Reverse Engineering of Source Code to Sequence Diagram Using Abstract Syntax Tree

Esa Fauzi[1], Bayu Hendradjaya[2], Wikan Danar Sunindyo[3]
School of Electrical Engineering and Informatics
Bandung Technology Institute
Indonesia
esa.fauzi@gmail.com, bayu@informatika.org, wikan@informatika.org

***Abstract*--Reverse engineering from a source code to a sequence diagram can be very important in the software maintenance process. On the process of reverse engineering, there are a lot of models/approaches that can be used. Extract-abstract-present model is one model that can be applied because every stage in the model can represent reverse engineering process. On stages in the model, the extracted source code is converted into a specific structure for easy analyzed in the next process. AST (abstract syntax tree) is one of the structures that can be used in the extraction process. AST is commonly used as a data structure in the compiler.**

This research focus on the process of reverse engineering of source code to sequence diagrams. AST is used to generate the sequence diagram. We have implemented this idea in an application called REVUML. REVUML has helped in the process of understanding the flow of the source code in a software application. Our research concludes that the structure of AST has been able to assist in the reverse engineering process. Additionally our implementation has shown that AST can reveal sequences of statements in the source code which can help developers in the reverse engineering process.

Keywords---reverse engineering, source code, abstract syntax tree, sequence diagram

I. INTRODUCTION

Reverse engineering into a sequence diagram has an important role in the software maintenance. This is because in the process of maintenance, reading sequence diagrams can help in understanding the logic flow of the application, and communication between each components or objects. Reverse engineering is the process of analyzing a subject system to identify system components and dependencies as well as making the representation system in a form or in a high level abstraction [8]. In other words, reverse engineering is the way to get back the requirement or design of a system.

Currently, there are a lot of models or approaches that could be used in the process of reverse engineering. Extract-abstract-present model [1] is one of the model approach that can be used. This model was originally used in the process of software architecture analysis, but also it can be used as process of reverse engineering model because this model has stages that have opportunity to represent the process of reverse engineering. The first stage is the extraction, which means performing the extraction of source code and mapping into a structure. Then the second stage is abstraction, which means applying the operation on the previous structure to get a result. Finally, the last stage is the presentation, which means displays the results visually.

On stages in the model of the extract-abstract-present model, the extracted source code is converted into a specific structure for easy analyzed in the next process. AST (abstract syntax tree) is one of the structures that can be used. AST is a representation of a graph in the form of a tree from abstract syntactic structure of source code. AST is commonly used as a data structure in the compiler. In addition, the AST is also usually used in analysis of source code, such as for clone detection [2], to understand the evolution of source code [3], and to detect plagiarism source code [4]. Then, in the field of reverse engineering, AST has ever used in reverse engineering into the class diagram [5]. But for reverse engineering into a sequence diagram, AST so far have never used directly. An example is on the CPP2XMI [6] and I2SD [7] application. Both the application using the library (Columbus/CAN and JavaCC) to transform the source code into the AST then stored it into XML. According to us by changing the AST into XML, it means that we first have to create a process to read the AST in XML format before analyzing the AST to be converted into a sequence diagram. This process will be faster if we can use the AST without having to read the XML first. According to us, XML format is better used when we have a specific reason. On the application of I2SD, actually the XML format is used because they are doing reverse engineering for enterprise java beans with interceptor, while in the CPP2XMI, XML format is used because for ease in getting information about class, activity, and sequence diagrams.

Other primary motivation in this paper, is that we are trying to perform reverse engineering to sequence diagram from several cases such as inheritance and polymorphism. Both these cases are generally described with class diagram. However, the goal we are trying to illustrate cases like this in our research is to provide convenience for the user to understand sequence diagrams that we produce. For example

978-1-5090-5671-2/16/$31.00 ©2016 IEEE

the method call that involves the inheritance, intended to facilitate users in knowing its superclass and subclass. Additionally, if the method call is derived from the superclass, then users can instantly find out from sequence diagram that we produce without having to explore each superclass one by one through the source code.

In this paper, we have investigated on how to trace AST in order to generate sequence diagrams and how to do reverse engineering sequence diagram from several cases such as nested classes, static method, polymorphism, interface, abstraction, and inheritance. We then present the design and implementation of the tool that performs reverse engineering of source code into a sequence diagram. We have used several case studies, and discussed the results. In the end, we conclude the research.

