AUTOMATED ERROR CHECKING FOR UML CLASS AND SEQUENCE DIAGRAMS TOOL (CseqD)

By:

ELHAM BANDEGI (GS42583)

Supervised By:

DR. NORHAYATI MOHD ALI

Thesis Submitted to the School of Graduate Studies, Universiti Putra Malaysia, in Fulfillment of the Requirements for the Degree of Master of Computer Science

June 2017

COPYRIGHT PAGE

DECLARATION

I hereby confirm that:

- This thesis is my original work;
- quotations, illustrations and citations have been duly referenced;
- this thesis has not been submitted previously or concurrently for any other degree at any other institutions;
- intellectual property from the thesis and copyright of thesis are fully-owned by Universiti Putra Malaysia, as according to the Universiti Putra Malaysia (Research) Rules 2012;
- written permission must be obtained from supervisor and the office of Deputy Vice-Chancellor (Research and Innovation) before thesis is published (in the form of written, printed or in electronic form) including books, journals, modules, proceedings, popular writings, seminar papers, manuscripts, posters, reports, lecture notes, learning modules or any other materials as stated in the Universiti Putra Malaysia (Research) Rules 2012;
- there is no plagiarism or data falsification/fabrication in the thesis, and scholarly integrity is upheld as according to the Universiti Putra Malaysia (Graduate Studies) Rules 2003 (Revision 2012-2013) and the Universiti Putra Malaysia (Research) Rules 2012. The thesis has undergone plagiarism detection software.

Elham Bandegi

GS42583

Date: 29th May 2017

AKNOWLEDGEMENT

First of all, I am grateful to The Almighty God to establishing me to complete this project.

I wish to express my sincere thanks to my project supervisor, Dr. Norhayati Mohd Ali for her guidance, advices, supports, motivation and also her time that she spend that are very useful and helpful towards the success of this project.

I take this opportunity to record my thanks to all of my course mates, lecturers and friends who are willingly to share their time and knowledge in order to assist me to complete this project. Lastly, I would like to thank my family for their support and motivation while I am completing this project.

ABSTRACT

Unified Modelling Language (UML) is the standard formalism for software analysis and design. Most modelling tools used the UML notations. The modelling tools are available on market or online to support the software engineers in the software development process. However, most of the modelling tools are suitable for professional software engineers and not for novice developers or end-user developers. Novice and end-user developers always have the difficulty in the software modelling task. A good knowledge of UML is required in order to perform the software modelling task. Most UML modelling tools do not provide the functionalities to check whether the models/diagrams created by novice and/or end-user developers are valid and correct models/diagrams. Thus, the objective of this project is to propose and develop an automated checking tool for UML Class and Sequence diagrams (CSeqD). A detection method is employed in checking the Class diagrams and Sequence diagrams. A prototype of CSeqD tool is developed as a proof-of-concept. Evaluation of the prototype tool is conducted by checking several existing students' modelling projects with the CSeqD tool. The results showed that the design errors from the modeling projects can be detected by the CSeqD tool. Hopefully this project will benefit the novice and end-user developers in the analysis and modeling task.

ABSTRAK

Unified Modeling Language (UML) merupakan formalisme standard untuk analisis dan reka bentuk perisian. Kebanyakan alat pemodelan menggunakan notasi *UML*. Alat pemodelan yang terdapat di pasaran atau dalam talian adalah untuk membantu jurutera perisian dalam proses pembangunan perisian. Walau bagaimanapun, kebanyakan alat pemodelan adalah sesuai untuk jurutera perisian profesional dan bukan untuk pembangun baru atau pembangun pengguna-akhir . Pembangun baru dan pengguna -akhir sentiasa menghadapi kesukaran dalam tugas pemodelan perisian. Pengetahuan yang baik tentang UML diperlukan untuk melaksanakan tugas pemodelan perisian. Kebanyakan alat pemodelan UML tidak menyediakan fungsi untuk memeriksa sama ada model / rajah yang dibuat oleh pembangun baru dan/atau pengguna-akhir adalah model / rajah yang sah dan betul. Oleh itu, objektif projek ini adalah untuk mencadangkan dan membangunkan alat pemeriksaan automatik bagi rajah *UML Class and Sequence* (*CSeqD*). Kaedah pengesanan digunakan dalam menyemak rajah Kelas dan rajah Turutan. Prototaip alat CSeqD dibangunkan sebagai satu bukti-konsep. Penilaian alat prototaip dijalankan dengan memeriksa beberapa projek pemodelan pelajar yang sedia ada dengan alat CSeqD. Hasil penilaian menunjukkan bahawa kesalahan reka bentuk dari projek pemodelan boleh dikesan oleh alat *CSeqD*. Diharapkan projek ini akan memberi manfaat kepada pembangun perisian yang baru dan pengguna-akhir dalam tugas analisis dan pemodelan.

TABLE OF CONTENTS

CHAPTER 1 – INTRODUCTION

1.1 INTRODUCTION

Software development is a very complicated process. Developers should consider a lot of aspects in order to design and implement a high quality product. Software system development process involves many functionalities and each of them include several classes and objects which are related to each other and have interact with each other. It is very challenging for developers to detect and solve the errors and faults during the development process which can affect the next steps of development.

