Object-Oriented Analysis and Design for one Algorithm of Computational Geometry: Forward, Reverse and Round-Trip Engineering

Muzafer H. Saračević, Predrag S. Stanimirović, Sead H. Mašović

Department of Computer Science, Faculty of Science and Mathematics,
University of Nis, Visegradska 33, 18000 Nis, Serbia.
muzafers@uninp.edu.rs, peckois@pmf.edu.rs ,sead.masovic@pmf.edu.rs

Case study

DOI: **10.7251/JIT1302096S** UDC: **378.046.4:004.41]:004.428**

Abstract: *Triangulation of the polygon is a fundamental algorithm in computational geometry. This paper considers techniques of object-oriented analysis and design as a new tool for solving and analyzing convex polygon triangulation. The triangulation is analyzed from three aspects: forward, reverse and round-trip engineering. We give a suggestion for improving the obtained software solution of the polygon triangulation algorithm using technique that combines UML modeling and Java programming.*

Keywords: *Software engineering, Computational geometry, Triangulation of Polygons, Modeling in UML, Java.*

Introduction

Triangulation enables to get a display of three-dimensional objects from a set of given points and provides a mechanism for so-called glazing of three-dimensional figures [8]. Polygon triangulation has many applications in computer graphics and it is used in the pre-trial phase of non-trivial operations of simple polygons [10]. Triangulation of convex polygons is an actual problem which appears in the two-dimensional computational geometry [14,19]. Triangulation of a convex polygon assumes decomposition of the polygon interior into triangles by internal diagonals that are not intersected.

Polygon triangulation is a complex problem that requires complex class for an efficient object-oriented implementation. In order that this class would be comprehensible, it is necessary to do their analysis. Dealing with the complexity of the same, there is a need for new techniques to develop alternative views and engineered for the field of object-oriented modelling.

This paper presents an object-oriented analysis and design (OOAD) based on Hurtado-Noy method for the triangulation of convex polygon, which is introduced in [11]. OOAD provides a comprehensive insight into the implementation of this problem.

We present analysis and design for the Hurtado-Noy method through three aspects, which can be briefly described as follows:

1. Direct development (*forward engineering*): this approach is based on generating the source code in a selected programming language from the *UML* model (Unified Modeling Language). In our case, we have choosen the programming language *Java*.
2. Feedback analysis (*reverse engineering*): it refers to the interpretation of the source code that is generated from defined *UML* models in a selected programming language.
3. Synchronization of feedback analysis and direct development (*round-trip engineering*):

this aspect investigates the synchronization between the source code changes and *UML* models.

Baselines and preliminaries

The number of all triangulations of a convex polygon with n vertices is equal to the $(n-2)$th Catalan number

$$C_{n-2} = \frac{(2n-4)!}{(n-1)!(n-2)!}, n \geq 3 \quad (1)$$

For more details about the convex polygon triangulation see for example [15].

Details of *Hurtado-Noy* method [11]: Let $T(n)$ be the set of triangulations of an n-gon. Every triangulation t that belongs to $T(n)$ has exactly one "predecessor" in $T(n-1)$ and one or more "descendants" in the set of triangulations $T(n+1)$. For a given set $T(n)$, there is a possibility to generate triangulations of the $(n+1)$-gon derived from arbitrary triangulation $t \in T(n)$. This principle is illustrated in Figure 1.

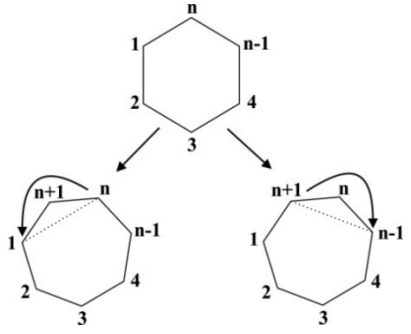

Figure 1. Forming the new triangulation for $(n+1)$-gon, according to *Hurtado-Noy method*

Algorithm 1 describes the Hurtado-Noy method from [11].

Based on the above principle of separation of predecessor, Hurtado and Noy provided the hierarchy which is important because of its inherent simplicity and also owing to the fact that it has a number of really exciting properties (see Figure 2, restated from [11]).

An implementation of this algorithm in *Java* programming language is presented in our paper [23]. In the mentioned implementation, the phase of coding on the basis of a given Algorithm 1 was performed without prior analyzing and creating a visual plan. This way of solving the problem can adversely affect the functionality and intelligibility of generated source code.

A better understanding and detailed analysis of Hurtado-Noy method is achieved applying forward engineering on the same algorithm. In addition, we get a developed visual model (plan for solving) which is independent of implementation and technology. This approach deals with the defined model that allows the transition to the phase of coding (programming). After finishing the programming phase, reverse and round-trip engineering play a key role in the maintenance and evolution of the obtained solution for Algorithm 1.

