A REVIEW OF SLICING TECHNIQUES IN SOFTWARE ENGINEERING

[1]Asadullah Shah, [2]Ali Raza, [3]Basri Hassan, [4]Abdul Salam Shah

[,1,2,3]*International Islamic University Malaysia*

[4]*Department of Computer Science*
Szabist, Islamabad
Pakistan
[1]*asadullah@iium.edu.my*, [2]*alirazarang@gmail.com*, [3]*basrihassan@iium.edu.my*

ABSTRACT

Program slice is the part of program that may take the program off the path of the desired output at some point of its execution. Such point is known as the slicing criterion. This point is generally identified at a location in a given program coupled with the subset of variables of program. This process in which program slices are computed is called program slicing. Weiser was the person who gave the original definition of program slice in 1979. Since its first definition, many ideas related to the program slice have been formulated along with the numerous numbers of techniques to compute program slice. Meanwhile, distinction between the static slice and dynamic slice was also made. Program slicing is now among the most useful techniques that can fetch the particular elements of a program which are related to a particular computation. Quite a large numbers of variants for the program slicing have been analyzed along with the algorithms to compute the slice. Model based slicing split the large architectures of software into smaller sub models during early stages of SDLC. Software testing is regarded as an activity to evaluate the functionality and features of a system. It verifies whether the system is meeting the requirement or not. A common practice now is to extract the sub models out of the giant models based upon the slicing criteria. Process of model based slicing is utilized to extract the desired lump out of slice diagram. This specific survey focuses on slicing techniques in the fields of numerous programing paradigms like web applications, object oriented, and components based. Owing to the efforts of various researchers, this technique has been extended to numerous other platforms that include debugging of program, program integration and analysis, testing and maintenance of software, reengineering, and reverse engineering. This survey portrays on the role of model based slicing and various techniques that are being taken on to compute the slices.

Keyword: Program Slicing, Model Based Slicing, Testing.

INTRODUCTION

In order to increase computer efficiency, a technique called slicing is introduced. Information regarding all the elements that part of slicing, along with the dependency between these elements is gathered from what is being sliced and why. There are two major steps that are a part of every slicing technique first, the program that has to be sliced and second, all the elements that are required for slicing are iteratively appended. The two major conditions that classify the slicing technique involve the programs, models that are being sliced and the

View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by The International Islamic University Malaysia Repository

purpose of slicing. In most cases achieving optimization and efficiency is the main reason behind slicing.The technique that is used for simplifying the programs by concentrating on the selective aspects of semantics is called program slicing (Tip 1995). It is used to reduce complexity of computer programs. The basic objective behind program slicing is that it divides the program into small parts based on certain criterion or point of interest. It deletes those parts of the program which do not affect the semantics of interest. If a certain chuck of a program has either a direct or an indirect effect on the variables or values that are computed at a certain slicing criterion, that it constitutes the program chuck with respect to that particular criterion. Slicing is applicable in fields like testing, re-engineering and program comprehension. It is also found in various steps of the program development life cycle, such as software testing, software debugging and software maintenance. For example, debugging the complete source code of a certain program becomes difficult and convoluted. Therefore, one can use slicing, method and remove those parts of the program that cannot cause that particular bug. Thus, saving time and increasing debugging efficiency. Until now, most of the researches are conducted on code based slicing (Kim et al. 2011). Due to the various advantages of slicing, it is now applied in the field of design, modeling because it helps in reducing the cost of model maintenance and model checking (Kim et al. 2011). A new language called the Unified Model language is used in order to create high level design structure (Felgentreff et al. 2014). This structure allows the architect to analyze and reason about all most all the properties of that system at an abstract level. UML is very easy to use and is very effective, thus it has become popular among the software designs to construct and represent the architecture of a software system. UML diagrams are used to explain both the behavioral and structural aspects of the different model diagrams. The structural parts of the architecture of software are defined by class diagrams, component diagrams or object diagrams. The relationship between the entities or the objects is explained by these diagrams. Whereas the state diagram, activity diagram or the sequence diagram explain the flow of work. They concentrate on the states, their sequence and their interaction. Use cases, which are used for testing of a program, are developed with the help of these behavioral models. However, this is not an exact definition of model slicing (Lallchandani & Mall 2011).

2 LITERATURE REVIEW

2.1 Introduction

As slicing has the ability to reduce program complexity, thus it is very useful in various stages of the software development life cycle such as testing, debugging and software quality assurance. It is basically a source of code that uses a manipulating and analyzing technique in order to identify a subprogram according to user specified point of interest. The automatic generation of the slice is generated by program slicing. A slice or a chuck of a program that is generated by program slicing includes all the program statements PS that effect variable V at a position in a certain Program P. For example, PS (b, 20) means that the slice all the program statements that affect the variable b in the line number 10. In short, all the statements that affect the program, to a certain point of interest become the part of that particular slice. Therefore, sometimes the data is dependent, thus there can be number of statements S that can affect the variable V. These are the statements that tell whether statement S is executable. The two point important aspects of a slice first, the behavior of the slice should be similar to the behavior of the original program(Weiser 1981). That is the main essence of the program should remain intact even after the deletion of certain set of codes. Second, the only way slice should be obtained is through deleting certain unwanted code lines from the original

program. You can't add line of codes in order to make the slice imitate the original code. There are different types or variants of program slicing such as backward slicing, dynamic slicing, static slicing, and forward slicing. Models play a vital role in the production of software, especially the design and specification. And this particular need has forced researchers to move from program slicing to model slicing. Model slicing has been inspired by the technique of program slicing as the need to model slicing has become very strong. Through model slicing, we extract a subset of model elements and this subset represents a model slice. The model slice transports many types of information and has various forms, depending on its purpose such as Architectural slicing and design slicing. The Architectural slicing focuses on the Architect of a program. There are many different languages that are a part of software modeling and the major one is UML. This is a very strong language that allows you to define both the structural model that defines the relationship among various objects. As well as the behavioral model that explains the sequence of action of software. Different tools such as EFSM Slicing Tool, SSUAM, UTG, UML Slicer and Archlice, UOST are also used for this very purpose.