UML (Unified Modeling Language) is a standard language for software modeling which helps designers and developers by visualizing, specifying, documenting and constructing the artifacts of their software design. As a result, these artifacts become more secure, robust and scalable during execution. The importance of UML mostly is in object oriented software development approach. Seven types of UML diagrams represent the structural specification and the other eight diagrams represent the behavioral aspects of the software design. (I. Zaretska, O. Kulankhina & H. Mykhailenko, 2013).

Structural UML diagrams are included class diagram, component diagram, deployment diagram while use-case diagram, activity diagram, state-chart diagram, collaboration diagram and sequence diagram are in behavioral category.

Class diagram is one of the most important and critical UML diagram. Class diagram shows the static aspect of a system and describe responsibility of the application. It also use for deployment and component diagram and shows the relationship among the other system classes. (P. Herout & P. Brada, 2016).

In addition, in terms of the importance of designing sequence diagram during software design process (K. Noda, T. Kobayashi, K. Agusa, 2009) Mentioned that in design and programming process based on object-oriented approach one of the most challenging tasks is related to understand large-scale object-oriented system behavior. One of the beneficial techniques to contribute designers in order to understand the object-oriented system behavior correctly is visualization. The sequence of messages passed in a program along a timeline are represented by a sequence diagram also showed the behavior of object-oriented programs.

There are tendencies in software technology where applications are developed by non-professional software programmer. They write by people who are not expert in software domain but may be expert in other working domains. Statistics reported by (M. Bureaut, 2009) shown that there are around three million expert programmer and near fifty-five million people who are not professional in programming but use databases, modeling tools and spreadsheets and writing queries and formulas to support their job (ref please). Also represented in (M. Bureaut, 2009) study that indicates that close to twenty-five percent of business applications are developed by non-professional developers which are called end-user.

Another Issue which is important in this project is relate to end-user Development (EUD) definition. (M. Burnett, 2009) end-user Development (EUD) is one of the recent research topics in the computer science area. (EUD) is describing by activities and techniques which empower not professional developers to create and modify a software artifact. End-users who are intended to develop a system do not have fundamental knowledge of the UML diagrams especially class diagram may cause a problem and error during the development phases. There are lots of UML diagraming tools which end-users can use them to design their models.

In conclusion, it is very important for end-users to ensure that their modeling is correct. Because modeling the software system especially class and sequence diagrams are the basic and fundamental step of development process. So, it will be very helpful for end-user to use an automated error checking tools for UML diagram to be sure about the correctness of their design.

1.2 PROBLEM STATEMENT

Most of the existing UML tools are not appropriate for educational purpose as the target users of these tools are professionals based on before surveys done by other researchers. So. It is very challenging for the novices to use them since they are not professional users (S. Akayama, B. Demuth, T. C. Lethbridge, M. Scholz, P. Stevens, and D. R. Stikkolorum, 2015). There are so many modeling tools that the novices may find this tools with much features and syntax. Moreover, most of these tools do not support much error checking. It means that non-professional end-users will need more time and effort to learn modeling tool and cannot be sure about the correctness of their design.

In addition, the existing modelling tools do not provide end-users with any feedback on their structural defects or errors of their class diagrams. So, this may cause end-users to draw a class diagrams with structural errors

such as incorrect class name, attributes or operations. These hidden errors will affect other steps of the software development process and may cause fault and failure during implementation which exposes huge cost for the project. Also, the time for finding and solving the fault reasons will increase, as a result the quality of final product will reduce. So, it could be very beneficial to detect and solve the design problems in early stages of development process.

Moreover, Most of end-users who model their diagrams with UML tools do not have comprehensive knowledge to design their models with UML tools. So, they are not able to draw correct and error-free diagrams. In addition, the main target users of majority of existing tools are the ones who have knowledge about the modelling and UML notations. So there is need to develop a UML error detection tool which can help the end-users to correct their designs. This kind of tool can assist the end-users to learn about their common structural errors so that they will be able to apply the best practices during designing a class diagram for a software system. It is because the defects that exist during the designing of the class diagram can give effects to the development of a software system as the class diagram is being used as the reference during the development phase.

1.3 OBJECTIVES

This project involves the extension of previous project that has been developed by previous student [include Ref for the bachelor student]. The previous project only focuses on error checking of UML Class diagrams. However, the objectives of this project are as follow:

1.3.1 To propose a UML supporting tool that checks for design errors in UML Class Diagram and Sequence Diagram created by end-users/novice designers.

1.3.2 To develop an extension of automated error checking tool on UML class diagram and sequence diagram for educational purposes.

1.3.3 To evaluate the functionalities of UML Class and Sequence Diagrams tool in detecting the design errors/mistakes.

1.4 PROJECT SCOPE

The scope of this project is focused on checking design errors in UML Class diagrams and Sequence diagrams only. The errors checking is done in an automated way.

Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) is an independent application which is implemented

in desktop computers. This application tool is extended by windows 10 operating system currently and will be tasted by same operating system.

The main purpose of extension of this tool is for educational purposes, and the target user of this tool will be end-users or novice designers. Based on personal experience in using UML modeling as an end-user, it could be useful for end-users to access the automated tool that is able to assist them in order to detect their UML diagrams mistakes and errors. Beside, this tool helps end-users to have more understanding about the UML diagrams by their own, and figure out their weakness area in diagraming according to their common mistakes which they have done.