II. Basic Concept and Related Works

A. *Sequence Diagram*

Sequence diagram is a diagram that illustrates the interactions between the object with the focus into the sequence of the messages which is interchangeable [9]. On a sequence diagram, there is a combined fragment that is the fragment interaction defined by operators (such as alt and loops) and the corresponding interaction operand [10]. This combined fragment represents the alternative (example: if) and looping (example: while) in the source code.

B. *Abstract Syntax Tree*

AST (abstract syntax tree) is a representation of a graph in the form of a tree from abstract syntactic structure of source code. As a tree graph that has node and edge, then on the AST, node is symbolic events in the source code (such as statements, loops, and if) while the edges describe the relationships between AST node.

Before building the AST, the thing to watch out for is how to define the correct AST from different types of input structures. According to Fowler in his book, ASTs should do the following [11]:

1. Record the meaningful input tokens (and only the meaningful tokens)
2. Encode, in the two-dimensional structure of the tree, the grammatical structure used by the parser to match the associated tokens but not the superfluous rule names themselves
3. Be easy for the computer to recognize and navigate.

C. *Related Research*

A lot of research has been conducted related to reverse engineering. Some studies focus into the graph as a metamodel that generated from source code. Tgraph [12], control flow graph [13] [14] [15], object flow graph [16], and abstract syntax tree [5] are some of them. Some of the metamodel are converted into another model. Control flow graph is modified into a sequence diagram [13] [14] [15], object flow graph changed into UML diagrams, and abstract syntax tree transformed into a class diagram.

At the time of this writing, the authors have not find a direct implementation of abstract syntax tree or AST into sequence diagram. Most of the research has converted AST to another form such as XML before it is converted into a sequence diagram which this process is requires more work and time. For example: the application of reverse engineering CPP2XMI [6] and I2SD [7]. On the process from the both application is indeed using AST, but the structure was changed into the form of a sequence diagram is obtained from XML. AST is converted first into XML before it is transformed into a sequence diagram.

III. Analysis and Design

We have designed and developed REVUML that implements the reverse engineering process from AST into sequence diagram. REVUML accepts source code as an input. The source code then is extracted to generate hierarchical structure of AST. In the abstraction stage, the structure of AST is analyzed and processed in order to produce a set of data to generate sequence diagram. Then in the presentation stage, this set is converted to a sequence diagram. Figure 1 shows the REVUML process flow.

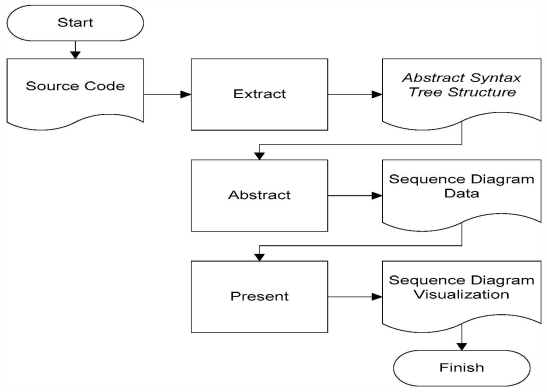

Fig. 1. Process flow of REVUML

A. *Extraction Of Source Code*

To form sequence diagrams, we have used AST to trace its class/objects, method calls, object creations, loop and alternative statements, return statement, assigment expression, package and import statements, and inheritance and polimorphism concepts. JavaParser API is used in the extraction process from source code to AST. JavaParser store AST in form of objects so we can use it directly in our implementation. JavaParser also performs lexical and parsing analysis.

Lexical analysis categorizes characters into sequence of tokens from the source code. The token is used in the process identification of a node in AST. After lexical analysis process,

the results of this lexeme and the token is parsed into an AST. Figure 2 shows the process that occurs at the stage of the extract.

Fig 2. The process of extract stages

B. Abstraction of The AST

Abstraction stage is the stage of the analysis process from AST data into sequence diagram. We begin by exploring important elements in the source code by executing AST Traversal algorithm.

B. I. Elements of Source Code

The followings are the elements involved in the process of Reverse Engineering of source code to Sequence Diagram.

1) Class/Object

Class on sequence diagrams is represented as an object. There are many types of classes that must be considered in the process of reverse engineering. This is because each type has a different pattern of class in the form of the calling method call. Figure 3 shows the types of classes that must be observed.