2.2 Related Work

Information-flow relations, dataflow equations and dependence graph use a slicing technique called static slicing. In static slicing the program is statically analyzed in order to create slices. It does not require computer execution. According to Weisre static slicing approach all the required slices are generated iteratively of the dataflow equation. The stopping certain for this approach is the last relevant statement. All the slices are created by calculating consecutive sets of relevant variables. These variables have a specific node in the control flow diagram (Weiser 1981). You can create different slices according to different point of interest of different dependency graph (Tip 1995). The program dependency graphs were introduced by Ottenstein and Ottenstein and graph reachability analysis is done in order to produce slices of it (Ottenstein & Ottenstein 1984). In Bergeretti and Caree approach relational calculus is used for computing the slices (Bergeretti & Carré 1985). The problem with Wesier approach for slicing was, it did not take into account the fact that there are chances that a slice crosses the boundary of a function call. All the slices are computed within one function and because of this there is a huge chance that slicing generates a wrong point of interest. Thus, this will affect the control flow of the whole program and slicing loses its precision (Weiser 1981). Weiser also defined another type of slicing, in which he highlights the upstream side of the chosen slicing criterion. A slice contains statements that may affect the value of Variable V at a program point p. That is, these control predicates and statements also affect the point of interest in a program and he named it backward slicing (Weiser 1981). On the other hand, another slicing technique was introduced by Horwitz, Rep and Binkely. According to this technique, multiple classes, procedures and packages are used for computing slices and this procedure is called inter-procedural slicing. Dynamic dependency summary graphs that are produced during the execution of the programs are used by inter-procedural slicing algorithm in order to create slices. Inter-procedural algorithm computes slices either by doing two phases traversing of SDG or by computing the summary edges of SDG. Thus, this technique overcomes the problem presented in Weiser slicing method, i.e. it overcomes calling context problem (Ottenstein & Ottenstein 1984). In 1989 Reps and Bicker came up with a technique in order to determine the effect of modification in one part of a program onto another part of the program. They name the technique forward slicing. Unlike, the backward slicing which affects the point of interest, forward slicing contains set of

programs that are affected by the slicing criterion (Bergeretti & Carré 1985). The formal definition is: At a certain program point p, set of program variables V consists of all the predicates and statements in that program. These statements can be affected by different values of variables in V at the point p. A particular variable affects the downstream code. Another type of slicing is dynamic slicing which works very well on a particular execution of a program. It is a technique that is used for program debugging and program understanding. In dynamic slicing, statements that do not have any relevance to the point of interest are not included (Miller & Choi 1988). Korel and Laski were the first who introduced the idea of dynamic slicing in an iterative way (Korel & Laski 1988). Dynamic slicing can also be approached in terms of dynamic dependence relations and dynamic flow charts. A dynamic dependence algorithm was first introduced by Miller and Choi; later Gopal introduced a similar algorithm based on Bergeretti information: flow relation (Agrawal & Horgan 1990). Likewise, Korel and Laski's developed an algorithm using Gopal's dependence relations. Another algorithm that uses the approach of dependence graph in order to compute non-executable dynamic slices was developed by Agarwal and Horgan (Korel & Laski 1988) (Korel & Laski 1990). An efficient and effective method for program comprehension was introduced by Venkatesh and is known as quasi static method. This method incorporates both the static concepts as well as dynamic concept of slicing. With fixed input parameters the slice is dynamic, whereas with unconstrained variable input parameters the slice is static (Venkatesh 1991). Another effective technique is Decomposition slicing. The idea behind this method is to capture those statements of a program that are needed in order to calculate the values of the provided variables. The unwanted statements are deleted. The slicing criterion for this kind of slicing is built from the union of static backward slices and a single variable V. This technique is widely used for software maintenance (Gallagher & Lyle 1991). An extension of dynamic slicing is relevant slicing. This method computes all the statements that might affect the slicing criterion. It also uses the method of dynamic computing with the help of forward algorithm. The relevant slice contains all the statement that affects a particular variable with respect to a variable. Forward algorithm also makes relevant slicing very space efficient (Agrawal et al. 1993).Interface slicing is a technique that allows the programmer to create a new module that contains the required components. It is applied to the module in order to extract the subset of module's functionality. Normally only a part of the module is imported and there are many procedures and functions that are part of a module. Interface slicing produces a program that is generated from the original program, but it contains only the required statements. The module produced by the interface slicer contains the desired components that belong to the slice. Thus, making interface the superset of static slicing based on certain point of interest. This results in, inability of interface slicing to deal with variable that are affected by the point of interest, but are beyond the scope of that particular program (Beck & Eichmann 1993).

In order to find those statements that carry the effect of a certain variable to another variable, chopping is used. It uses the concept of both backward slicing as well as forward slicing. Chopping finds all the statements that carry the affect from source to sink. This method was further reduced by the use of different barriers (Krinke 2004).In this method, all the infeasible paths are discards thus generating a reduced program. This reduced program is created on the bases of certain set of initial states of a program. This set of initial states is created with the help of some set of conditions (Ning et al. 1994).

