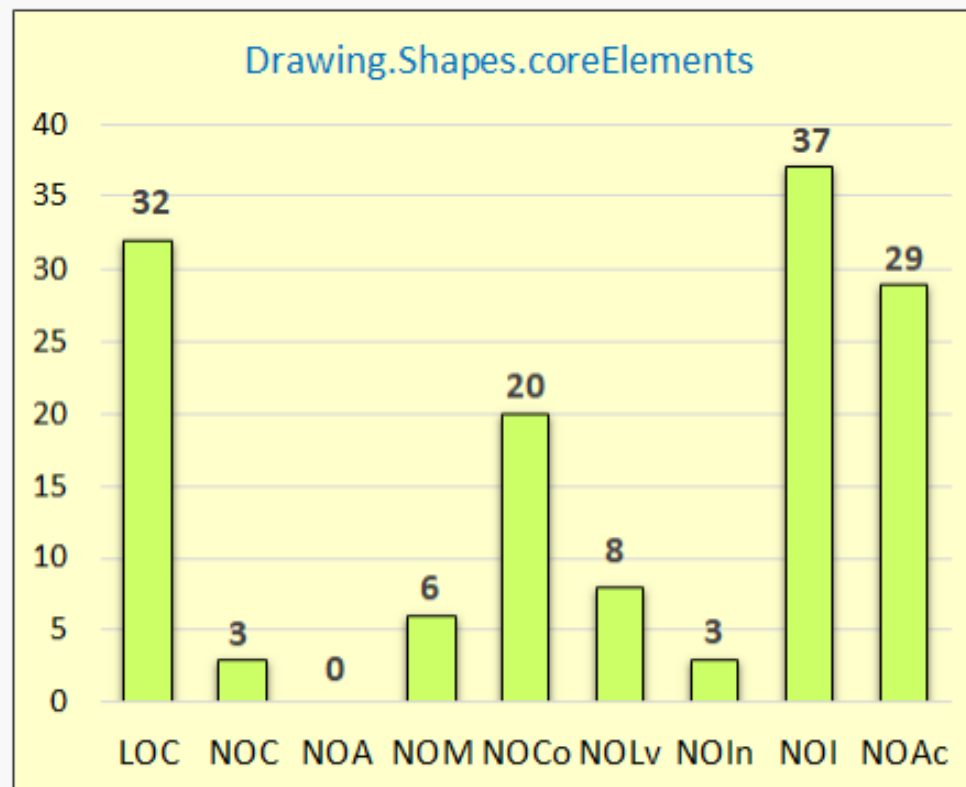
```
1 <!--By Ra'Fat Al-Msie'deen-->
2 <Project ProjectName="Drawing shapes software">
3   <Metrics>
4     <LinesOfCode LOC="213" />
5     <NumberOfPackages NOP="4" />
6     <NumberOfClasses NOC="6" />
7     <NumberOfAttributes NOA="16" />
8     <NumberOfMethods NOM="29" />
9     <NumberOfComments NOCo="112" />
10    <NumberOfInheritances NOIn="6" />
11    <NumberOfInvocations NOI="99" />
12    <NumberOfAccesses NOAc="125" />
13  </Metrics>
14 </Project>
```

Listing 5: Software code metrics for the drawing shapes software.