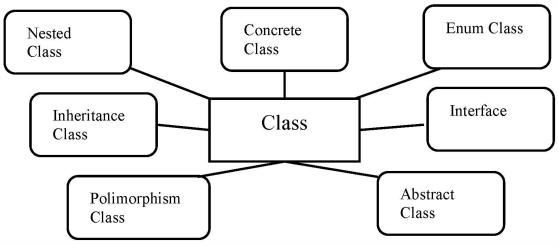

Fig. 3. An influential class in reverse engineering sequence diagram

2) Method Call

Method call is portrayed as a message (an line-arrow sign) in sequence diagram. A method call can be affected by the scope. Scope is the token from the category of expression that written in front of the method call. Table 1 shows the form of the method call with his scope.

TABLE 1. FORM OF METHOD CALL

No	Form of *Method Call*	Example
1	Without *scope*	`getData()`
2	*Scope name expression*	`a.getData();`
3	*Scope field access expression*	`a.b.getData();`
4	*Scope method call expression*	`get().add();`
5	*Scope Enum Object Creation*	`Level.HIGH.get();`
6	*Scope Static method call*	`Buku.getName();`
7	*Scope this expression*	`this.getData();`
8	*Scope super expression*	`super.getData();`
9	*Scope Object Creation*	`new Data().getData();`
10	*Combination Scope*	`a.get().x.count();`

Besides the method call form on table 1, method call in java can be implemented in several ways, such as with a lambda expression or default method. Lambda expression and default method are a new feature in java 8.

3) Loop dan Alternative Statements

Loop and alt statement are one of the components that portrayed as a combined fragment in a sequence diagram. Alt statement itself consists of an 'if-then', 'if-then-else', 'if-then-elseif', 'switch-case', and unary expression (e.g. minVal = (a < b) ? a : b;) while loop statement consists of a 'do-while', 'for each', 'for', and 'while'.

4) Object Creation

Object creation is portrayed like a method call on a sequence diagram. But unlike the method call, on a method-call-message arrow points lifeline, while at object creation, the arrow points directly at the object. Just as method call, the form of object creation should be defined. Table 2 shows the form of the object creation.

TABLE 2. FORM OF OBJECT CREATION

No	Form	Example
1	Object creation	`A = new ZObjCreationA();`
2	Object creation inner class (non static)	`ZObjCreationA A = new ZObjCreationA(); ZObjCreationA.InnerClass inner = A.new InnerClass();`
3	Object creation inner class (static)	`ZObjCreationA.InnerClass inner = ZObjCreationA.new InnerClass();`
4	Object creation inner class (*anonymous*)	`A = new ZObjCreationA(){ public void my_method(){ ..... } };`
5	Object creation enum class (*anonymous*)	`Level level = Level.HIGH;`

5) Return Statements

The return statement in the process of reverse engineering into the sequence diagram is illustrated in the form of a message. But in contrast to the message from the method call, on message return statement, the message is shown with the arrow in the opposite direction from the method call message.

6) Package and Import Statements

In this research, package and import used as elements that help in the search process method call from outside the class.

7) Assigment Expression

Method call of a variable can be made after known the data type of the variable. One of the functions of expression assigment in programming is to provide 'value' to a variable. The assigment expression of variable may change the data type of the variable. This is due to the presence of polimerfisme concept in OOP.

8) Inheritance and Polimorphism Concepts

On the concept of inheritance or polymorphism, if the object was created from a subclass then other related object as superclass also created. This is because attributes (fields and methods) of the superclass could be used also in a subclass object. Therefore, in the process of reverse engineering of the cases such as inheritance and polymorphism, if a subclass object was created then in the depiction of sequence diagram, a subclass and also it superclass are described.

B. II. AST Traversal Algorithm

To begin the process of reverse engineering source code into a sequence diagram by using AST, then the first thing to do is define the input for the reverse engineering process. The input is a node (vertex) of the method declarations that want to generate. The node is then traced down one by one in sequence in accordance with the original source code. In order for AST node can be traced sequentially, appropriate traversal algorithm is required. The algorithm that can be used to navigate nodes AST sequentially in accordance with source code is the DFS Post Order algorithm.