Another variation of static slicing is hybrid slicing. This technique uses the idea of dynamic slicing and it was first introduced by Gupta et.al. This method is smaller in size when compared to static slicing and it is very cost effective when compared with dynamic slicing. All the information required by this method is gathered from the debugging process of the program (Gupta & Sofia 1995). There is a generalized static slicing which is called end slicing. Unlike static slicing, generalized form of static slicing uses a set of points rather than a single point of interest. All the slices are created by the union of the static slices (Danicic & Harman 1997). Another slicing technique that is used to generate a sliced program which is smaller in size when compared to the orthodox program slicing is amorphous slicing. This technique simplifies the program. Because of its ability to understand what is required by the users and simplify the program on the bases of these requirements, it is a very useful slicing technique (Harman & Danicic 1997). In 1997, Sivagurunathan et al proposed an algorithm that was able to overcome the problem of incorrect slices due to the presence of data operations and I/O. In order to overcome this problem Sivagurunathan et al introduced an extra variable which is associated with the I/O operations. This variable makes the external state accessible to the slicer. However, there is a major problem associate with this solution, i.e. in order to make the I/O operations, accessible to the slicer; you have to the concept of transformation schema. The transformation schema maps the original program language to a new language. Later on, Tan and Ling proposed a solution for database operation. Their idea used the basic concept of Sivagurunathan et al algorithm. In order to constantly update the database operations, they used an implicit variable. Then Willmor et al came up with the solution of Database-Oriented Program Dependence Graph for updating the values of database operations. Two different types of data dependencies which are program-database dependencies are computed. These dependencies relate to all the non-database statements present in the database. These dependencies are then added into to the Program Dependence Graph and the resultant is Database-Oriented Program Dependence. The effect of one database statement on another is captured by these dependencies (Sivagurunathan et al. 1997). In architectural slicing there are three major dependency components and two phase algorithm. Dependency components are the connection dependency, the connector component dependency and last the additional dependency. The architectural algorithm works well on the software architectural dependency graph (Zhao 1997).To apply the slicing technique on software architecture, Architectural Information Flow Graph also has three types of information flow, namely Connector-component, Component-connector and internal low arcs (Zhao 1998). Nishimatsu et al. proposed a better and a more feasible idea as compared to static and dynamic slicing called the call-mark slicing. The main objective of this idea was developing a slicing technique which reduces the cost of dynamic slicing. The slices that are generated through this slicing method are smaller than the slices generated by the static slicing technique. Also, the slices generated by this method are less expensive as compared to the slices generated by the dynamic slicing method. All the procedure call statements that are executed in the PDG are marked with the help of dynamic information. Once the executed statements are marked, they are removed from the PDG, thus making a more precise PDG. Call mark slicing criterion is generated the same way as the static slicing criterion, but it is amplified with complete inputs. The precise PDG is then traversed using the standard static technique (Nishimatsu et al. 1999).

DSAS is another effective slicing technique. In this method a small number of connectors and components

are created for every slice and these connectors and components are generated according to the set slicing criteria. This technique is mostly used in situations where there is a large number of ports present and the invocation of these ports can affect the occurrence or a certain event or a value of a variable. The method removes all the events that are irrelevant according to a certain slicing criterion, only the most relevant are forward (Kim et al. 1999)(Kim et al. 2000). The basic concept of this approach is the mapping of software architecture that is generated by Architecture description language onto program statements, thus creating executable architecture. Information regarding the connectors, their parameters and their events is gathered when the dynamic slicer reads the ADL source code of the architecture and it takes slicing criterion as input. There are more than 24 different types of data dependences and the classifications of these dependencies are based on the various levels of complexity that are introduced by the pointers in a program. Incremental slicing uses this idea and it acknowledges the fact that all data dependencies are not equal. This method makes it easy for us to understand the slices as it ignores the weak data dependences in the start and focus on the strong data dependencies (Orso et al. 2001). Later on, the approach incrementally incorporates the weak data dependencies as well. The dependence cache slicing technique was proposed by Takada et al. and it uses the dynamic information in order to generate a precise PDG. The method permits us to prune the PDG in accord with the dynamic information, thus creating a PDG called PDGDS in two steps. PDGDS has only those data dependence relations that are possible with the input data. Dependence-cache slicing does not involve data dependence edges during the construction of PDGDS, and the edges that are added to PDGDS are calculated by a data dependence collection algorithm (Takada et al. 2002). In order to give a programmer more control of the construction of the slice, Jens Krinke introduced a new slicing technique. A method that allows the programmer to specify the part of the program that is to be traversed for the construction of the slice. In order to restrict the traversing of PDG, the programmers use barriers. Once a barrier is met it is important to end the process of locating the transitive closure of the program dependencies. The barriers which are specified through a set of nodes are included in the slicing criterion and this technique is very useful for program debugging (Krinke 2003). Two types of slicing technique came into existence because of research done on Extended Finite State Machines. First as deterministic and second was nondeterministic. Another research was done that isolated those parts of the model that might result in an error and the technique mainly focuses on data flow analysis. This approach was able to produce slices that were smaller in size as compared to the slices produced by EFSM. In order to automate the computation of slicing a tool was introduced (Korel et al. 2003). Path slicing is one of the slicing techniques to identify the statements that may change the path of a program during its execution. Path slicing is carried out along the paths in a control graph. In a control flow graph in computer programming, a path is a set of all possible inputs and can run numerous infeasible executions. Path slicing turns out to be a better option than program slicing. It is so because path slicing runs along a complete path of any execution program. This yields much better results than program slicing technique in computer programming (Jhala & Majumdar 2005).