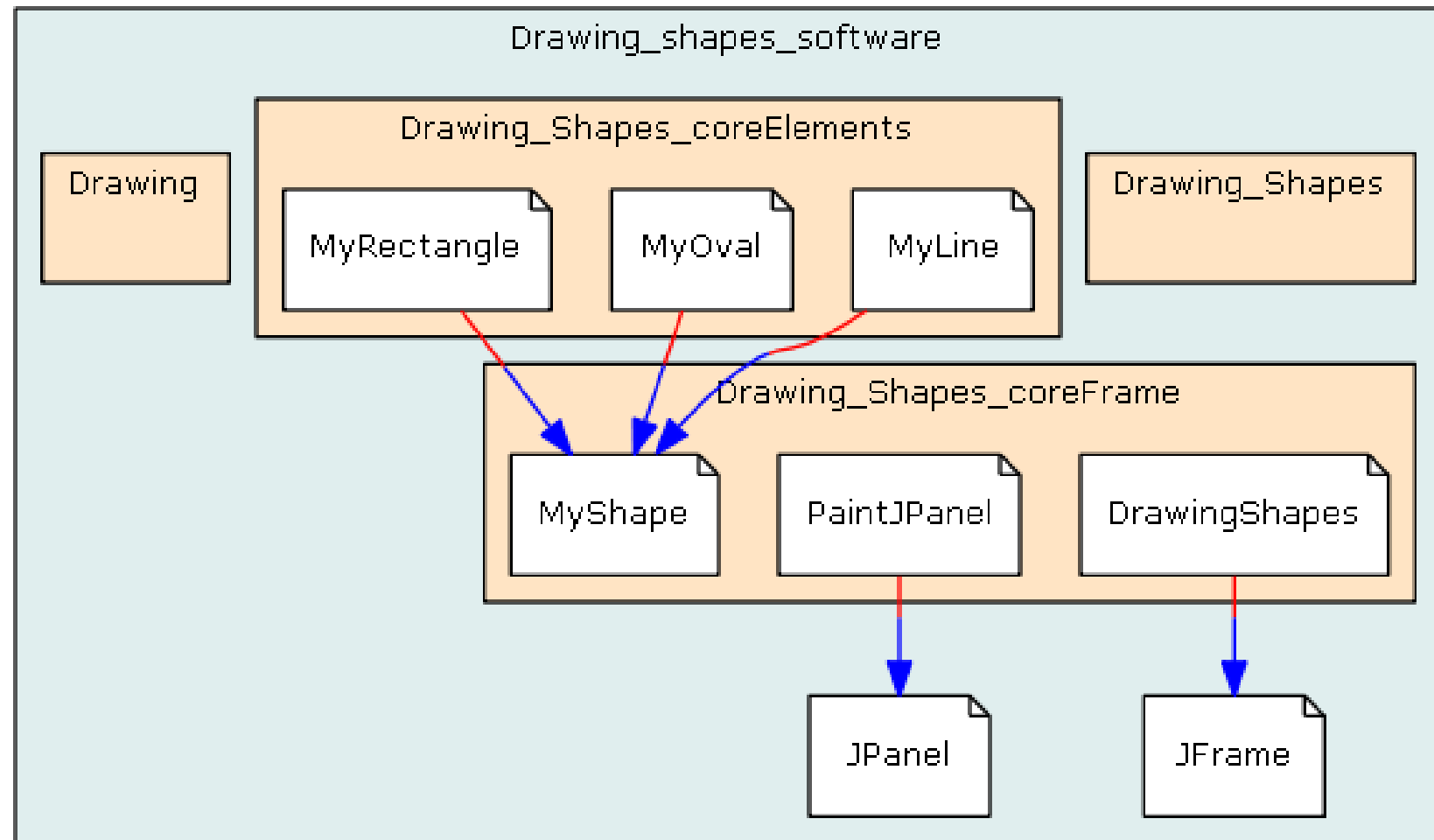
```
1 <!--By Ra'Fat Al-Msie'deen-->
2 <Project ProjectName="Health watcher software">
3   <Metrics>
4     <LinesOfCode LOC="8217" />
5     <NumberOfPackages NOP="29" />
6     <NumberOfClasses NOC="135" />
7     <NumberOfAttributes NOA="256" />
8     <NumberOfMethods NOM="894" />
9     <NumberOfComments NOCo="300" />
10    <NumberOfLocalVariables NOLv="602" />
11    <NumberOfInheritances NOIn="118" />
12    <NumberOfInvocations NOI="2703" />
13    <NumberOfAccesses NOAc="4465" />
14  </Metrics>
15 </Project>
```

Listing 7: Software code metrics for the health watcher software.