The main purpose of automated error checking for UML Class and Sequence Diagrams Tool (CSeqD) is to assess class and sequence diagrams which are created by Software Ideas Modeler tool. Software Ideas Modeler is one of the wildly spared modeling tools which is used by end-users to design UML diagrams. The output of Software Ideas Modeler can be exported in ".simp" (as the extension file) and the (CSeqD) only can be worked with same extension file.

1.4 THESIS STRUCTURE

This thesis includes six chapters which will be started with the Introduction as the first chapter and will be ended with Conclusion as the sixth chapter. Each chapter will discuss as following:

CHAPTER 1 INTRODUCTION: The project thesis introduction including project's introduction, project's problem statement, project's objective and the scope of the project is discussing in this chapter.

CHAPTER 2 LITERATURE REVIEWS: The purpose of this chapter is to identify the researches that have been done by reading the journals, thesis, websites or any other appropriate resources that related to the project like the existing systems or about the techniques and methods used during the implementation of the project.

CHAPTER 3 METHODOLOGY: In this chapter the methodology which being used for this project, not only just give the justification about the methodology but also give explanation in details about each phases that involved in methodology that have been chosen and relates it to the project will discuss in details.

CHAPTER 4 SYSTEM DESIGN AND IMPLEMENTATION: The main focus of this chapter is on the design and implement phases of the project, including all of the diagrams which is related to the project, such as the use-case diagram, architectural design, interface design and also the algorithm or technique which will use in the project. After that the explanation in details about the implementation steps in order to development of this project, including all the software and hardware tool that being used from the beginning of the implementation until the end.

CHAPTER 5 SYSTEM EVALUATION, TESTING AND DEPLOYMENT: This chapter is involved the explanation about the evaluation technique that is used and then testing activity that being conducted after the evaluation activity ends in order to test the project functionalities. This chapter will be end by the discussion about the deployment activities involved in producing final product.

CHAPTER 6 CONCLUSION: In the last chapter of this thesis, the whole project in detail all will discuss in terms of the advantageous and the disadvantages of the project and also suggestion about any ideas on the ways of improvement the project for the future.

CHAPTER 2 - LITERATURE REVIEW

According to the study of (I. Zaretska, O. Kulankhina & H. Mykhailenko, 2013) software design has become one of the most important stages of software lifecycle due to the increasing complexity of software under development. The Unified Modeling Language (UML) is the significant standard for modeling software systems. The UML supports a wide range of diagrams for modeling software. UML diagrams are independent but connected. Their meta model describes them under a common structure. Moreover, detecting of faults and errors in the software design stage contributes in reducing a great number of problems in the late stages of software life cycle.

A number of UML visual design tools provide model verification support including syntax checking, structural and consistency analysis. There are various definitions of a design supporting tools in the literature. A supporting tool can be considered as a user interface that evaluates a design made by an end-user and provides feedback to assist the user to improve the design. Generally, supporting tools detect potential problems, give advice and alternative solutions and possibly automated or semi-automated design improvements to the end-users. Design supporting tools have been used in design tools for various domains, including software engineering, design sketches, education, etc. Several studies report the benefits of applying

design critic tools in software developments activities. (I. Zaretska, O. Kulankhina & H. Mykhailenko, 2013)

2.1 EXISTING SYSTEMS

The emerging of Java and .Net programming languages, object oriented approach has becoming very popular and important in the world of programming. But, still there are many errors and faults were introduced into the applications. And the reason why is that, there are many different standards, developers and analysis that could easily share the various correct modelling and designing diagrams for the same question(J. Brewer and L. Lorenz, 2003).

We should have a complete understanding about the software system development process to have a better understanding of object-oriented paradigm or OOAD (Object-Oriented Analysis and Design). The Unified Modeling Language (UML) can be used to help us to specify, visualize and document models of software system including their structure and design. The UML consist of eight diagram types which one of them is the class diagram. The class diagram is a type of diagram that will describe the structure of the system in the static structure by showing the system's classes and the relationships between them (J. Soler, I. Boada, F. Prades, J. Poch and R. Fabregat, 2010).

There are several number of modeling tools that can be used by end-users to model their UML diagrams such as software Ideas Modeler, ArgoUML and Rational Rose. A description about some of these modeling tools that can be used by end-users to model their design is provided as follow.

- UMLGrader (R. W. Hasker & Y. Shi, 2014)
- UMLint (R. W. Hasker, A. Rosene, and J. Reid, 2012)
- NClass (B. Tihanyi, 2011)
- Software Ideas Modeler (D. Rodina, 2009)
- minimUML (Turner et al, 2005)
- UML/Analyzer (A. Egyed, 2007)
- StudentUML (E. Ramollari & D. Dranidis, 2007)
- UMLet (Auer et al, 2003)
- ArgoUML (A. E. Robbins. 2003)
- QuickUML (E. Crahen, C. Alphonce, and P. Ventura, 2002)

2.1.1 UMLGrader

One of the tools for teaching class diagram notation in the Unified Modeling Language (UML) is UMLGrader which is proposed by (R. W. Hasker & Y. Shi, 2014). UMLGrader is a web-based tool which is based on UMLint. Designed diagrams by students import to the tool. Then, imported diagrams compare against standard solutions and the tool provide students with feedback on errors and lost elements of their design. According to the evaluation which is done in this study, UMLGrader is useful in order to teaching class diagram to the students. However, the tool just can support diagrams which is made by IBM Rational Rose and IBM Rhapsody right now.