Model slicing is another technique to identify the bugs in a given large UML models. This concept was initiated by Kagdi who took the initiative to divide the large models into UML class model. By analyzing these class models through proper understating and querying, it becomes easy to do maintenance of software. The approach of Kagdi was to

extract the class diagrams from the whole models. However, this class model lacks the explicit behavioral characteristics and demonstrated only the structural characteristics. In this technique, a model "M" is defined which is then analyzed on the basis of elements, relationship between these elements, and functions that these elements perform on the basis of relationships between them (Kagdi et al. 2005). Stop-list slicing technique, which almost functions like dicing technique, is another slicing technique which use variable that a programmer is never interested in to decrease their slice size. In a program, two kinds of variable exist normally. One is that carryout computation and the other is that assist in performing computation. Stop-list slicing aims at removing the variables from programs that are of no interest to programmer. In this way, this technique decreases the dependence graph by identifying and eliminating all variables in the stop-list set (Gallagher et al. 2006). Slicing technique can also be used on UML Sequence Diagram. Role of sequence diagram is to identify the series of time dependent interactions between components and different objects. After having a sequence diagram, complete functionality of the process can be visualized and test cases can be designed. This test data can be selected by extracting a conditional predicate from a program. Then this predicate is subjected to the slicer and test against different inputs and methodologies until a final solution is achieved (Samuel et al. 2005). A representation was introduced known as "Ctest" that was capable of generating test cases from a given UML communication diagram. First of all, a communication tree is required to be built out of communication diagram considering its data flow and control flow. A tool named UTG (UML Behavioral Test Case Generator) transforms the predicate selected from the tree according to Ctest representation to identify the test data. Tool takes the communication diagram in the form of xml format. Document parser class is responsible for parsing the XML file for various inputs and constructs a communication tree. Whereas Test Data Finder locates the test data in the form of string on the basis of parsed information (Samuel et al. 2007).

Another algorithm was formulated to generate the test cases on the basis of sequence diagram. This algorithm takes the first step of converting the UML sequence diagram into graphical representation named SDG (Sequence Diagram Graph). Graph based methodology was used to pass through SDG and to make test cases according to given message sequence path coverage criteria. A template was used to fetch the information related to a specific input and output. This use case template, along with class diagram and data dictionary, was utilized to fetch the pre and post scenarios for the test cases as well (Sarma et al. 2007).

A UML metamodel was introduced to deal with the complex UML metamodels. This technique focused on modularizing of large complex UML metamodels into small metamodels. This approach functions on the basis of diagram-specific metamodels. It extracts the diagram-specific metamodels from the complex UML metamodel. These diagram-specific metamodel contains less number of elements in them and carry less relationships. Against the set of key elements of "KEdt", this technique generates the metamodel "MMdt". Constituents of the key are analyzed to identify the model elements for criteria of slicing which are related to diagram (Bae et al. 2008) (Bae & Chae 2008).

Another technique was developed to produce the dynamic slices of a given UML model through integrated state based information. Architectural model slicing through MDG traversal concept was utilized to achieve this technique. This concept first draws a graph that is capable of fetching all information regarding dependency at different states of variables. Dynamic slices are generated on the basis of traversal of

model dependency graph that has all information about dependency variables. These dynamic slices come out to be very helpful in determining various sorts of characteristics of program like impact if the design is changed, and reliability prediction. Using this same tool, researchers also developed a prototype architecture slicing tool (Lallchandani & Mall 2008).

Later on, another approach was introduced regarding the pruning of metamodels. A given pruner get the input slicing criteria which may include classes, operations etc of a metamodel under observation. This slices the architecture and fetches all the information and dependencies. In this way, a pruner gives rise to an output which satisfies all the structural constraints (Sen et al. 2013) to produce slices for UML Architecture model which is static in nature (Lallchandani & Mall 2009). Afterwards, another concept was introduced that utilize the flow dependent graph FDG of a diagram of activity to produce dynamic slices through an edge marking method. It has the ability of producing the test data with high path coverage. This concept can also be implemented on concurrency and polymorphism (Samuel & Mall 2009).One more technique is to utilize the uml state machine to slice with a purpose of producing simple slices without making any change in model based on dataflow analysis and control analysis. It has also got the ability of defining pre and post conditions and semantic. After such marvelous start, this work can now be implemented on state and activity diagrams (Lano 2009).

Slice method is precise and can generate exact slice calculation which is based on high precision dependency of data using sequence diagram. It is also capable of supporting various forms of programs and tools for slice calculation on eclipse platform (Noda et al. 2009). Another technique is well known for supporting model analysis, maintenance and reactive system, and testing. This technique is named Statechart. It demonstrated the control data dependency and backward slices from graph. Statechart can also do program slicing for reactive and slice embedded system (Luangsodsai & Fox 2010).