A method call may have a scope of expression name or variable that caused the method call gets called from outside the class (example: a variable from a.get() method). Therefore before starting the process to generate sequence diagrams, we perform initialization of variables that exist in the field of the class (see algorithm 1). The initialization is basically records the location of related variables with a method which will be in reverse engineering (such as variables that are written on a field or parameter). The Purpose of this initialization is to make the variable searches easier without tracing the AST again from the root or the leaf when AST traversal algorithm finds a method call that has a scope name expression/variable. The initialization also include from field in superclass or nested class.

Algorithm 1. generateSequence

```
void generateSequence(Node classDecl, Node
methodDecl){
   initializationVarFromField(classDecl);
   String seqData =
           generateMethod(methodDecl,seqData,
                          lstVarRec);
   transformToSeqDiagram(seqData);
}
```

The node class declaration (from the variable classDecl) is needed to obtain information about the 'extends', 'implements', or inner class parent from generated method. The first step, the algorithm performs the initialization of the variable from field of the class (from initializationVarFromField method) followed by generate process of the method to get a sequence diagram (in the generateMethod method). In this research, the type of sequence data is string because we use PlantUML API to draw sequence diagram (input for PlantUML is string) so the result of the process of generate method is stored into string variable seqData. Then the seqData variable is used as parameter in the transformToSeqDiagram method. In the transformToSeqDiagram method, the seqData variable is converted into a sequence diagram.

To get the sequence diagram from a method then the AST of the method should be traced and analyzed. The search algorithm approach is using DFS Post Order. Each node in the body method is analyzed and checked whether it includes elements that build sequence diagrams or not (see algorithm 2). The node is traced one by one using DFS post order algorithm. Each node is then examined and analyzed whether it includes the combined fragment, variables (declaration and assigment), or a method call and the object creation as well. Specially when the node is the variable declaration or expression of assigment then the application will do the process of variable recording. Note the variable was recorded because the influence of polymorphism that can change the variable type. This process is useful later to find variable from the method call or object creation.

Algorithm 2. generateSequence

```
String generateMethod(Node node, String dataSeq,
List listVarRecord){
  node = nextDFSNode(node);
  if(endOfNodeMethod(node) == false){
    if (altCombinedFragmentExpr(node)){
      dataSeq = dataSeq +
                generateAltCombinedFragment(node);
    }else if (loopCombinedFragmentExpr(node)){
      dataSeq = dataSeq +
                generateLoopCombinedFragment(node);
    }else if (VariableDeclExpr(Node)){
      VarRecord vr = createVarRecord(node);
      listVarRecord.add(vr);
    }else if (AssigmentExpr(Node)){
      VarRecord vr = getVarRecord(node);
      checkVarRecord(vr);
    }else if (MethodCallExpr(Node)){
      dataSeq = dataSeq +
                generateMethodCall(node);
    }else if (ObjectCreationExpr(Node)){
      dataSeq = dataSeq +
                generateObjCreation(node);
    }
    generateMethod(node, dataSeq);
  }
}
```

Then, after the tracing to find method calls or object creations in AST, the algorithm looks for a declaration of the method or the constructor. This is done to validate the method call/object creation and for search into the next level. To obtain the declaration of method and constructor then that

should be done first is to distinguish the form of method call or object creation by looking at the scope. Scope must be identified so that the location of the class from the declaration of the method or constructor can be found.

If the scope is a field access or method call then the location of the class can be known by looking for the type of field or method declaration. However, if the scope is the name of the expression (such as variables) then the thing to do is look for the data type of the variable. The search process of data type is to trace the record variables that have been made.

C. *Presentation of Sequence Diagram*

At this stage of the process of the drawing of sequence diagram is assisted by PlantUML API. Plant UML is an application that can draw UML diagrams and store it in the form of image format (png) or vector format (svg) files.

IV. CASE STUDY

This section reports a small experiment how REVUML performing reverse engineering into a sequence diagram. A project is created as a test. The project contains methods and objects creation from various examples of cases that could be represented into the sequence diagram. For example, it can be seen from the source code below. REVUML application tried to reverse engineer the main method in the example.

```
import revuml.test.OOConcept.inheritance.Animal;
import revuml.test.OOConcept.inheritance.Cat;
public class REVUMLTest {
    public static void main(String[] args) {
        int a, b;
        String name = "Kitty";
        a = 10;
        b = 5;
        Animal A = new Animal();
        A.setName(name);
        while (a < 100 && b > 10) {
            Cat B = new Cat();
            if (b > 5) {
                B.getSpecies();
            } else {
                B.setName("New Kitty");
            }
        }
    }
}
```