2.1.2 UMLint

UMLint (R. W. Hasker & M. Rowe, 2011) is an automated tool for finding defects in UML diagrams specifically in use case and class diagram. This tool provides improvement in object-oriented models which is developed by students. Because Standard tools like IBM Rational Rose deliver little feedback on the quality of designed model, so students should wait for feedback from their teachers. The main contribution of UMLint is to fill this gap by identifying common defects made by students.

UMLint is available as a web service. Students browse their model file through the website, then click a button and a list of defects will appear in

a priority order. More detailed description of the detecting defects as well as recommendation on how to improve the model are provided by “More Information” link. UMLint has two main goal. The first one is helping students to avoid common errors. On the other hand, the tool helps students to correct their errors before submission time. As a result, when students are able to find and fix their simple mistakes, during the time they will be able to examine their diagrams and detect deeper errors.

2.1.3 NClass

Nclass (http://nclass.sourceforge.net/) also is another software tools which can help in designing a class diagram. The most important purpose of Nclass is to provide a powerful, simple class diagram which is very easy to use. One of the ability of Nclass is to provide users with generated code from designed models. Nclass can also suport both C# and Java language platforms. In addition, this software tool also able to reverse engineering from .NET assemblies. Another functionality of the NClass is that it provides simple and understandable user interface and also printing/saving the diagram into image functionality. However, NClass can be considered as software tool that not quite suitable for the educational purpose.

2.1.4 Software Ideas Modeler

Software Ideas Modeler (https://www.softwareideas.net/) is one of the existing designing tools that are supports all 14 diagram types of UML diagrams that being specified in UML 2.4. Moreover, this tool also can be used to draw SysML (System Modeling Language), BPMN (Business Process Model and Notation) diagrams, ERD and other diagrams which are used in the professional fields. Software Ideas Modeler is an easy, fast and intuitive tool to use. Other common functionalities of this tool are XMI import/export, documentation generation, print function, source code generation and exports diagrams to various image formats.

2.1.5 minimUML

minimUML is one of the educational software tools which is uses the minimalist approach .minimUML covers the small portion of the subset of the UML notation as it is just enough for the introductory (Turner et al, 2005). minimUML only support class diagam and provide classes, connection among cases and two types of annotations. Other functionalities of minimUML are undo/redo function, clipboard support, code generation and printing function, together with the “drag and drop” function.

2.1.6 UML/Analyzer

In software design process one of the important best practices is to provide error feedback. There are many tools which can provide

consistency checking for UML design models. However, most of the tool do not provide instant feedback on design errors. The problem is that make correct decision about what consistency rules should be considered when the model is changing seems impossible.

UML/Analyzer (A. Egyed. 2007) is an instantly consistency checking for UML models. The tool provides designers with tracking and detecting inconsistencies with all changes in design quickly. The tool is completely automated and does not need any manual assistance. The tool is able to deliver feedback in both intrusive or non-intrusive way. This tool can check both batch consistency of whole UML model also incremental consistency of design change. In order to apply incremental check of design change fast, the tool should be able to detect all elements of model which affect the value of all consistency rules. The tool requires to reevaluate consistency rule if at least one of these model element change.

2.1.7 StudentUML

StudentUML (E. Ramollari and D. Dranidis, 2007) emphasize on the educational environment, simplicity and capability of the tool satisfy correctness and consistency of the UML models, specially class diagram. The main goals for development of StudentUML are create a supporting tool which provide correctness, simplicity and consistency that can support students in the process of software development. StudentUML is written by

Java programing language but it is platform independent. There is a main project toolbar in the tool's graphical user interface which provide user with several functionalities such as open, close and save the complete projects. Users also are able to export their diagrams by different image formats. Users can use the option of automatic check consistency between diagrams.

2.1.8 UMLet

UMLet (Auer et al, 2003) is a simple UML tool which is helpful for the educational purpose as it has many of features and UML notation. This tool has a features that caused this tool is easy to use among the students, however it also the tool is not strong enough in flexibility to the check the consistencies among the diagrams which have been designated. In addition, the diagrams that have designated by UMLet are not managed as one development project and it also does support for forward or reverse engineering.

2.1.9 ArgoUML

ArgoUML is another software tools which can be used in modeling the UML diagrams (http://argouml.tigris.org/). The ArgoUML supports all 9 UML 1.4 diagrams that strictly follows the UML standard. More than that, ArgoUML also has the XMI support, several diagram export formats, code generation, diagram editing and design critics, reverse engineering also the

corrective automations which are especially useful for the educational purposes.

2.1.10 QuickUML

A tool called QuickUML introduced by (E. Crahen, C. Alphonce, and P. Ventura, 2002). This tool specially designed for educational purpose. The target group was students with great tendency to model their diagrams with UML but they did not know how to apply UML diagrams correctly. QuickUML support only class diagrams with a limit features and some part of code generation. Also this tool helps students to understand how a class diagram translate in to code. As a result, this understanding contribute them to develop a real system with higher quality and more quick.