Model Checking is one of the fully automated techniques that are used to decrease the size of model through the process of slicing. This technique utilizes the Behavior Tree Dependency Graph that fetches all the functional requirements and also the dependency between the modules and components. Slicing criterion is used in it that contains state-realization nodes which are responsible for updating the state of attributes and components to decrease model size and improvise the checking (Yatapanage et al. 2010).Verification Technique is another technique which is being greatly used to find put the accuracy and precision of a given model by dividing it into sub-models utilizing the slicing. This technique demonstrates the strength of the sub-models in two categories i.e. weak satisfiable and strong satisfiable. It can be implemented on other diagrams and models to find out the precision and satisfaction linked with that model (Shaikh et al. 2010). Slicing technique comes out to be very effective in terms of efficiency and time to check a model. It breaks the whole model into sub-models and then checks the each sub-model one by one. If at any stage, model breaks, then the model is unsatisfiable. If succeeds successfully, then the model is satisfiable. Moreover, it helps in identifying other useful properties in the model as well. Since after its development, this approach has been used greatly to check the results of a program and its speed (Shaikh et al. 2011b).Model Transformation is one slicing technique for the slicing of UML model which can be particularly implemented to those parts as well that has the properties of subset in them. This technique brings into usage the class diagrams, communicating sets of state machines, and individual state machines in order to slice the UML model. Client suppliers' relation is used to form a tree structure. Slicing is done between the invariants of class as well as

pre and post conditions of operation considering predicate P. Then different criteria are defined that are then compared to get conclusion. This technique focuses on keeping the structure same without making any significant changes in the program (Lano & Kolahdouz-rahimi 2010).There have been developed many verification tools for UML/OCl class. Among these tools, four tools are for verification purposes and two are for validation purposes. Verification tools include HOL-OCL, UML to CSP, UML to Alloy, and Alloy, whereas validation tools include USE and MOVE. Here, crucial point of verification in tool arises. This issue can be avoided by UOST slicing technique. Researchers use UOST slicing technique to increase the efficiency of this tool. In this way, a tool automatically categorizes the model into sub-models and then concludes the result whether the model is satisfiable or unsatisfiable. Later on, this test is subjected to other tools as well for proof (Shaikh et al. 2011a). Another technique that is being used now a day is novel technique on the feature model. In this technique, feature of slicing the model is done through the constraint of crosstree. In this technique, an algorithm denotes the sets of rules and underlying configurations (Acher et al. 2011). Dynamic backward slicing technique is being greatly into usage of researchers for the model transformation. In order to slice the model, model transformation language was used as a key step of this technique through Dynamic Backward Slicing. This technique takes the three inputs i.e. model transformation program, the model required for the operation of MT program, and the slicing criterion. It produces the output in the form of transformation slices and model slices. This conversion of model into the MT Language is done by three processes that are Graph Pattern, Control Language, and Graph transformation Rules. The whole algorithm keeps the track records so that it could provide traceability information between target models and source models. It slices the MT Program using these traces and also slices the model program at the same time (Ujhelyi et al. 2011) (Ujhelyi et al. 2012).Another technique is being used greatly in generation of data dependence graph. This approach highlights the hierarchy of the system and orthogonal problems linked with it. This whole process is carried out during the tracking of data dependency in slicing process of UML state machine diagram. This exhibits the hierarchy relationships in regions, control and parallel flows, and behavior states. Then in last, it fetches the data dependence graph (Kim et al. 2011).Dynamic slicing helps in slicing of model dependence graph on the basis of Dynamic Slicing of UML Architectural Model (DSUAM). This approach can be applied to get the alterations in designs, regression testing, grasping large architectures, and reliability prediction. It can be achieved by code based slicing techniques and remodeling the slicing model (Lallchandani & Mall 2011). Approach of model slicing technique was also utilized in automation of safety inspection system. A tool named “Safe Slicer” was introduced in order to achieve this. This tool uses the technique of model slicing to extract the safety concerned slices of design model. This tool has got the ability to trace the links that are essential to get the automated slices. Methodology and approach that has been used in this technique are the principle basis for the Self Slicer tool. This tool ensures that the information that is required to be inspected in design slices has been reduced and is precise and accurate (Falessi et al. 2011).

Using the slicing technique for state machine models of reactive systems and also for UML class diagrams incorporates new features of input or output events of interests. This emphasizes on reducing model semantically rather than syntactically. It also indicated the conditions during the path predicate coverage. Focus of this technique is on class rather than on models of class. A state machine is

required for the slicing process in this (Lano & Kolahdouz-Rahimi 2011). Utilizing the communication diagram, this produces the control flow graphs (CFG). It further utilizes communication dependence graph representation (CoDG), dynamic slicing algorithm of communication diagram, and edge marking to get results (Mishra et al. 2012). Slicing approach is the one that is used to improvise the verification process's efficiency. In this tool, parent model is divided into the sub models. All these sub models are then converted into the constraint satisfaction problem. Then these sub models are subjected to scrutiny of this technique which identifies the satisfiability or non-satisfiability of the sub model (Shaikh & Wiil 2012).Condition slicing was utilized in an approach for production of test casesfrom UML interaction diagram. In this approach, message guard position is identified first of all. Then on the basis of this information, condition slicing is used for development of test cases. Message dependency graph is built first of all in this. Then, on a predicated node, condition slicing is carried out on one of the predicate nodes. Guard condition from the message flow is implemented to generate the test cases (Swain et al. 2012).