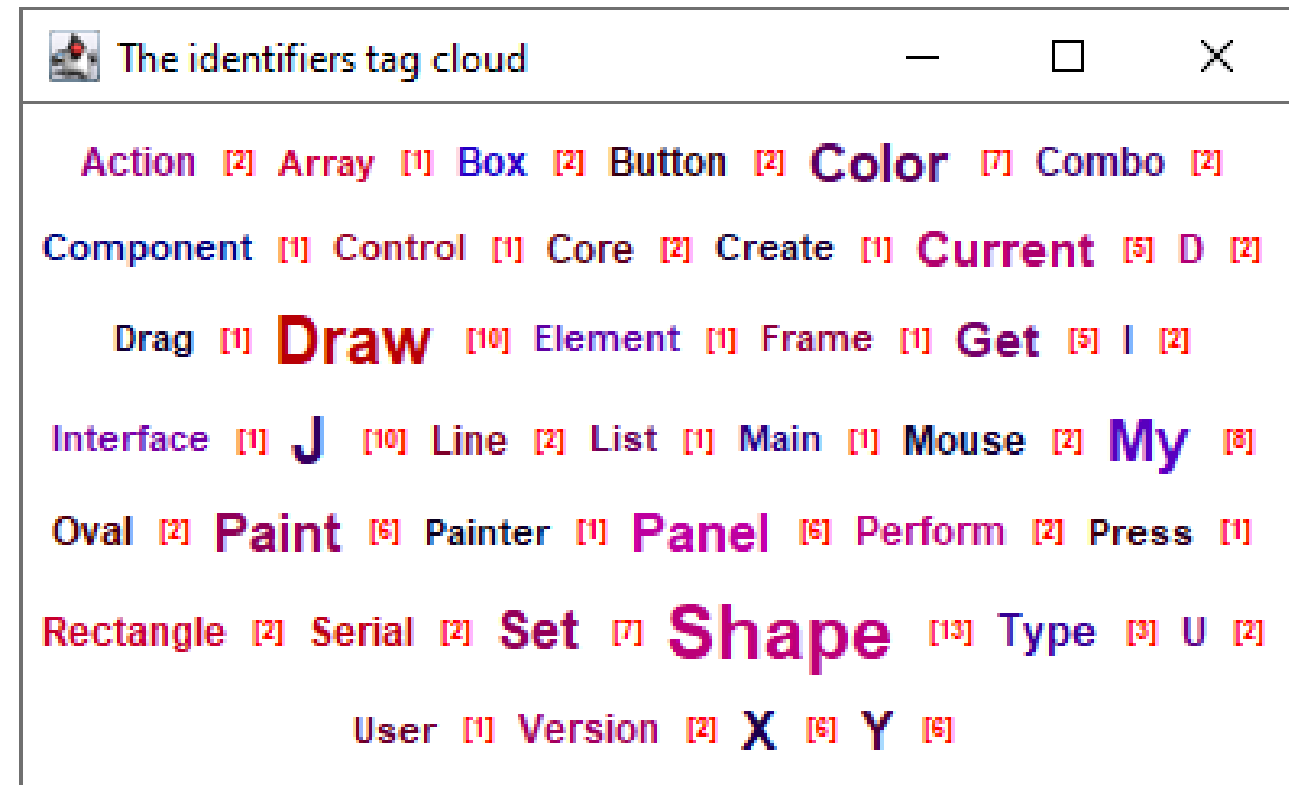
Polymetric view of drawing shapes software based on its packages. This view uses the following metrics for each package: LOC, NOC, NOA, NOM, NOCo, NOLv, NOIn, NOI, and NOAc.



Visualization of **class inheritance relations** for drawing shapes software.



Tag cloud extracted from the drawing shapes software using ScaMaha.



The figure above shows a **tag cloud** generated from the source code of the **drawing shapes software** using ScaMaha. It highlights key **software identifiers**, including package names, class names, attribute names, and method names. **Larger tags represent higher frequency and importance**, with **tag frequencies** displayed in red within square brackets.

The code file of drawing shapes software as an XML file (partial) [16].

```
1 <!--By Ra'fat Al-Msie'deen-->
2 <Project Name="Drawing shapes software">
3   <Packages>
4     <Package Name="Drawing.Shapes.coreElements">
5       <Classes>
6         <Class Name="MyLine" AccessLevel="public"
7           isInterface="false" Superclass="MyShape"
8           ">
9         <SuperInterfaces />
10        <Comments>
11          <Comment CommentText="Class that
12            declares a line object" />
13        </Comments>
14        <Attributes />
15        <Methods>
16          <Method Name="draw" AccessLevel="public"
17            " ReturnType="void" isStatic="
18            false">
19            <Parameters NumberOfParameters="1">
20              <Parameter Name="g" Type="Graphics"
21                />
22            </Parameters>
23            <LocalVariables />
24            <AttributeAccesses>
25              <AttributeAccess Name="g" Type="
26                Graphics" HowIsItUsed="g.
27                setColor(getColor())" />
28            </AttributeAccesses>
29            <MethodInvocations>
30              <MethodInvocation Name="setColor"
31                Arguments="[getColor()]" />
32            </MethodInvocations>
33            <MethodAssignments />
34            <MethodExceptions />
35          </Method>
36        </Methods>
37      </Class>
38    </Classes>
39  </Package>
40 </Packages>
41 </Project>
```

Conclusion

- This paper has presented ScaMaha, a tool for parsing, analyzing, and visualizing OO source code like Java.
- ScaMaha is designed to prevent software engineers from wasting their resources, like effort and time, on manual review of software source code in order to understand it.
- The main goal of ScaMaha tool is to assist software engineers in the process of understanding complex and large-sized software systems.
- Various categories of users, such as scholars in the field of software analysis and tool creators, will exploit ScaMaha tool in their works.