Table 1 : Comparison of ten existing modeling tool.

Tools	UMLGrader	UMLint	NClass	Software Ideas Modeler	minimUML	UML /Analyzer	StudentUML	UMLet	ArgoUML	QuickUML
	R. W. Hasker & Y. Shi 2014	R. W. Hasker & M. Rowe 2011	http://nclass. Sourceforge .net	https://www. Softwareidea s .net	E. Ramollari and D. Dranidis 2007	A. Egyed. 2007	E. Ramollari and D. Dranidis 2007	E. Ramollari and D. Dranidis 2007	http://argouml .Tigris .org	E. Crahen, C. Alphonce, & P. Ventura, 2002
Class Diagram	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Sequence Diagram	✓	✓	X	✓	X	✓	✓	✓	✓	✓
Other UML Diagrams	✓	✓	X	✓	X	✓	✓	✓	✓	✓
Target User	Student	Student	Professional	Professional	Student	Professional	Student	Student	Professional	Student
Ease of use	Easy	Easy	Easy	Easy	Easy	Easy	Easy	Easy	Moderate	Easy
Guideline	✓	✓	X	X	X	X	✓	X	✓	X
Error detection Class disgram	✓	✓	X	X	X	✓	✓	X	✓	X
Error detection sequence disgram	X	X	X	X	X	X	X	X	X	X
Diagram consistency	X	X	X	X	X	✓	X	X	X	X

According to the TABLE 1, it can clearly see that most of the existing modeling tools are able to design the class, sequence diagram and other UML diagrams except for NClass and minimUML. These two tools just can be used only to design the class diagram. For all these ten existing modeling tools, four of them are used by the professionals, but they do not have guideline and error detection supports. However, for the ArgoUML, even though it is specialized for the professional, but still have guideline and error detection supports. The rest of other tools are specialized for the students and have guideline and error detection supports, except for the minimUML, QuickUML and UMLet. From the table also we can see that just the UML/Analyzer provide support for the project consistencies.

2.2 COMMON MISTAKES

As class and sequence diagrams are two of the most important UML diagrams which are modeled the classes, objects relationships and sequence of massage passing. Also, the main function of this tool is to assess the class and sequence diagrams that have been designated by the end-users by detecting their faults and errors in order to allow them to produce a better class and sequence diagrams. By studying the guidelines of class and sequence diagrams we are able to find out and classify the most important common mistakes which end-users may have done during their diagraming. Through reading the various online journal articles, the guidelines of the class and sequence diagrams that can be divided into five

parts for class diagram including class, attribute, association, generalization and the general guideline. And two part consist of lifeline and message for sequence diagram. Following is the guidelines of the class and sequence diagrams.

2.2.1 CLASS DIAGRAM GUIDELINE

2.2.1.1 CLASS NAME RULES

1. The class name should be written as nouns, singular and start with capital letters.

2. The class name should not Using reserved words in programming languages.

3. The class name should not Using space character.

4. The class name also should use a common terminology or easy to understand.

5. We should avoid representing interface element or the architectural element in the class diagram.

6. We also should avoid create two or more different classes name that indicate the same thing. (Should considered create only one class).

7. We also should avoid draw a class with only two compartments.

2.2.1.2 ATTRIBUTES RULE

1. We should check whether the attributes of the class represent the data that each instance must have. (Example: String, Boolean).

2. The attributes name should not Using reserved words in programming languages.

3. The attributes name should not Using space character.

4. Make sure to use the associations if there are plural attributes for a class.

5. List the attributes is listed in decreasing visibility.

2.2.1.3 ASSOSIATIONS RULES

1. Ensure that there is multiplicity at both ends of the association.

2. Avoid 1-to-1 association because usually this kind of association is incorrect.

3. Avoid creating an association that should be the operation.

4. Make sure that the role name that used for an association is a meaningful terminology and starts with lower case capital.

5. Avoid creating a class with no associations to any other classes.

2.2.1.4 GENERALIZATIONS RULES

1. Avoid multiplicity, multiple inheritances and having several subclasses that would not be meaningfully different.

2. We should check for the “is-a” relationship between subclasses and the superclass.

3. Also, ensure that the ( ) is pointed towards the superclass, not towards the subclass.

4. Make sure that the subclass is designated below the superclass.

5. Avoid Classes with multiple super classes.

2.2.1.5 GENERAL GUIDELINE

1. Make sure that the class diagram is designated horizontally (from left to right).

2.2.2 SEQUENCE DIAGRAM DUIDELINE

2.2.2.1 LIFELINE RULES

1. the lifeline class should have a name.

2. the lifeline name should be singular and start with capital letters.

3. lifeline should have visibility.

2.2.2.2 MESSAGE RULES

1. message should have description.

2.message better to have a return value.

3. message should have visibility.

2.3 ERROR DETECTION TECHNIQUES

It is important to study on the several error detection techniques that can be used and suitable for (CSeqD) tool. Some of the techniques that can be used are rule-based approach, knowledge-based approach and pattern-matching approach (Ali N. M, Hosking J., and Grundy J., 2013).