Feature demonstrating makes use of set of specialist e.g. total union and slice give rise to many helpful advantages. It gives boosts to the productive packing partition. This approach starts with the changes into the predicate followed by the changes in these predicates into a slice feature model. Slicing procedure used in this technique is semantic as well as syntactic. This is why dissection of cross slicing obligation is done to identify the features that can be sliced and also those that cannot be sliced (Acher et al. 2012). In order to provide more elasticity in the configuration environment, a slice feature diagram was used to design three different input diagram changes. Input diagram keeps record of a better structure than a sliced diagram. It is absolutely syntactic as it does not involve the intersection of cross cutting constraints. Best thing about this technique is that it gives rise to valid configurations. However, if features are from more than one view, then it can become a problematic incident in this technique (Hubaux et al. 2011). With the help of program slicing technique, Domain Specific Model Language (DSML) is brought into usage to model a specific domain of slicer. Metamodels are introduced in this technique. This helps in development of two-level generic approach utilizing Kompren. Complier in the Kompren automatically produces the models for slicer function. Then this fetches the model slices ion its own from domain specific models (Blouin et al. 2012). Another approach was introduced for generation of test cases on the basis of domain abstraction. This technique is based on the syntactic abstraction as well as variable elimination utilizing the model slicing. Source model is considered to be the input along with the sets of abstract variables. Then these are reduced by syntactic abstraction that is then followed by abstraction semantically with a purpose of extracting abstract models. From these models are extracted the symbolic tests as per criteria. Three methodologies were presented for the identification of relevant variable and production of abstract models. Data flow dependency is the first in this number. Second approach involves the both flows, data flow dependency and control flow dependency. In third approach, data flow and partial flow dependencies are used to find strong and relevant variables. Principle behind this is to use the syntactic and semantic abstraction for the precise results in generation of test cases (Julliand et al. 2011).Unsatisfiability was the main advantage of this technique. After the actions of slicing technique, developers get the sub models from the original models. If there is any unsatisfiable sub model, this technique will diagnose the invariant hence making the developer able to get the bug or problematic constraint. In this way, this approach

increases the efficiency of the slicing technique in UML diagrams. Automatic correction or suggestion is delivered to the user automatically by this technique (Shaikh & Wiil 2013).Another slicing technique for the UML is to carry it out with OCL invariants. It initiates with the decomposition of the model into modes that are known as the fragments with split invariants and operations. It also has the ability to perform the quick analysis and reduces the time required by the previous techniques in analysis. This slicing approach was first applied to the metamodels. It becomes easier to indicate the interested sub models for the process if the invariant sub models are identified from this technique (Sun et al. 2013). This technique is the practical example of the ideology for the extraction of sub model from the architecture software. It has the ability to visualize the software model after slicing the sequence diagram (Singh & Arora 2013). This technique also has the depiction of semantic language independent framework as well as the technique for the confirmation of model transformation. This also represents a detailed analysis of different sorts of transformations. Best advantage of implementing this technique is the usage, verification and determination. This model has the ability of mapping the questions and also the invocation of implicit rules (Lano et al. 2014).

CONCLUSION

Literature review discussed in the previous section shows the techniques, tools, and approaches that are being used for model slicing and program slicing. It can be inferred from this literature review that Model Transformation Verification through Slicing, Data and Control Flow, UML Model Verification, Model Dependency Graph, B Model Dependency Graph, Metamodel Diagram, and featured based slicing are being used for model based slicing. It is difficult to slice the UML based architecture as it is sometimes developed on complex diagrams and information is distributed randomly in them. Further, implicit dependencies are also its part. In this regard, developer first has to develop the immediate construction in which information is spread evenly in the form of various architectural elements. Later on, these segments are utilized to put them on scrutiny of various aspects like changes in the design. Then we have to analyze this intermediary model for more enhancements towards the integration of state and activity models. This technique has emphasized on the chunk study for better and precise results. Although the areas like validation and verification of slicing are still required to be studied more in detail. Nevertheless, much work has been done related to these techniques. This paper has shed some light on some of the slicing techniques like backward slicing, static slicing etc. No doubt, implementing the techniques of slicing has brought many advantages along with them. This approach has opened a new dimension for the developers of object oriented, web applications, and content based. There were many problems in the real life of the researchers that seemed to be unsolvable. But now, after the development of such slicing techniques, a developer can come up with better ideas and architectures.

REFERENCES

Acher, M. et al., 2012. Separation of Concerns in Feature Modeling : Support and Applications. In *AOSD '12 Proceedings of the 11th annual international conference on Aspect-oriented Software Development*. pp. 1–12.

Acher, M. et al., 2011. Slicing feature models. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. Ieee, pp. 424–427. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6100089.

Agrawal, H. et al., 1993. Incremental Regression Testing. In *Proceedings of the Conference on Software Maintenance. IEEE Computer Society*. Ieee, pp. 348–357.

Agrawal, H. & Horgan, J.R., 1990. Dynamic Program Slicing. In *Proceedings of*

the ACM SIGPLAN'90 Conference on Programming Language Design and Implementation. pp. 246–256.

Bae, J.H. & Chae, H.S., 2008. UMLSlicer: A tool for modularizing the UML metamodel using slicing. In *2008 8th IEEE International Conference on Computer and Information Technology*. Ieee, pp. 772–777. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4594772.

Bae, J.H., Lee, K. & Chae, H.S., 2008. Modularization of the UML Metamodel Using Model Slicing. In *Fifth International Conference on Information Technology: New Generations (itng 2008)*. Ieee, pp. 1253–1254. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4492681 [Accessed November 29, 2014].

Beck, J. & Eichmann, D., 1993. Program and Interface Slicing for Reverse Engineering. In *15th international conference on Software Engineering. IEEE Computer Society Press*. pp. 509–518.

Bergeretti, J.F. & Carré, B.A., 1985. Information-Flow and Data-Flow Analysis of while-Programs. *ACM Transactions on Programming Languages and Systems*, 7(1), pp.37–61.

Blouin, A. et al., 2012. Kompren: modeling and generating model slicers. *Software & Systems Modeling*, pp.1–17. Available at: http://link.springer.com/10.1007/s10270-012-0300-x [Accessed November 29, 2014].

Danicic, S. & Harman, M., 1997. Program Slicing using Functional Networks. In *Proceedings of 2nd UK Workshop on Program Comprehension*. pp. 54–65.

Falessi, D. et al., 2011. SafeSlice : A Model Slicing and Design Safety Inspection Tool for SysML. In *ESEC/FSE '11 Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. pp. 460–463.

