1. Introduction

Due to the increased size of the software products, the complexity in debugging or maintaining the software products is also increased. Program slicing is used to extract out the irrelevant parts of a program as per the criteria specified by the user and has given a positive impact in different areas like maintenance, debugging etc with reduced complexity. Apart from these, the technique based on program slicing is used in prioritization of the test cases while testing so as to find the affected test cases after modification of the program. Slicing can also be done at designing phase. By using the program slicing techniques in the diagrams of UML, the complexity of the diagram is reduced and became easy to trace the problem. Lot of research has been done for program slicing and for slicing or extraction at designing phase in the form of architectural slicing or model slicing but mingling of slicing at designing phase with program slicing is done here, that is not done before as per our knowledge. The architecture of the software defines its high level design structure. Slicing in the architecture or models allow us to see the part of software needs to be tested again or the parts of software that are affected by any change done in software etc in its designing phase only. Extraction done at designing phase eases the work at implementation level. Slicing at designing phase can bring improvement in various phases like interaction between the objects in object oriented software can be visualized, relationship between specific class and other classes in a UML system model can be discovered etc. Model slicing means to decompose the large model into sub models and extracting out the sub models that are not related to criteria specified by the user. A criterion to be specified for model slicing is different from the criterion to be specified for Program slicing. Model slices are defined via generalized slicing criteria e.g. method c in case of class diagram or variable c used in conditional predicate of message guard condition in case of sequence diagram etc whereas the slicing criteria in program slicing is to specify the statement number giving the wrong output. The input value given at the time of testing can also be added to the criteria i.e. in case of dynamic slicing. Dynamic Slicing is one of the Slicing technique, other than dynamic slicing there are several other program slicing techniques like static slicing etc but dynamic slicing reduced the size of
the slice compared to other slicing techniques. Similarly, Model slicing can be applied with any of its seven diagrams i.e. slicing can be done on sequence diagram, on class diagram etc. This paper is based on the idea to further reduce the complexity by doing the possible extraction before the actual implementation of a program. Slicing is done at designed models and then program slicing is applied on the code related to sliced model. Here Class diagram is considered for applying slicing at designing phase and dynamic slicing is considered as a program slicing technique due to its reduced slice size. Program slicing has numerous applications; here the major application of slicing i.e. finding the statement containing bug is focused. Class model of large systems comprises of thousand of classes, which neither can be viewed at one time nor is beneficial for maintenance work. Using a case study, slicing is first applied to the class diagram of a sample program to generate subset of UML class model based on a user specified criteria and then dynamic slicing is applied on the code related to remaining model. It is shown that the size of the sliced code is reduced as compared to the sliced size if dynamic slicing is applied directly on the code. This paper is organized as follows. Section 2 gives the brief overview regarding the Model Slicing, Program Slicing and about how the graph can be transformed back to the code. Section 3 presents the steps of the methodology adopted in slicing the code with mingling of model slicing and program slicing that are implemented manually by taking a sample program in section 4 and proves that the size is reduced by adopting the methodology. Finally Section 5 concludes the paper with the discussion on the scope of work that can be carried in future.

2. Literature Survey

2.1 Brief Analysis on UML Model Slicing
Model slicing means to extract the sub model from model diagrams corresponding to the point of interest. UML models are differentiated into structural models to describe relations among object and behavioral models to describe sequence of actions. Information of the system is distributed across several model views. Based on different diagrams various methods have been presented to slice the models. UML class models are very useful in understanding the system, but for large systems UML class models comprises of so many classes that cannot be viewed properly hence coming the need of slicing. The idea to slice the model is to enable the users to break the information into smaller parts so as to examine properly. The complete UML model is distributed can be presented via a large number of small UML class diagram which can be united to represent the entire systems design. The use of converting the model designs into sub model is to describe the features or concerns of the system design separately. Designs of these sub-models were initially developed manually then introduced the method to automatically generate subset of a UML class model based on a user specified criteria. UML Model Slicing is thus a process of breakdown to extract and identify relevant model parts or related to the criteria specified by the user. There are special slicing tools like UML etc available that can be used directly for slicing the UML diagrams.