2.3.1 RULE-BASED APPROACH

According to the study of (M. J. Pazzani, and C. A. Brunk, 1991) Rule-based approach is an technique that includes a condition and action, where it is usually use IF-THEN statement. The IF statement in this rule is known as a condition, which it tests the truth value of a set of fact. if the condition is true, then the THEN statement that known as the action will perform. The action (THEN) statement can consist of any argumentations, suggestions, messages, explanations or example of problems. Rule in a rule-based approach are called as production rule. This approach is easy to use and to understand for implementation.

2.3.2 KNOWLEDGE-BASED APPROACH

The other approach which can be used for detecting errors is the knowledge-based approach (A. O. Elfaki, S.Amnuaisuk, and C. Kuan Ho, 2002) . This approach is consist of a set of rules and associations of compiled data which most usually taken form of the IF-THEN statements in the production rules. The knowledge base represents the most important components of a knowledge-based system. A knowledge base covers all true statements in a system, either the statements are predefined rules or facts derived during the execution of the system.

2.4 END-USER DEVEOPMENT

Another Issue which is important in this project is relate to end-user Development (EUD) definition. (M. Burnett, 2009) end-user Development (EUD) is one of the recent research topics in the computer science area. (EUD) is describing by activities and techniques which empower not professional developers to create and modify a software artifact. Users can be able to create their programs by end user programming approach. During these recent year developers and researchers make effort on empower users in order to create their own programs. End users use various programing environment such as web authoring tools, spreadsheets, dynamic web applications and educational simulation such as UML modeling. End users are able to create these programs by edit and write the formulas, connect objects in diagrams, drag and drop objects on to the workspaces.

However, there are some problem with end-user programing. Software which are created by end-users may contain errors. These errors impose significant unpleasant impacts on the quality of software. For instance, huge shortfall imposes to a school because of some problem in adding up budget in spreadsheet. One of the main problem in order to end user programing is that most of the programs which are created by end users apply with very low quality. And the reason why is that end users have different work limitations and motivations in comparison with professional

programmers. They do not know much about quality assurance issues, development processes and modeling diagrams.

In terms of UML modeling end-users might be are not familiar enough with the concepts of diagrams, relations and consistency or other important UML issues. So, Making mistakes and error will be inevitable during design UML diagrams. These kinds of error checking tools could help not only students but also end users in order to design UML diagrams correctly and teach them how to use modeling tools.

CHAPTER 3 – METHODOLOGY

3.1 INTRODUCTION

The software development lifecycle (SDLC) provide a framework to define tasks which should be performed at each phase of software development process. SDLC includes a detailed plan to describe how to develop and maintain of a system. Generally, the software lifecycle provide a methodology in order to improve the software quality and development process. Based on (PK. Raguuath, S. velmourougan, P. Davachelvan, S. Kayalvizhi, and R. Ravimohan, 2010) SDLC defines as a process that provides the structure and the steps of the application development from the early stage and studding a feasibility to the end of software development process which is deployment and maintenance. There are three model SDLC methodology which the development team should choose one of the according to the context of the project. For instance, the best model for web based which is used for developing a relational database could be waterfall model thus this model is not suitable for developing a web based applications.

Many methodologies can be used in order to development of a software project like Agile methodology, Spiral methodology and waterfall

methodology. But for (CSeqD) tool iterative and incremental development methodology is being used.

3.2 ITERATIVE AND INCREMENTAL DEVELOPMENT

The iterative methodology is introduced by Harlen Mills in 1970s and it has been further extended by Vic Basili by adding the incremental concept. Iterative means develop a system through repeated cycles while the incremental means develop a system in smaller portions at a time. The most important reason for introducing the Iterative and Incremental methodology is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental) that it will allow the software developers to take advantages of what was learned during development of earlier parts of the system.

During each iteration process, requirement analysis, design, repetitive modular implementation, test and integration and system test will be done. And this iterative cycle repeated again.

Based on (PK. Raguuath, S. velmourougan, P. Davachelvan, S. Kayalvizhi, and R. Ravimohan, 2010) this methodology is usually chosen for developing project because, this methodology current iteration can use the feedback from preciouses iterations. Moreover, the incremental

implementation can monitor any changes and all issues to be isolated and resolved in order to reduce the risks. Also this methodology helps to provide faster results and offer greater flexibility in comparison with the other methodologies.

3.3 PROJECT PHASES

This project includes 5 main phase as following:

- ✓ **Phase1:**

Review on UML diagrams specially class and sequence diagrams and analyze their characteristics. Also, review on existing UML error detection tools and their features in order to gain enough knowledge in the field of study.

- ✓ **Phase2:**

Identify existing technique and different approaches in UML diagrams defect detection and finding common mistake which end users done during UML modeling process. Study on other existing tools and detecting their strengths and weaknesses in order to be more familiar with previous works and the area of study.

- ✓ **Phase3:**

Identify the requirements based on the previous reviews and make decision about detection techniques to be applied for this automated error checking tools of UML diagrams.

- ✓ **Phase4:**

Design and develop the prototype of the automated error checking tools of UML diagrams specifically on class and sequence diagrams.