Figure 4 is the result of reverse engineering of the main method. From the picture on the first sequence we can see object creation from Animal class and setName method call from Animal object through the variable A. After that we came to know object creation of Animal in 'while' condition through variabel B. On the object Animal we saw the sign '<superclass>'. This sign indicates that the object is a superclass. This means the creation of the next object (Cat object) is the subclass from Animal object. The next thing, in 'while' conditions there are method call in 'if-else' condition. The methods are getSpecies method call on the condition of 'if' and setName method call on the condition of 'else'. On the calling method getSpecies we can see inside it there are the making of the Species object and a return statement with sp variable. If we see in the source code, the method of getName and getSpecies are called through the variable B. But we can see in the sequence diagram that the setName method refers to the Animal object instead of the Cat object. This means showing that the setName method is derived from the parent of Cat object (the Animal object).

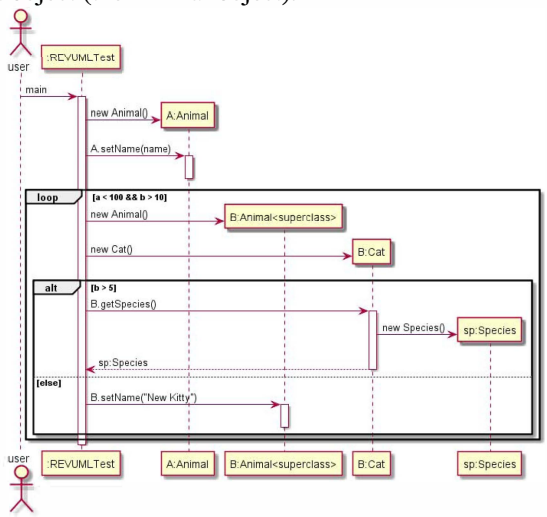

Fig.4 The screenshot of Sequence Diagram from REVUML

To validate the results, we have tested using 126 case studies, with different categories (see table 3).

TABLE 3. TOTAL OF TEST CASES

No	Categories	Test Cases Total
1	Combined fragments (alt and loop) cases	32 test cases
2	Method call locations	25 test cases
3	Method call forms	19 test cases
4	Object creation locations	4 test cases
5	Object creation forms (include nested class)	18 test cases
6	Inheritance and polimorphism cases (include from interface and abstraction forms)	20 test cases
7	Static method	2 test cases
8	Generate sequence diagram from library class cases	6 test cases

V. DISCUSSION

In this research, we have exercised any possibilities that can be represented in reverse engineering sequence diagram. The results of testing against the test case scenarios showed that application of REVUML have been able to draw the right sequence diagram in different cases. The following is a variety of feature that is deployed in REVUML:

1. The application is able to implement the reverse engineering and get combined fragment from the

condition ' if ', ' if-else ', ' elseif',-if the ' switch-case ', ' if ternary ', ' while ', ' do-while ', ' for ', and ' for each

2. The application is able to create a sequence diagram above 1 level. It means, if inside the selected method call turns out to have more another method call then the method call is traced until the last level or no more method call.
3. The application is capable of searching for a method call in many locations (e.g. the condition of an if, while, the parameters of the method, and others) and many form .
4. The application is capable of detecting recursive and cyclic method call (e.g method A() calls method B() and the method B() calls again the method A()).
5. The application is able to make sequence diagrams from cases like nested classes, inheritance, polymorphism, interfaces, static method, default method, lambda expression, and abstraction.

REVUML has been exercised many case studies. In general from the test cases, we can see the order of every method call or the object creation that involved. We can also see the association of if (alt) statement and while (loop) statement on each method call. In addition, the application can tell us the depth of content of each method call/ object creation so that we can know all the method call/object creation from a method call/object creation.

VI. CONCLUSION

This research indicates that the abstract syntax tree or AST results that obtained from extracts source code can serve as a structure that helps the process of reverse engineering into a sequence diagram. This is because object-object sequence diagrams (such as calling methods and classes) can be identified in the AST. Additionally AST can also reveal a sequence of statements in the source code so that the order of method on a sequence diagram can be identified.

AST uses DFS post-order to traverse the tree. This is most suitable for reverse engineering process. With DFS post-order algorithm, the visits of statements in the AST is ordered properly according to the sequence in the source code. There are some complexities in implementing the AST traversal algorithm. The complexity is when the algorithm has to find another node from unknown location. To solve this problem, we record or mark each node that has the possibility to get searched again. With this solution, when the node is needed again then no need to search it through the AST but we can search it at the nodes that have been recorded previously.