Conclusion ...

- In summary, ScaMaha extracts code identifiers and relationships, generates a metrics file with statistical data, and produces unique graphs to effectively visualize software code.
- ScaMaha had been validated and evaluated on several case studies, including drawing shapes, mobile photo, health watcher, rhino, and ArgoUML software.
- The results of the experiments show the capacity of ScaMaha to recover all software artifacts in an efficient and accurate manner.
- The evaluation metrics of ScaMaha, like precision and recall, show the accuracy of ScaMaha in parsing, analyzing, and visualizing software source code, as all source code artifacts were correctly obtained.

Future Work

- Regarding ScaMaha's future work, the author plans to extend the current tool by developing a comprehensive tool's parser for all OO languages, like Java and C++.
- Moreover, additional experimental tests can be performed to verify ScaMaha contributions utilizing open-source and industrial software systems.
- Also, the author plans to conduct a comprehensive survey regarding current approaches that relate to ScaMaha contributions.
- Finally, the author of ScaMaha tool plans to extend the current work by developing a general tool for performing various kinds of analyses and visualizations of any OO software system.

References

[16] R. A. A. Al-Msie'deen. (2023) ScaMaha. [Online]. Available: https://drive.google.com/drive/folders/11_CltJ2pPq_1CAWswcjRsAZnhmQO6OQ8 (Accessed: Feb. 5, 2024).

REFERENCES





- [1] N. Anquetil, A. Etien, M. H. Houekpetodji, B. Verhaeghe, S. Ducasse, C. Toullec, F. Djareddir, J. Sudich, and M. Derras, "Modular moose: A new generation of software reverse engineering platform," in *Reuse in Emerging Software Engineering Practices*, S. Ben Sassi, S. Ducasse, and H. Mili, Eds. Cham: Springer International Publishing, 2020, pp. 119–134.
- [87] S. A. Butt, M. Acosta-Coll, and S. Misra, "Software product maintenance: A case study," in *Computer Information Systems and Industrial Management - 21st International Conference, CISIM 2022, Barranquilla, Colombia, July 15-17, 2022, Proceedings*, ser. Lecture Notes in Computer Science, K. Saeed and J. Dvorský, Eds., vol. 13293. Springer, 2022, pp. 81–92. [Online]. Available: https://doi.org/10.1007/978-3-031-10539-5_6

SCAMAHA TOOL

A Tool for Parsing,
Analyzing, and
Visualizing Object-
Oriented Software
Systems

DR. RA'FAT AL-MSIE'DEEN



Ra'fat Al-Msie'Deen has been a Full Professor of Computer Science at Muthah University in Al-Karak, Jordan, since 2014. He received his PhD in Software Engineering from the Université de Montpellier, Montpellier - France, in 2014. He received his MSc in Information Technology from the University Utara Malaysia, Kedah - Malaysia, in 2009. He got his BSc in Computer Science from Al-Hussein Bin Talal University, Ma'an - Jordan, in 2007. His research interests include software engineering, requirements engineering, software product line engineering, feature identification, word clouds, and formal concept analysis. Dr. Al-Msie'Deen aimed to utilize his background and skills in the academic and professional fields to enhance students expertise in developing software systems. Contact him at rafatalmsiedeem@mutah.edu.jo. Also, you can reach him using different alternatives:  author's page @ github.io,  LinkedIn,  ResearchGate, or  Orcid.

ScaMaha:

A Tool for Parsing, Analyzing, and Visualizing Object-Oriented Software Systems

Dr. [Ra'Fat Al-Msie'deen](#)

Department of Software Engineering, Faculty of IT, Mutah University,
Mutah 61710, Karak, Jordan

rafatalmsiedeen@mutah.edu.jo

<https://rafat66.github.io/Al-Msie-Deen/>