- ✓ **Phase5:**

Evaluate and validate the automated error checking tool for UML class and sequence diagrams. Also, finish the documentation of the results.

For the requirements gathering phase, the information that related to the project were collected through reading articles, working with similar tools, working with UCDAT tool which proposed by (N.N. Johazmi, 2016) to find out the strength and weakness of the tool in order to get the functionalities that can fulfills the objectives of the project.

The information that had been gathered throughout the reading activity was being discussed in the Chapter 2. So, in this chapter it will discuss about the UML Class Diagram Assessment Tool (UCDAT) tool activities and improvements that will done. UML Class Diagram Assessment Tool (UCDAT) is an assessment tool which is written with Java programming language in Netbeans IDE platform . This tool detect the UML class diagram name, attributes, operations and relationship errors as shown in Table 2.

Table 2 : UCDAT tool detected errors

	Element	**Detected Error**
1	Class name	Class name Start with upper case Class name ends with singular word
2	Attribute	Check attributes type
3	Operation	Check attributes type
4	Relationship	Association name start with lower case

Based on Table 2 and common errors done by end-users during UML diagraming which is discussed in chapter2, there are more errors that need to be detected by this tools. This project is focused on the error checking and detection for UML Class and Sequence Diagrams.

3.4 DETECTING ERRORS BY (CSeqD)

3.4.1 CLASS DIAGRAM'S ERROR DETECTING BY (CSeqD)

The class diagram errors include class name, class attributes, class operations and class relationships errors detected by Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) is introduced by Table 3 and 4 as follow:

Table 3 : Class name, attributes, and operations detected by (CSeqD).

Class Name	**Class Attribute**	**Class Operation**
Class should have a name.	Check attributes visibility.	Check operations visibility.
Class name should not include space letter.	Attributes should have a Data Type.	Operations should have a Data Type.
Class name should not include 'And', 'Or' words.	Attributes should have a name.	Operations should have a name.
ass name should not include programming reserved words.	Attributes name should not include space letter.	Operations name should not include space letter.
	Attributes name should not include 'And', 'Or' words.	Operations name should not include 'And', 'Or' words.
	Attributes name should not include programming reserved words.	Operations name should not include programming reserved words.

Table 4 : Class relationship errors detected by (CSeqD).

Association	**Generalization**	**Dependency**
Associations should have a name.	Generalizations should have a name.	Dependencies should have a name.
Association name should not include space letter.	Generalizations name should not include space letter.	Dependencies name should not include space letter.
Association name should not include ‘And’, ‘Or’ words.	Generalizations name should not include ‘And’, ‘Or’ words.	Dependencies name should not include ‘And’, ‘Or’ words.
Association name should not include programming reserved words.	Generalizations name should not include programming reserved words.	Dependencies name should not include programming reserved words.
Check Association multiplicity.		

3.4.2 SEQUENCE DIAGRAM’S ERROR DETECTING BY (CSeqD)

The sequence diagram errors include sequence lifeline and sequence message errors detected by Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) is introduced by Table 5 follow:

Table 5 : Sequence diagram lifeline and message errors detected by (CSeqD).

LIFELINE	MESSAGE
Lifeline class should have a name.	Message should have description.
Lifeline name should be singular and start with capital letters.	Message better to have a return value.
Lifeline should have visibility.	Message should have visibility.
Lifeline class name should not include ‘And’, ‘Or’ words.	Message description should start with lower case.
Lifeline class name should not include programming reserved words.	Message better to have a parameter.
Lifeline class name should not include space letter.	

After completing the information gathering activity, the next phase took place was the analysis and design phase. This phase is being discussed in the next chapter which is Chapter 4 followed by Chapter 5 where it is discussing on the implementation and testing that are conducted in this project.

CHAPTER 4 - SYSTEM DESIGN AND IMPLEMENTATION

4.1 SYSTEM DESIGN

In iterative and incremental software development methodology one of the most important phase is system design. In this phase system architecture and user interface will design and user interface will be shown in the low-fidelity interface form. Low-fidelity interface form means interface sketching that were designed based on the functionality and characteristics of the project.

In order to find out the most significant requirements and functionalities for the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD), all needed information gathered from literature reviews study, analyze and survey. The requirements and functionalities of the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) graphically shown by UML use case diagram as following in FIGURE 1.

FIGURE 1: Use-case diagram that represented all of the functionalities of (CSeqD).

Based on the above use-case diagram, it can be seen that there only a single actor which interact with the tool. This actor will be the end-user that interact with five functionalities which provide the (CSeq). The first functionality is related to the uploading the class or sequence diagram XML

file in (.simp) format. Only the .simp format is acceptable for (CSeqD) because the tool that is used in order to design the class and sequence diagram is Software Ideas Modeler modeling tool and the exporting XML file from this tool is in (.simp) format. For this use-case, it is will have a “include” relationship with other six use-cases, which are check the class’s name, check class’s attributes, class’s operations, class’s relationship, sequence’s lifeline and sequence’s message. “include” relationship was used in the use-case diagram because all the requirements of UML call and sequence diagrams are related to these six others functionalities. Moreover, end-users are able to choose either save the detecting error list in (.txt) format as summary page or print the hardcopy of error detection list and send the summary page directly to the printer.