This paper is organized as follows. Section 2 describes the *UML* modeling process appropriate for generating triangulations of the convex n-gon. This section presents the possibility of generating Java source code from UML model. Also, in this section we presented Java experimental results obtained by the developed software solution. A method for improving already created software solutions in a selected programming language is given in the section 3. The improvement is based on advanced techniques

Algorithm 1.*Hurtado-Noy* method

Require: Positive integer n

1: Check the structure containing $2n-5$ vertex pairs looking for pairs $(i_k, n-1)$, $i_k \in \{1, 2, ..., n-2\}$, $2 \leq k \leq n-2$, i.e. diagonals incident to vertex n-1. The positions of these indices ik within the structure describing a triangulation should be stored in the array.

2: For every i_k perform the transformation $(i_l, n-1) \rightarrow (i_l, n)$; $i_l < i_k$, $0 \leq l \leq n-3$.

3: Insert new pairs (i_k, n) and $(n-1, n)$ into the structure.

4: Take next i_k, if any, and go to Step (2).

5: Continue the above procedure with next $(n-1)$-gon triangulation (i.e. structure with $2n-5$ vertex pairs) if any. Otherwise halt.

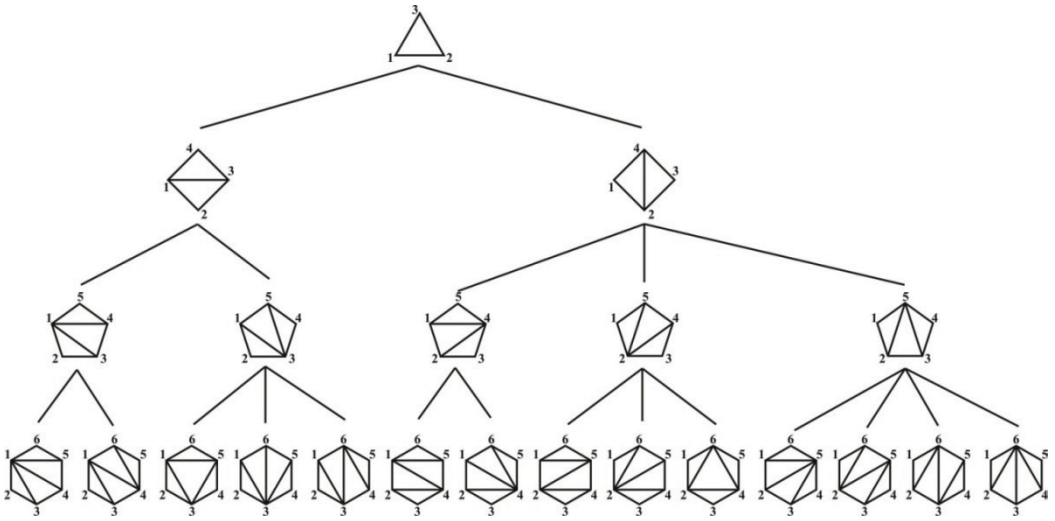

Figure 2. Levels three to six of the tree of triangulations - *Hurtado Noy Hierarchy*

for reverse engineering and synchronization of the *UML* models and the *Java* source code. Advantages of all three approaches are listed in the last section.

Related works

UML modelling has found various applications that cover a wide spectrum of different application domains. During software evolution, programmers devote most of their effort to the understanding of the structure and behaviour of the system.

The paper [20] proposes an UML-based software maintenance process. The authors give the descriptions as variants of UML profiles, describing the styles and rules relevant for a particular application domain. A reverse engineering sub-process, combining top-down and bottom-up reverse engineering activities, aims at constructing the architectural models. The authors describe some of the most advanced techniques that can be employed to reverse engineer several design views from the source code. The paper [18] presents the form driven object-oriented reverse engineering methodology by using forms to recover semantics of legacy applications. The authors propose the application to demonstrate the practical usability of the object-oriented reverse engineering methodology by transforming the resulting object models into well-known UML-based models [12].

The paper [7] presents code reverse engineering problem for identification object-oriented source codes. Paper [21] proposes reverse engineering sequence diagrams from enterprise *Java beans* with interceptors. In the paper [2] authors present an approach and tool to automatically instrument dynamic web applications using source transformation technology and to reverse engineer an UML sequence diagram from the execution traces generated by the resulting instrumentation. The authors of the paper [24] propose combination of the three relations in such way that enables a comprehensive measure of complexity of class diagrams in reverse engineering. A research that is relevant to the application of UML use case diagrams and their comparison, in order to obtain the best possible software at the stage of verification and validation of the software, is presented in [9].

The paper [4] describes the procedure of modelling at the level of hardware, systems and algorithms. The article [16] describes the interaction between behaviour diagrams (activities and states) and interaction diagrams. The method of automatic consistency checking between the two given diagrams is given in [5]. The paper [13] represents the solution of the object-oriented approach in the design and implementation of web based solutions through *UML* and *Java* code. Full analysis of how to model one problem in educational purposes and represent it in a comprehensible way is given in the paper [3].