Felgentreff, T., Borning, A. & Hirschfeld, R., 2014. *Babelsberg: Specifying and solving constraints on object behavior*, Universitatsverlag Potsdam. Available at: http://www.jot.fm/contents/issue_2014_09/article1.html [Accessed November 29, 2014].

Gallagher, K., Binkley, D. & Harman, M., 2006. Stop-List Slicing. In *proceedings of the 2006 Sixth IEEE International Workshop on Source Code Analysis and Manipulation*. Ieee, pp. 11–20. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4026851.

Gallagher, K.B. & Lyle, J.R., 1991. Using Program Slicing in Software Maintenance. *IEEE Transactions on Software Engineering*, 17(8), pp.751–761. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=83912.

Gupta, R. & Sofia, M. Lou, 1995. Hybrid Slicing : An Approach for Refining Information Slices Using Dynamic. *ACM SIGSOFT Software Engineering Notes*, 20(4), pp.29–40.

Harman, M. & Danicic, S., 1997. Amorphous Program Slicing. In *Program Comprehension, 1997. IWPC'97. Proceedings., Fifth Iternational Workshop on*. Ieee, pp. 70–79.

Hubaux, A. et al., 2011. Supporting multiple perspectives in feature-based configuration. *Software & Systems Modeling*, 12(3), pp.641–663. Available at: http://link.springer.com/10.1007/s10270-011-0220-1.

Jhala, R. & Majumdar, R., 2005. Path slicing. In *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation - PLDI '05*. New York, New York, USA: ACM Press, p. 38. Available at: http://portal.acm.org/citation.cfm?doid=1065010.1065016.

Julliand, J. et al., 2011. B model slicing and predicate abstraction to generate tests. *Software Quality Journal*, 21(1), pp.127–158. Available at: http://link.springer.com/10.1007/s11219-011-9161-8.

Kagdi, H. et al., 2005. Context-Free Slicing of UML Class Models. In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*. IEEE, pp. 635–638.

Kim, H.-J. et al., 2011. Deriving Data Dependence from/for UML State Machine Diagrams. In *2011 Fifth International Conference on Secure Software Integration and Reliability Improvement*. IEEE, pp. 118–126. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5992010 [Accessed November 29, 2014].

Kim, T. et al., 1999. Dynamic Software Architecture Slicing. In *Computer Software and Applications Conference, 1999. COMPSAC '99. Proceedings. The Twenty-Third Annual International*. IEEE, pp. 61–66.

Kim, T. et al., 2000. Software Architecture Analysis : A Dynamic Slicing Approach. *ACIS International Journal of Computer & Information Science*, 1(2), pp.91 – 103.

Korel, B. et al., 2003. Slicing of state-based models. In *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.* IEEE Comput. Soc, pp. 34–43. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1235404.

Korel, B. & Laski, J., 1988. Dynamic program slicing. *Information processing letters*, 29(3), pp.155–163.

Korel, B. & Laski, J., 1990. Dynamic slicing of computer programs. *Journal of Systems and Software*, 13(3), pp.187–195. Available at: http://linkinghub.elsevier.com/retrieve/pii/0164121290900943.

Krinke, J., 2003. Barrier Slicing and Chopping. In *Source Code Analysis and Manipulation, 2003. Proceedings. Third IEEE International Workshop on*. IEEE, pp. 81–87.

Krinke, J., 2004. Slicing, Chopping, and Path Conditions with Barriers. *Software Quality Journal*, 12(4), pp.339–360.

Lallchandani, J.T. & Mall, R., 2011. A Dynamic Slicing Technique for UML Architectural Models. *Software Engineering, IEEE Transactions*, 37(6), pp.737–771.

Lallchandani, J.T. & Mall, R., 2008. Slicing UML architectural models. *ACM SIGSOFT Software Engineering Notes*, 33(3), p.4. Available at: http://portal.acm.org/citation.cfm?doid=1360602.1360611.

Lallchandani, J.T. & Mall, R., 2009. Static Slicing of UML Architectural Models. *ACM SIGSOFT Software Engineering Notes*, 8(1), pp.159–188.

Lano, K., 2009. Slicing of UML State Machines. In *Proceedings of the 9th WSEAS international conference on Applied informatics and communications*. World Scientific and Engineering Academy and Society (WSEAS), pp. 63–69.

Lano, K., Clark, T. & Kolahdouz-rahimi, S., 2014. A framework for verification of model transformations. *Formal Aspects of Computing*.

Lano, K. & Kolahdouz-rahimi, S., 2010. Slicing of UML Models Using Model Transformations. In *Model Driven Engineering Languages and Systems*. Springer, pp. 228–242.

Lano, K. & Kolahdouz-Rahimi, S., 2011. Slicing Techniques for UML Models. *The Journal of Object Technology*, 10(11), pp.1–49. Available at: http://www.jot.fm/contents/issue_2011_01/article11.html.

Luangsodsai, A. & Fox, C., 2010. Concurrent Statechart Slicing. In *Computer Science and Electronic Engineering Conference (CEEC),2010 2nd*. IEEE, pp. 1 – 7. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5606493.

Miller, B.P. & Choi, J.-D., 1988. A mechanism for efficient debugging of parallel programs. *Proceedings of the 1988 ACM SIGPLAN and SIGOPS workshop on Parallel and distributed debugging - PADD '88*, 23(7), pp.135–144. Available at: http://portal.acm.org/citation.cfm?doid=68210.69229.

Mishra, A., Mohapatra, D.P. & Panda, S., 2012. Dynamic Slicing of UML Communication Diagram. In *Advance Computing Conference (IACC), 2013 IEEE 3rd International*. IEEE, pp. 1394–1399.