The brief overview of slicing done on UML model other than class diagram is as follows-

Scenario oriented program slicing method is presented to slice the program under specified execution scenarios that are specified through UML sequence diagram. Sequence Diagram represents the behavior of programs by representing the sequence of messages passing along a time line. Sequences diagram slicing is introduced to analyses data dependency more precisely and handles the information about thread and exception to acquire the capability to be used in a wider range of applications. Tool Reticella is introduced to visualize the object oriented program’s behavior and supports sequence diagram slicing.

Different analysis method is presented using behavioral diagram that can be used to assure the quality of UML model.

2.2 Program Slicing
Program slicing means to get a sliced program that contains the statements relevant to the criteria specified by the user. It is different from Model Slicing in sense that it extracts the program statements that affects or may be affected by the statement mentioned in the criteria whereas the model slicing extract the subs model that implement the method containing the bug.

2.3 Graph Transformation
Converting the code to UML diagram is to represent the complete system graphically for better understanding.
Graph transformation is transforming the diagram back to the code. To transform the UML diagram to code, FUJABA tool[^14] is used that combines class diagram, UML activity diagram and a graph transformation language and offers a formal, visual specification language that can be used to completely specify the structure and behavior of a software system under development. Combining the diagrams is needed because class diagram specifies the structure but lacks to specify behavioral information that is specified by story diagrams i.e., a combination of UML activity diagrams and Story Patterns. Story Patterns are graph transformation rules specifying modifications of object structures (models). Based on these structural and behavioral specifications executable code (e.g., Java code) is generated by FUJABA.

Overview of FUJABA: FUJABA is an Open Source UML CASE tool that supports forward and reverse engineering and hence known as From UML to Java and Back Again. It generates code from the UML diagrams. FUJABA is an ongoing topic of research today; at least six rather independent tool versions are under development in Paderborn, Kassel, and Darmstadt for supporting, reengineering, embedded real-time systems, education, specification of distributed control systems, integration with the ECLIPSE platform, and MOF-based integration of system (re-) engineering tools[^15].

3. Methodology

The purpose of this paper is to unite the two techniques i.e., Model Slicing and Program Slicing so as to reduce the complexity and improve the results as compared to the complexity and results of the two when used separately. To fulfill the purpose, the steps are discussed below which has been implemented manually at this stage.

**Step 1: Create UML Model:** In case of new system to be developed, UML model is created at designing phase and in the case of locating faults in the system or to work with existing source code, the source code can be converted to class diagram discovering relationships between classes. It can be done using tools like magic draw, visual paradigm, UML lab etc. Figure 1a Shows the sample program whose class diagram is created using Visual Paradigm Software[^16] as shown in Figure 1b.

```cpp
#include <iostream>
using namespace std;

class Rectangle
{
  protected:
    float length, breadth;
  public:
    Rectangle(): length(0.0), breadth(0.0)
    {
      cout<<"Enter length: ";
      cin>>length;
      cout<<"Enter breadth: ";
      cin>>breadth;
    }
};

// Area class is derived from base class Rectangle. */
class Area : public Rectangle
{
  public:
    float calc()
    {
      return length*breadth;
    }
};

// Perimeter class is derived from base class Rectangle. */
class Perimeter : public Rectangle
{
  public:
    float calc()
    {
      return 2*(length*breadth);
    }
};

int main()
{
  cout<<"Enter data for first rectangle to find area. 
";
  Area a;
  cout<<"Area = "<a.calc()<<" square meter\n";
  cout<<"Enter data for second rectangle to find perimeter. 
";
  Perimeter p;
  cout<<"\nPerimeter = "<p.calc()<<" meter\";
  return 0;
}
```

(a)
Step 2: Apply Model Slicing: It is to be done to find the class that implements a method with a known bug and to extract out the other classes as they don't relate to the criteria. In the diagram, traverse back from that class to the starting point to extract out all the classes that are not related to the bug. It will give us the sliced Model.