Forward engineering for polygon triangulation algorithm

UML is a language that creates an abstract model of the system through a set of graphical diagrams. It

can be used for specification, visualization, designing and documentation of the systems development. General classification of standard *UML* views (models) can be divided into: static, dynamic and physical model. Modeling in *UML* has various applications [1,12,20,25] that cover a wide spectrum of different application domains. We monitor the implementation of the convex polygon triangulation by means of *UML* modeling. This monitoring is carried out from abstract ideas, through particular classes, activities, states and behavior of the system, until the physical distribution.

Modeling in UML: static, dynamic and physical model

Our project for modeling Hurtado-Noy algorithm contains 47 diagrams totally, 36 of which are associated with the dynamic model. Table 1 presents all listed diagrams.

Table 1. Overview of *UML* diagrams

Type	UML diagrams	No. of diagrams
Static models	Class diagrams	1
	Object diagrams	8
Dynamic models	Use case diagrams	8
	Activity diagrams	7
	State-Chart diagrams	7
	Sequence diagrams	6
	Collaboration diagrams	4
	Interaction overview diagram	4
Physics models	Deployment diagram	1
	Component diagram	1

All models of the system which are obtained from the developed environment (*Visual Paradigm for UML* and *NetBeansUML*), can be downloaded from[27].

Class diagram describes the static structure of the system. Classes are modeled and mutually connected using these diagrams, while the objects are described by their attributes and relationships with other objects. Each object has a number of methods that can be performed, which is modeled by his behavior. The *UML* class diagram for *Hurtado-Noy* algorithm is presented on Figure 3. These classes are called *GenerateTriangulations, Triangulation, Node, LeafNode, Point and PostScriptWriter*.

The class *Triangulation* is responsible for displaying a convex polygon triangulation. This class provides verification of all the vertices of the polygon. Method *Draw* is a member of the class *Triangulation*. This method is responsible for making an individual triangulation, which is defined by the underlying combination of internal diagonals. Method *DrawAll* also belongs to the class *Triangulation* and it provides iteration where the method *Draw* is called C_{n-2} times. The *Hurtado* method (member of the class *GenerateTriangulations*), creates string of objects of the class *Node* that is used to obtain the appropriate number of vertices (nodes) to form the regular convex polygon. The method *drawLine* is responsible for drawing regular convex polygons. The main executive method in *Java* is defined in the class *GenerateTriangulations*, that requires the input parameter .

Three types of connections are defined in the *Class Diagram*: dependency, generalization and association. Dependency is most commonly used when one class uses another one as an argument. Example of the dependency connection is the relation between the class *Triangulation* and the main class *GenerateTriangulations* (see Figure 3). The Association is a relationship which specifies that objects are associated with other objects (e.g. *Triangulation* with *Node* and *Triangulation* with *Point*). Generalization is a relationship between classes where one class shares the structure and/or behavior of one or more classes. Generalization also defines a hierarchy in which a subclass inherits one or more of the superclass (e.g. *Node* with *LeafNode*).

Operations from the *Class diagram* are further described with behavior and interaction diagrams that together form a dynamic model of the system. Behavior diagrams include activities and state chart views of the system. Interaction diagrams provide data flow between the objects through sequence and communication diagrams.

- Activity diagrams implement the following methods: *createTriangulation, Hurtado, DisplayTriangulations, Draw, DrawAll* and etc. (all methods from the Class diagram).