Ning, J.Q., Engberts, A. & Kozaczynski, W., 1994. Automated support for legacy code understanding.

Communications of the ACM, 37(5), pp.50–57.

Nishimatsu, A. et al., 1999. Call-mark slicing: An efficient and economical way of reducing slices. In *proceedings of the the 21st International Conference on Software Engineering*. ACM, pp. 422–431.

Noda, K. et al., 2009. Sequence Diagram Slicing. *2009 16th Asia-Pacific Software Engineering Conference*, pp.291–298. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5358692 [Accessed November 29, 2014].

Orso, A., Sinha, S. & Harrold, M.J., 2001. Incremental slicing based on data-dependences types. In *Proceedings IEEE International Conference on Software Maintenance. ICSM 2001*. IEEE, pp. 158–167. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=972726.

Ottenstein, K.J. & Ottenstein, L.M., 1984. The program dependence graph in a software development environment. *ACM Sigplan Notices*, 19(5), pp.177–184.

Samuel, P. & Mall, R., 2009. Slicing-Based Test Case Generation from UML Activity Diagrams. *ACM SIGSOFT Software Engineering*, 34(6), pp.1–14.

Samuel, P., Mall, R. & Kanth, P., 2007. Automatic test case generation from UML communication diagrams. *Information and Software Technology*, 49(2), pp.158–171. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0950584906000474.

Samuel, P., Mall, R. & Sahoo, S., 2005. UML Sequence Diagram Based Testing Using Slicing. In *INDICON, 2005 Annual IEEE*. IEEE, pp. 176–178.

Sarma, M., Kundu, D. & Mall, R., 2007. Automatic Test Case Generation from UML Sequence Diagram. In *15th International Conference on Advanced Computing and Communications (ADCOM 2007)*. IEEE, pp. 60–67. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4425952.

Sen, S. et al., 2013. Meta-model Pruning. In *Model Driven Engineering Languages and Systems*. Denver, CO, USA: Springer Berlin Heidelberg, pp. 32–46.

Shaikh, A. et al., 2010. Verification-Driven Slicing of UML / OCL Models. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*. pp. 185–194.

Shaikh, A. & Wiil, U.K., 2013. A feedback technique for unsatisfiable UML / OCL class diagrams. *Software: Practice and Experience*, 44(11), pp.1379–1393.

Shaikh, A. & Wiil, U.K., 2012. UMLtoCSP (UOST): A Tool for Effi cient Verification of UML / OCL Class Diagrams Through Model Slicing. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, p. 37.

Shaikh, A., Wiil, U.K. & Memon, N., 2011a. Evaluation of Tools and Slicing Techniques for Efficient Verification of UML/OCL Class Diagrams. *Advances in Software Engineering*, 2011, p.5. Available at: http://www.hindawi.com/journals/ase/2011/370198/.

Shaikh, A., Wiil, U.K. & Memon, N., 2011b. UOST : UML / OCL Aggressive Slicing Technique for Efficient Verfication of Models. In *System Analysis and Modeling: About Models*. Springer Berlin Heidelberg, pp. 173–192.

Singh, R. & Arora, V., 2013. A practical approach for model based slicing. *IOSR Journal of Computer Engineering*, 12(4), pp.18–26.

Sivagurunathan, Y., Harman, M. & Danicic, S., 1997. Slicing , I/O and the Implicit State. *measurement*, 16(14), pp.59–68.

Sun, W., France, R.B. & Ray, I., 2013. Contract-Aware Slicing of UML Class Models. In *6th International Conference, MODELS 2013, Miami, FL, USA*. pp. 724–739.

Swain, R.K., Panthi, V. & Behera, P.K., 2012. Test Case Design Using Slicing of UML Interaction Diagram. *Procedia Technology*, 6, pp.136–144. Available at: http://linkinghub.elsevier.com/retrieve/pii/S2212017312005622.

Takada, T., Ohata, F. & Inoue, K., 2002. Dependence-cache slicing: a

program slicing method using lightweight dynamic information. In *Program Comprehension, 2002. Proceedings. 10th International Workshop on*. IEEE, pp. 169–177. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1021338.

Tip, F., 1995. A Survey of Program Slicing Techniques. *Journal of programming languages*, 3(3), pp.121–189.

Ujhelyi, Z., Horváth, Á. & Varró, D., 2012. Dynamic Backward Slicing of Model Transformations. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. IEEE, pp. 1 – 10.

Ujhelyi, Z., Horváth, Á. & Varró, D., 2011. Towards Dynamic Backward Slicing of Model Transformations. In *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference*. IEEE, pp. 404–407.

Venkatesh, G. a., 1991. The semantic approach to program slicing. *ACM SIGPLAN Notices*, 26(6), pp.107–119. Available at: http://portal.acm.org/citation.cfm?doid=113446.113455.

Weiser, M., 1981. Program Slicing. In *Proceedings of the 5th international conference on Software engineering*. IEEE, pp. 439–449.

Yatapanage, N., Winter, K. & Zafar, S., 2010. *Slicing Behavior Tree Models for Verification*, Springer Berlin Heidelberg.

Zhao, J., 1998. Applying slicing technique to software architectures. In *Engineering of Complex Computer Systems, 1998. ICECCS'98. Proceedings. Fourth IEEE International Conference on*. IEEE, pp. 87–98. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=706659.

Zhao, J., 1997. Software Architecture Slicing. In *In Proceedings of the 14th Annual Conference of Japan Society for Software Science and Technology*. Citeseer, pp. 85–92.