Step 3: Code Transformation - To transform the Sliced Model to Code: UML Lab offers software developers a complete and reliable adjustment of source code and diagrams. For generating code from diagram, the combined generation of two tools i.e. UML lab and FUJABA that is consistent with each other is recommended. UML lab does not generate code for story diagrams and the code generation of FUJABA is not automatically linked to the previously reverse engineered classes. So First the FUJABA generated source code for story diagrams, then it is passed to UML lab and integrated into its generated source code.

Step 4: Apply Program Slicing: After step 3 the resultant code obtained will be of reduced size that is remained after extracted out the classes that do not relate to the specified criteria. In this step the code is further reduced by applying the dynamic slicing that will provide the sliced code with only those statements that can affect or

Figure 1. (a) Sample program to calculate the area and perimeter of rectangle. (b) Class diagram of sample program.

Figure 2. Flowchart of the steps of the Methodology.
is affected by the criteria. After completion of a step 4 the output will be sliced code. The steps discussed above have been shown in the form of flowchart in Figure 2.

4. Case study and Implementation

The program to calculate the area and perimeter of rectangle using class is taken as a sample program shown in Figure 1a. The statement numbers are added to the program as shown in Figure 3a. And taking each statement number as a node the dependency graphs is created shown in Figure 3b. Such that node x is connected to node y means that ‘y’ is dependent on ‘x’. Various related graphs are available like PDG(Program Dependence Graph)\textsuperscript{19} for showing dependency between the statements within the procedures, SDG(System Dependence Graph)\textsuperscript{20} that extend PDG to represent the inter-procedural programs, CDG(class dependence graph)\textsuperscript{21} to representing the dependencies of distinct classes etc. Firstly the dynamic slicing has been applied directly on the source code using

![Program with statements number](image1.png)

![Program dependence graph of sample program](image2.png)

**Figure 3.** (a) Program with statements number (b) Program dependence graph of sample program.
its dependence graph and then the steps discussed in the methodology is applied on same program to show that size and effort can be reduced further by the methodology proposed in this paper. It is supposed that the wrong output is coming of ar_sq at statement 21 when the value 4 is given to length. The criteria will be \( <21, L, 4> \) i.e. value 4 is given to l at 21 statement number. To get the slice, traverse back the dependence graph from node 21 to the starting node that will give the nodes i.e. the statement numbers that affects or are affected by the criteria. In sliced code, the nodes related to the class ‘perimeter’ are included as the calc() function is used by the both. But in case the output of perimeter is coming to be correct then there is no use of containing those nodes. Secondly the methodology is followed by applying model slicing first, Now, the perimeter class is ignored as shown in Figure 4a.as the bug is in the class ‘Area’ which is not connected to the class ‘Perimeter’. The statements related to the class

![Diagram](image-url)

**Figure 4.** (a) Sliced model (b) Code related to transformed model.
perimeter are hence not included in the code obtained by transforming the graph Figure 4b. Now applying dynamic slicing will lead to reduced size and reduced effort as compared to the dynamic slicing applied on the complete source code.

5. Conclusion and Future Scope

Proposed methodology is depicted to reduce further the complexity level than any program slicing technique that slice the source code. Case study is done to first slice a program code directly using dynamic slicing and then to slice the same program code by applying the discussed methodology. It is shown that instead of applying slicing directly, the slicing done on the code obtained by transforming from the sliced model allows us to obtain the result with reduced effort. The Case study is implemented manually. In future algorithmic steps will be implemented to create a tool that will take code or UML as input and will provide sliced code as output with less complexity. This paper has taken fault localization as a single application, other applications like improvement in the changeability and reusability of source codes can be added.

6. References