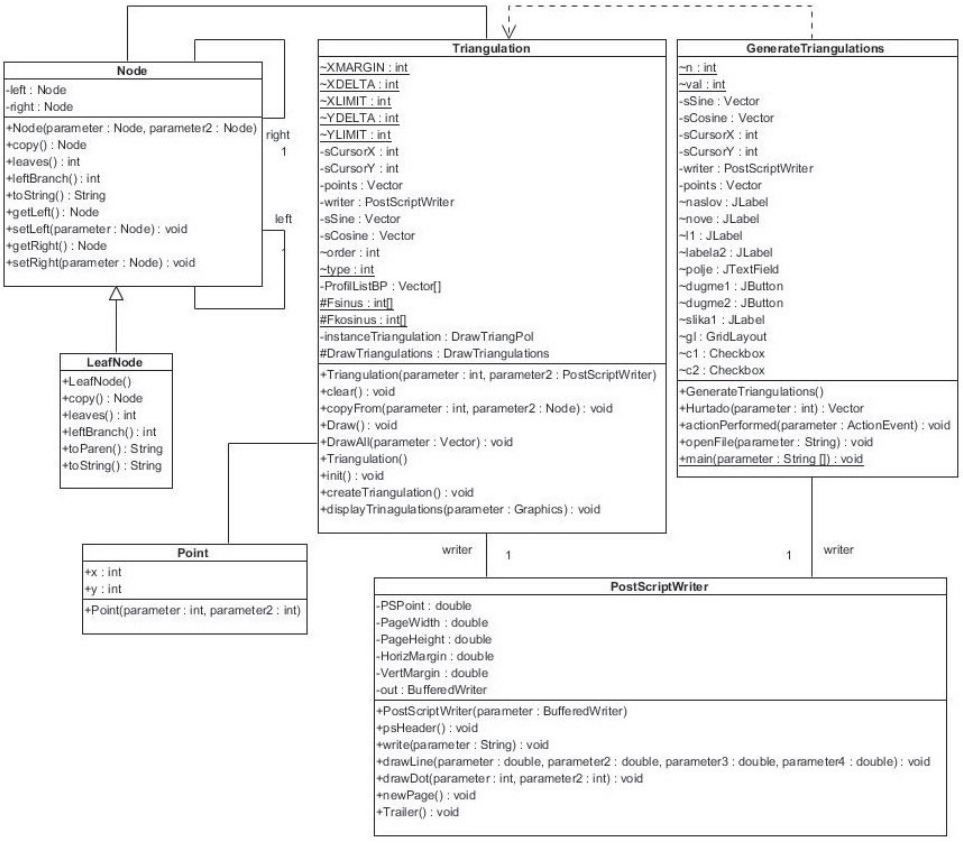

Figure 3. Class diagram

- State-Chart diagrams are used to give an abstract description of the algorithm's behavior. An example of the transition from one state to another is a process of generating triangulation using appropriate methods for moving into their recording and drawing states. Each of these transition states has three optional parts: alerted event (starting methods for *GenerateTriangulations*), security criteria (aimed to generate an exact number of triangulations, equal to the Catalan number) and activity (drawing triangulation with their notations).
- A sequence diagram shows object (class) interactions arranged in time sequence and gives a clear display of cooperation between the class *Triangulation* and the main class *GenerateTriangulations*.
- Diagram of collaboration refers to the interaction of objects (all classes in Figure 3).

Use Case Diagram represents the functional requirements which are imposed to the system. Within the problem of triangulation of polygons, we can observe the following use cases illustrated on Figure 4: *generating triangulation, storage (recording) and drawing triangulation.*

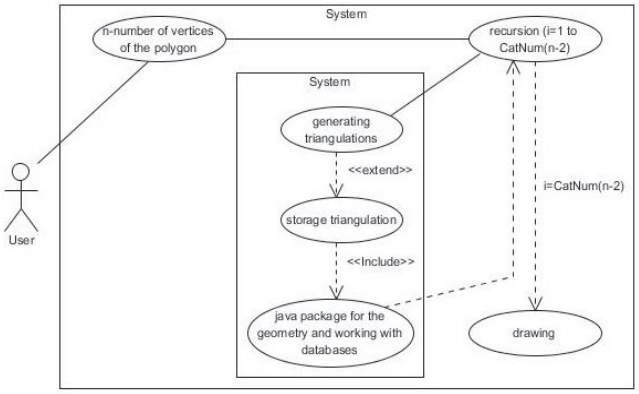

Figure 4. Use case diagram

Figure 4 illustrates the general division of all methods into three groups:

- methods that generate triangulation (with drawing and storing it in the output file),
- methods responsible for the assignment of appropriate notation,
- methods that store the triangulation in different formats through JDBC API.

In addition to these activities, the last stage is drawing triangulation. This stage is supported by corresponding Java package for the geometry [22].

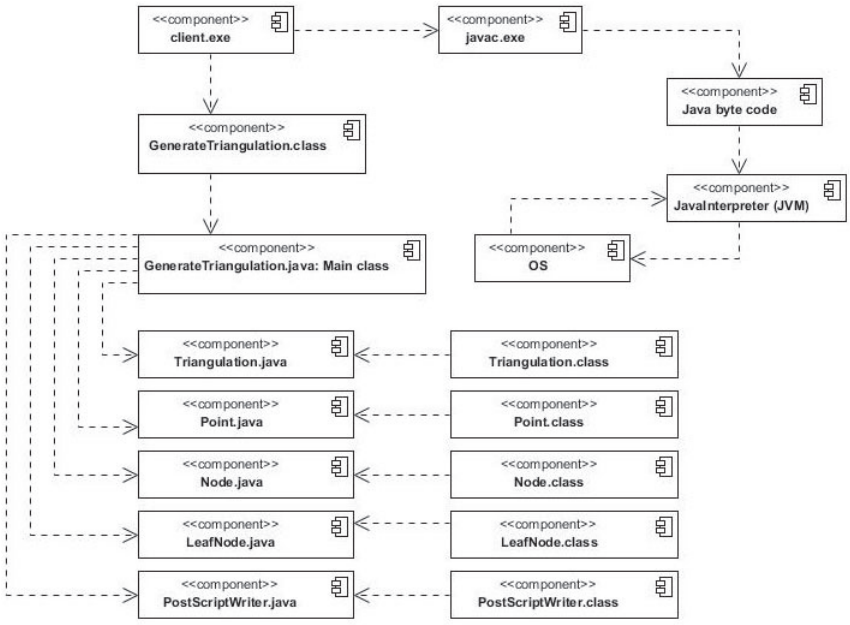

Figure 5. Component diagram

The physical model is implemented through the component diagrams and development (or deployment) diagrams. Deployment diagram shows the hardware structure of the system, i.e. the communication between hardware and software components (dependency connection *supports*). Software component is the implementation of methods, while hardware component is *Java* JDK platform with its components to support the implementation.