For the future development, we expect that this process of reverse engineering can be ported to other programming language. To do this, we are planning to propose a modified process and a tool that can modify the generated AST into intermediate formats such as XML. The format here is not the AST from each language, but we take AST nodes from each language that involved in the establishment of the sequence diagram then change it into the intermediate format. Besides that, further application development can also be extended to support round-trip-engineering. By supporting a round-trip-engineering, modified sequence diagram can directly affect the source code (the code is changed too). In this way, an IDE (Integrated Development Environment) tools can adopt this technique. We believe that by employing this feature, it could be easier to perform the maintenance process.

VII. REFERENCE

[1] L. Feijs, R. Krikhaar, and R. Van Ommering, "A relational approach to support software architecture analysis," *Softw. Pract. Exp.*, vol. 28, no. 4, pp. 371–400, 1998.

[2] F.-M. Lazar and O. Banias, "Clone detection algorithm based on the Abstract Syntax Tree approach," pp. 73–78, 2014.

[3] L. Jiang, Z. Zhang, and Z. Zhao, "AST Based JAVA Software Evolution Analysis," *Web Inf. Syst. Appl. Conf. (WISA), 2013 10th*, pp. 180–183, 2013.

[4] H. Kikuchi, T. Goto, M. Wakatsuki, and T. Nishino, "A source code plagiarism detecting method using alignment with abstract syntax tree elements," *2014 IEEE/ACIS 15th Int. Conf. Softw. Eng. Artif. Intell. Netw. Parallel/Distributed Comput. SNPD 2014 - Proc.*, 2014.

[5] W. Xin and Y. Xiaojie, "Towards an AST-based approach to reverse engineering," *Can. Conf. Electr. Comput. Eng.*, no. May, pp. 422–425, 2007.

[6] E. Korshunova, M. Petkoviè, M. G. J. Van Den Brand, and M. R. Mousavi, "CPP2XMI: Reverse engineering of UML class, sequence, and activity diagrams from C++ source code," *Proc. - Work. Conf. Reverse Eng. WCRE*, pp. 297–298, 2006.

[7] A. Mazoyer, A. Serebrenik, M. G. J. van den Brand, S. Roubtsov, and E. Roubtsova, "I2SD: reverse engineering Sequence Diagrams from Enterprise Java Beans with interceptors," *IET Softw.*, vol. 7, no. 3, pp. 150–166, Jun. 2013.

[8] E. J. Chikofsky and J. H. Cross, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Softw.*, vol. 7, no. 1, pp. 13–17, 1990.

[9] Omg, "OMG Unified Modeling Language TM (OMG UML), Superstructure v.2.3," *InformatikSpektrum*, vol. 21, no. May, p. 758, 2010.

[10] K. Fakhroutdinov, "UML sequence diagram combined fragment." [Online]. Available: http://www.uml-diagrams.org/sequence-diagrams-combined-fragment.html. [Accessed: 16-May-2016].

[11] M. Fowler and B. Mcwhirter, *The Definitive ANTLR Reference*, vol. 67, no. 4. Dallas, Texas: The Pragmatic Bookshelf, 2007.

[12] J. Ebert, V. Riediger, and A. Winter, "Graph Technology in Reverse Engineering, the TGraph Approach," *10th Work. Softw. Reengineering*, vol. 126, pp. 67–81, 2008.

[13] O. N. Volgin, "Control Flow Analysis for Reverse Engineering of Sequence Diagrams," 2005.

[14] A. Rountev, O. Volgin, and M. Reddoch, "Static control-flow analysis for reverse engineering of UML sequence diagrams," *ACM SIGPLAN/SIGSOFT Work. Progr. Anal. Softw. Tools Eng.*, pp. 96–102, 2005.

[15] Y. Labiche, B. Kolbah, and H. Mehrfard, "Combining static and dynamic analyses to reverse-engineer scenario diagrams," *IEEE Int. Conf. Softw. Maintenance, ICSM*, pp. 130–139, 2013.

[16] P. Tonella, "Reverse engineering of object oriented code," *Proceedings. 27th Int. Conf. Softw. Eng. 2005. ICSE 2005.*, no. 2, pp. 1–2, 2005.