The component diagram (Figure 5) represents the structure of the system and describes the dependence of the components of the system. The elements of the diagram are the source codes, library, dynamic components and executable programs.

Generating Java source code from UML models

The idea of generating source code is always associated with the tools and techniques that are based on *UML*. The source code based on the class diagram could be generated in some environments, such as *Visual Paradigm for UML* and *NetBeansUML*. The process of generating source code based on the created model leads to the general structure of a software solution for the algorithm. In this way, we can provide a fast and efficient transfer model in a customizable *Java* source code. The application of the direct development on the *Class Diagram* we get all the classes with the general structure (i.e. declarations of variables and headers of their methods).

The complete structure that is obtained from the UML models can be downloaded from [28].

Example 1. Here we specify one example of generating one segment of the Java source code for the classes *Triangulation* and *GenerateTriangulations*. The sign "*" denote that there is a possibility for the code modification in order to achieve the necessary and/or desired functionalities.

```
public class Triangulation{

//attributes public
Object private int sCursorX;
public Object private int sCursorY;
public Object private Vector<Point>
points;
public Object private PostScriptWriter
writer;

***

//operations
public void Draw() {*}
public void DrawAll(Vector<Node> trees)
() {*}
public void clear(){*}
public void copyFrom(Object int aOffset,
Object Node t) {*}
}

public class GenerateTriangulation
implements Triangulation
//operations
```

```
public void Vector<Node>
createTriangulations (Object int limit)
{*}
public void static void main(Object
String args[]){*}
}
```

Experimental results

Programming phase is the next step that comes after the procedure of direct development. We used the *NetBeans IDE environment* available in *Java* for the implementation of this phase. A comparative analysis of the implementations of the Hurtado-Noy algorithm in three programming languages (*Java*, Python and C++) is presented in our paper [23]. Numerical experience from this paper shows that the implementation in *Java* programming language produces the best results.

Table 2 contains CPU times required for generating all possible triangulations of convex polygon (denotes the number of the polygon vertices).

Table 2. Execution time for the *Java* application

Num. of vertices	*No. of triangulations*	*Execution time (in sec.)*	*File output size (in Kb)*
5	5	0.2	0.1
6	14	0.3	0.4
7	42	0.4	1.7
8	132	0.5	6.4
9	429	0.6	24.5
10	1430	0.9	93.1
11	4,862	2.2	355.6
12	16,796	5.8	1,363.2
13	58,786	15.2	4,121.4
14	208,012	46.34	11,523.6
15	742,900	124.18	29,874.29

Numerical results are derived using personal computer with performances: *CPU - Intel(R) Core2Duo, T7700, 2.40GHz, L2 Cache 4 MB (On-Die,ATC),RAM Memory -2 Gb, Graphic card - NVIDIA GeForce 8600M GS.*

Based on the obtained results, it can be observed that increasing the values of *n* (number of vertices) increases the number of generated triangulations per second (Figure 6). The vertical axis of the graphical representation contains the number of displayed triangulations per second while the horizontal axis contains values for *n*.

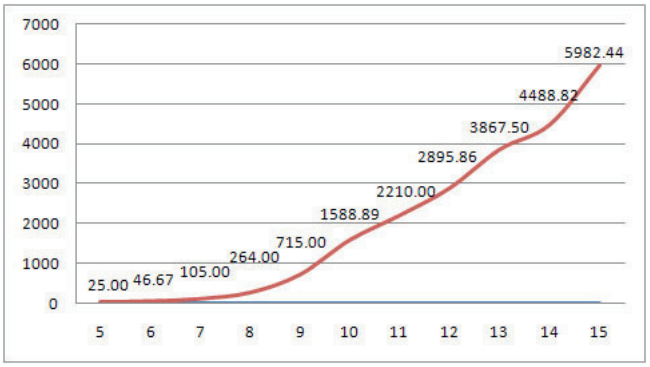

Figure 6. Number of generated triangulation per second for $n = 5,\ldots,15$.

Java application can be downloaded from [29].

Reverse and round-trip engineering: maintenance and evolution of software solution

This section provides a procedure for the feedback analysis and the ability to synchronize the implementation of algorithm for triangulation, which turned out to be the best solution.

Reverse engineering and visualization of source code lead to improved program comprehension. The main advantages are: *learning unfamiliar code, code reuse, software maintenance, change impact analysis, integrating open source code* and etc. This approach has various applications for identification of the object-oriented codes [6,26].

The reverse engineering for the implementation of a polygon triangulation is implemented through two phases:

1. It begins with the classification of the complete source code in formal units (classes) to obtain the static model (*use–case* and *class diagrams*).
2. On the basis of modeled attributes and operations, their descriptions are further decomposed into dynamic diagrams.

Round-trip engineering presents synchronization of direct development and feedback analysis, which is a good practice in the analysis and maintenance of the implementation [7,17]. Their benefits are di-

rectly related to the change of the source code from *UML* model and vice versa (see Figure 7).

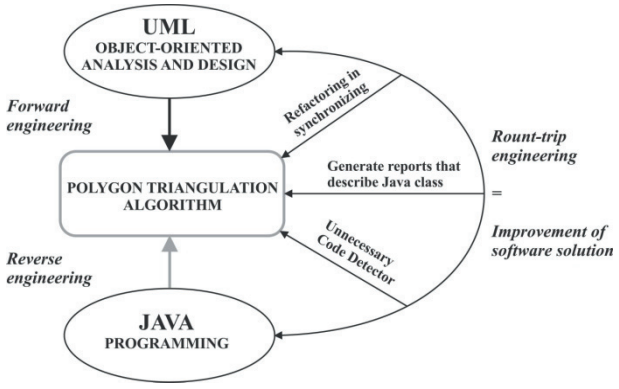

Figure 7. Synchronization of direct development and feedback analysis

Example 2. For synchronizing UML project and Java project for the triangulation polygon, NetBeansUML module will log various lines of text to the Output Window as follows:

> *"...Initial reverse engineering into a new project: Begin processing Reverse Engineering, Parsing 56 elements, Analyzing attributes and operations for 72 symbols, Resolving 54 attribute types, Integrating 72 elements,Building the query cache...".*

The output result describes the operations that took place: 72 model elements were used to generate Java source code files.Table 3 presents the fulfillment requirements in the process of synchronizing UML project and Java project for triangulation of a convex polygon *(*DP - Design pattern, A - Attributes, O - Operations, I - Implementation, R - Relationships*).

Table 3. Requirements in the round-trip engineering

	REQUIREMENTS	Triangulation	Generate Triangulations
1	Navigate to Source	+	+
2	Generate Dependency diagram	+	+
3	Generate Code	+	+
4	Generating report	+	+
5	Element Navigation	+	+
6	Refactoring in synchronizing	+	+
7	Find and Replace in *UML* model	+	
8	Apply *DP* and source code readability	+	
9	Manipulation with *A, O, I* and *R**	+	+

Description of Requirements: In the *NetBeansUML* module there is the possibility of automatic detecting source code if the synchronization between the *UML* project and *Java NetBeans* project is properly set up (requirements 1,2,3 and 5 from Table 3.1). In this way, it reduces the complexity of the triangulation problem.

Programming and adding new functionality of the system is also facilitated. For the reverse and round-trip engineering process it is important to mention the procedure for generating a report of the model. The report describes all classes defined in the project and the use of packages, interfaces and data types in the implementation (requirement 4).

The main categories of reverse engineering are automatic restructuring and automatic transformation. The first category refers to re-factoring and re-modularization that is applied with the aim of obtaining a better source code (requirements 6 and 7). The second category refers to the application of standards in coding, which is applicable in order to obtain the source code readability (requirement 8).

Re-factoring changes the internal structure of the software (requirement 9) in order to be easier to understand and simpler to modify, without visible changes of his behavior.

The following actions are implemented in our project in the procedure of code re-factoring: *extract and move method, class and super-class; extract interface; use super-type where is possible; creating a template method and encapsulate fields.*

Obtained results in the improved software solution

Table 4 presents the results of improving software solution for the Hurtado-Noy algorithm. Data derived after improvements are presented by the sign '*'. Three criteria are used in the comparison: the number of source code lines, the size of *Java* file (in bytes) and measuring the time (speed) in seconds (individually for each *Java* file), for $n = 5,\ldots,15,$ cumulative.

Table 4. Improvement of source code for Hurtado-Noy algorithm

Segment	line	**line***	bytes	**bytes***	speed	**speed***
Triangulation	79	**67**	2275	**1715**	29.12	**26.14**
GenerateTriangulation	222	**195**	8544	**7194**	37.56	**34.33**
Node	45	**41**	745	**578**	4.51	**4.01**
LeafNode	27	**19**	342	**216**	3.21	**2.85**
Point	10	**9**	134	**112**	1.52	**1.51**
PostScriptWriter	56	**55**	1830	**1791**	2.74	**2.62**
TOTAL	439	**386**	13870	**11606**	78.66	**71.46**
Improvement (%)	13.73		19.51		10.08	

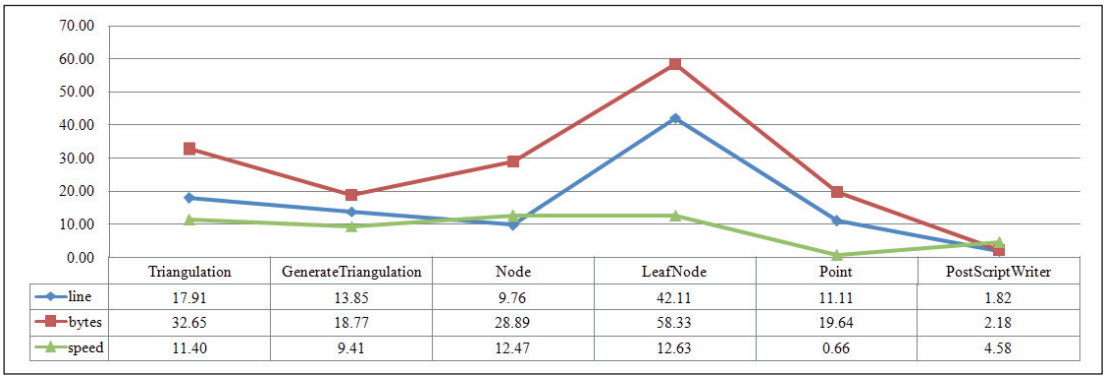

Figure 8. Improvemets (in %) for Hurtado-Noy algorithm (individually by classes)

Figure 8 shows the percentage of improving the source code for three criteria, individually for each segment of the implementation.

In process of code review for Hurtado-Noy algorithm, *NetBeans module "Unnecessary Code Detector"* recognizes the following: *unused imports , unread local variable , unread parameter , unnecessary method or constructor , unread private method, constructor or type and unread local or private members .*

Some advantages that occur in the application of the reverse and round-trip engineering in our implementation are:

1. Better understanding of defined classes and their methods, identifying interdependencies, ways of communication and data flow. Hence, given technique offers the possibility of generating alternative views of the problem.
2. Source code analysis obtained through several models (primarily static and dynamic) provide the possibility to expand the source code and simplify methods. This allows the detection of repeated cases in the code.

Table 5. Advantages of three approaches

Engineering	Advantages
Forward	Multi-dimensionality of the system and the high level of abstraction
	Efficient transparency of the system structure
	Spotting the functional wholes
	Generating the source code
Reverse	Analysis and interpretation of the implementation problems
	Logical design and better visibility of source code
	Disassemble of *Java* code
	More effective maintenance of a software solution
Round-Trip	Synchronized changes from model to source code or vice versa
	Generate reports that describe class (graphic and program description)
	Combining the advantages of the first two approaches

3. After locating and removing the source code or modules that are not used anymore, we reduce the complexity of the problem and simplify the source code. Therefore, we achieved better results concerning the speed of generating triangulations per second.

Table 5 shows the identified advantages of all three approaches in the implementation.

Conclusion

This paper outlines the key advantages of the object-oriented analysis and design in solving the convex polygon triangulations, which is a fundamental algorithm in computational geometry. Direct development has the advantage of generating the source code in some of the object-oriented programming languages, while reanalysis technique aims to describe the implementation of the problem through various aspects. Synchronization procedure combines the advantages of these two approaches.

Based on the presented testing, we conclude that the best practice is the synchronization technique that combines *Java* programming and *UML* modeling. Some of the advantages of reverse engineering and synchronization of direct development and feedback analysis to solve the problem of triangulation are coping with the complexity of the problem, better understanding of the classes and their methods, identifying interdependencies, ways of communication and data flow. In addition, a given technique offers the possibility of generating alternative views of the problem. Source code analysis obtained through several models allows you to see the problem from the aspect of expandability of the source code (adding new methods), possibilities of simplification of methods, redefining the methods and synthesis and analysis methods. In this way they can link certain implementation on the basis of their dynamic and static models. This allows the detection of secondary occurrences and repeated cases. OOAD technique enables reuse of already implemented classes, in terms of easy and efficient adding new attributes and operations.

Obtained results indicate improvement of software solutions through three aspects: the number of source code lines, the size of output file and speed of execution. The archival value of the paper is a contribution to the engineering education through a case study in the computational geometry. This technique of three approaches can be applied as a new method for solving and analyzing related problems. Generally, the suggested approach is suitable for the implementation of some other algorithms in computational geometry.

Acknowledgement
The authors gratefully acknowledge support from the Research Project 174013 of the Serbian Ministry of Science.

Authorship statement
Author(s) confirms that the above named article is an original work, did not previously published or is currently under consideration for any other publication.

Conflicts of interest
We declare that we have no conflicts of interest.

References

[1] Aghasaryan, A., Jard, C., and Thomas, J. (2004). UML Specification of a Generic Model for Fault Diagnosis of Telecommunication Networks. In Proceedings of 11th International Conference on Telecommunications, Fortalezza, Brasil, 841-847.

[2] Alalfi, M.H., Cordy, J.R., and Dean, T.R. (2009). Automated Reverse Engineering of UML Sequence Diagrams for Dynamic Web Applications. In IEEE international conference on software testing, verification, and validation workshops, 287-294.

[3] Ayachi-Ghanouchi, S., Cheniti-Belcadhi, L., and Lewis, R. (2013). Analysis and modeling of tutor functions. *Computer Applications in Engineering Education*, 21 (4), 657–670.

[4] Bahill Tand Daniels, J. (2002). Using object-oriented and UML tools for hardware design: A case study. *Systems Engineering*, 6 (1), 28-48.

[5] Belt, J. (2005). Automated Consistency Checking between UML State Charts and Sequence Diagram. CIS 798.

[6] Bruegge, B. (2004). Object-Oriented software engineering: Using UML, patterns and Java. Pearson Education, New Jersey.

[7] Bringer, J., and Chabanne, H. (2012). Code Reverse Engineering Problem for Identification Codes. *IEEE transactions on information theory*, 58(4), 2406–2412.

[8] Chen Jand Chen, C. (2008). Foundations of 3D Graphics Programming: Using JOGL and Java3D. Springer, New York.

[9] Funkhouser, O., Etzkorn, L., and Hughes, W. (2008). A Lightweight Approach to Software Validation By Comparing UML Use Cases with Internal Program Documentation Selected Via Call Graphs. *Software Quality Journal*, 16 (1), 131-156.

[10] Garey, M.R., Johnson, D.S., Preparata, F., Pand Tarjan, R.E. (1978). Triangulating a simple polygon. *Inform. Process. Lett*, 7, 175–180.

[11] Hurtado, F., and Noy, M. (1999). Graph of Triangulations of a Convex Polygon and tree of triangulations. *Computational Geometry*, 13, 179–188.

[12] Huynh, S., Cai, Y., and Shen, W. (2008). Automatic Transformation of UML Models into Analytical Decision Models. Technical Report DU-CS-08-01, Drexel University.

[13] Jayaramaraja, S. (2005). An object-oriented design and reference implementation for web-based instructional software. *Computer Applications in Engineering Education*, 13 (1), 26-39.

[14] Klawonn, F. (2012). Introduction to Computer Graphics: Using Java 2D and 3D: Second Edition. Springer, New York.

[15] Koshy, T. (2009). Catalan Numbers with Applications. Oxford University Press, New York.

[16] Knapp, A., and Merz, S. (2002). Model Checking and Code Generation for UML State Machines and Collaborations Tools for System Design and Verification. Institut fur Informatik, Universitat Augsburg, 59-64.

[17] Lano, K. (2005). Advanced Systems Design with Java, UML, and MDA. Elsevier publisher.

[18] Lee, H., Yoo, C. (2000). A form driven object-oriented reverse engineering methodology. *Information systems*, 25 (3), pp. 235-259.

[19] Loera Jand Santo, F. (2003). Triangulations: Structures for Algorithms and Applications. Springer Verlag, New York.

[20] Riva, C., Selonen, P., Systa, T., and Xu, J. (2004). UML-based reverse engineering and model analysis approaches for software architecture maintenance. In Proceedings of 20th IEEE international conference on software maintenance, USA, 50–59.

[21] Roubtsov, S., Serebrenik, A., Mazoyer, A., and Brand, M. (2011). I2SD: Reverse Engineering Sequence Diagrams from Enterprise Java Beans with Interceptors. In 11th IEEE international working conference on source code analysis and manipulation (SCAM 2011), 155-164.

[22] Saračević, M., Stanimirović, P., and Mašović, S. (2013). Implementation of some algorithms in computer graphics in Java. *Technics Technologies Education Management*, 8 (1), 293–300.

[23] Saračević, M., Stanimirović, P., Mašović Sand Biševac, E. (2012). Implementation of the convex polygon trangulation algorithm. *Facta Universitatis, series: Mathematics and Informatics*, 27 (2), 67–82.

[24] Sheldon, F.T., and Chung, H. (2006). Measuring the complexity of class diagrams in reverse engineering. *Journal of software maintenance and evolution-research and practice*, 18 (5), 333-350.

[25] Stanimirović, P., Tasić, M., Saračević, M., and Mašović, S. (2012). UML-based modeling for Moore-Penrose inverse computation. *Revista Metal. International*, 17 (12), 99–106.

[26] Tonella, P. (2005). Reverse engineering of object oriented code. In Proceedings of 27th International Conference on Software Engineering. Missouri, USA, 724–725.

[27] Link for all UML models:http://muzafers.uninp.edu.rs/triangulation/UMLTriang.rar

[28] Structureof Java Code:http://muzafers.uninp.edu.rs/triangulation/GeneralStructureHurtado.rar

[29] Link for Java application:http://muzafers.uninp.edu.rs/triangulation/Hurtado.rar

Submitted: October 30, 2013.
Accepted: December 7, 2013.