4.1.1 LOW-FIDELITY PROTOTYPES

The next step after finishing the requirements and functionalities gathering of the tool was designing and drawing the low-fidelity interface for the tool. Some sketches of the low-fidelity interface of Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) are shown as follow:

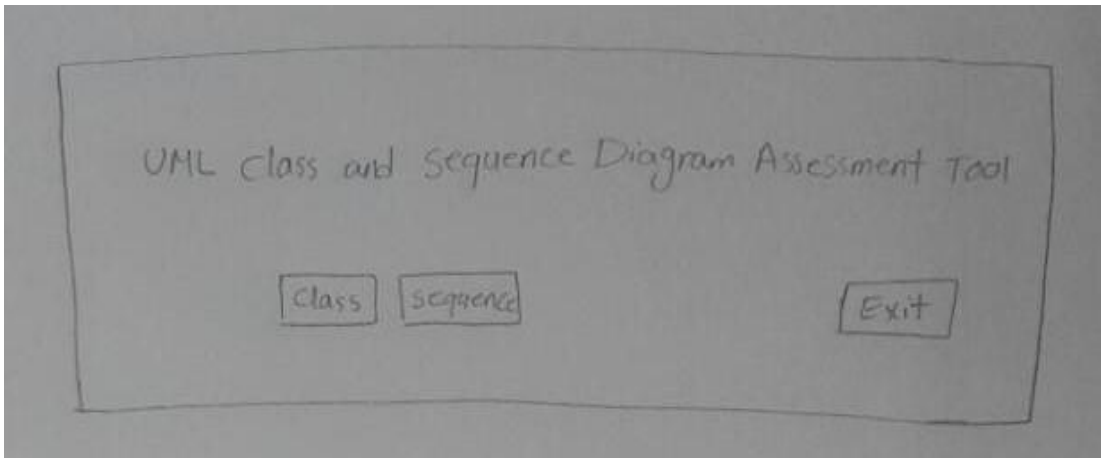


FIGURE 2: The sketch for the “Home” interface.

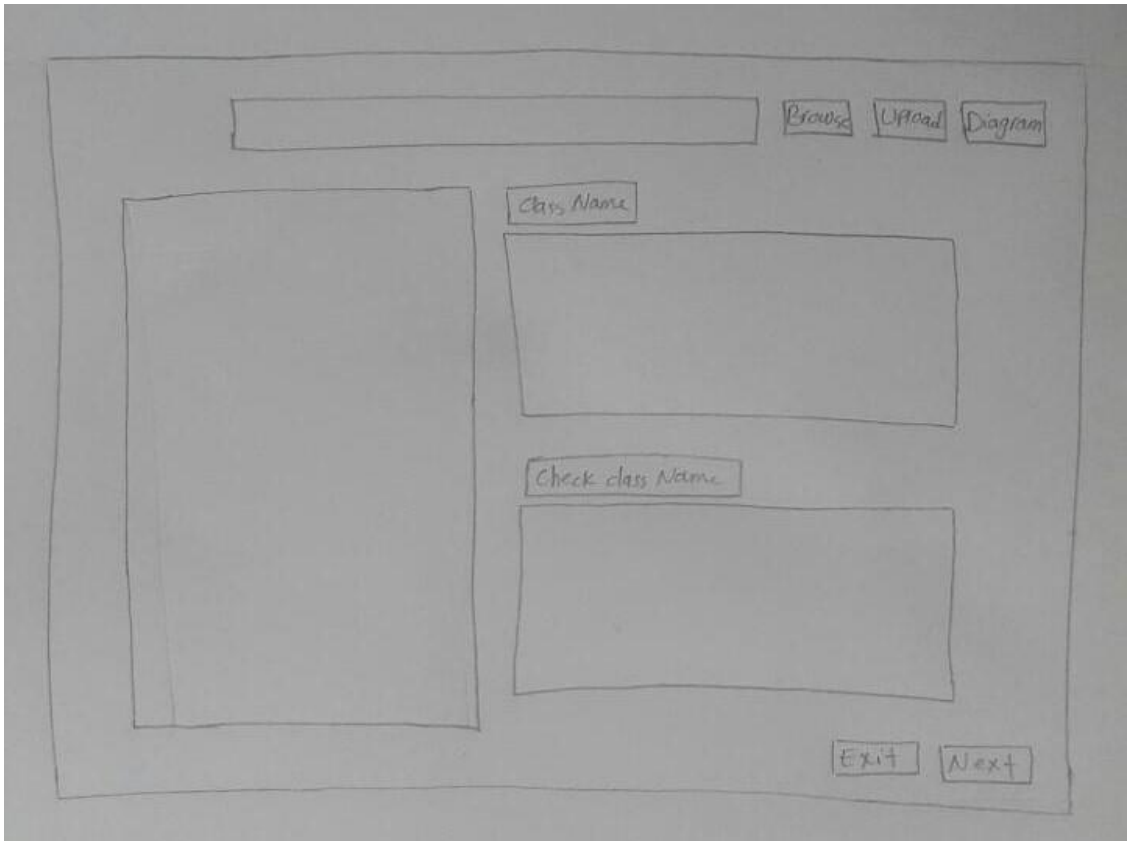


FIGURE 3: The sketch for “Class Diagram Upload” interface.

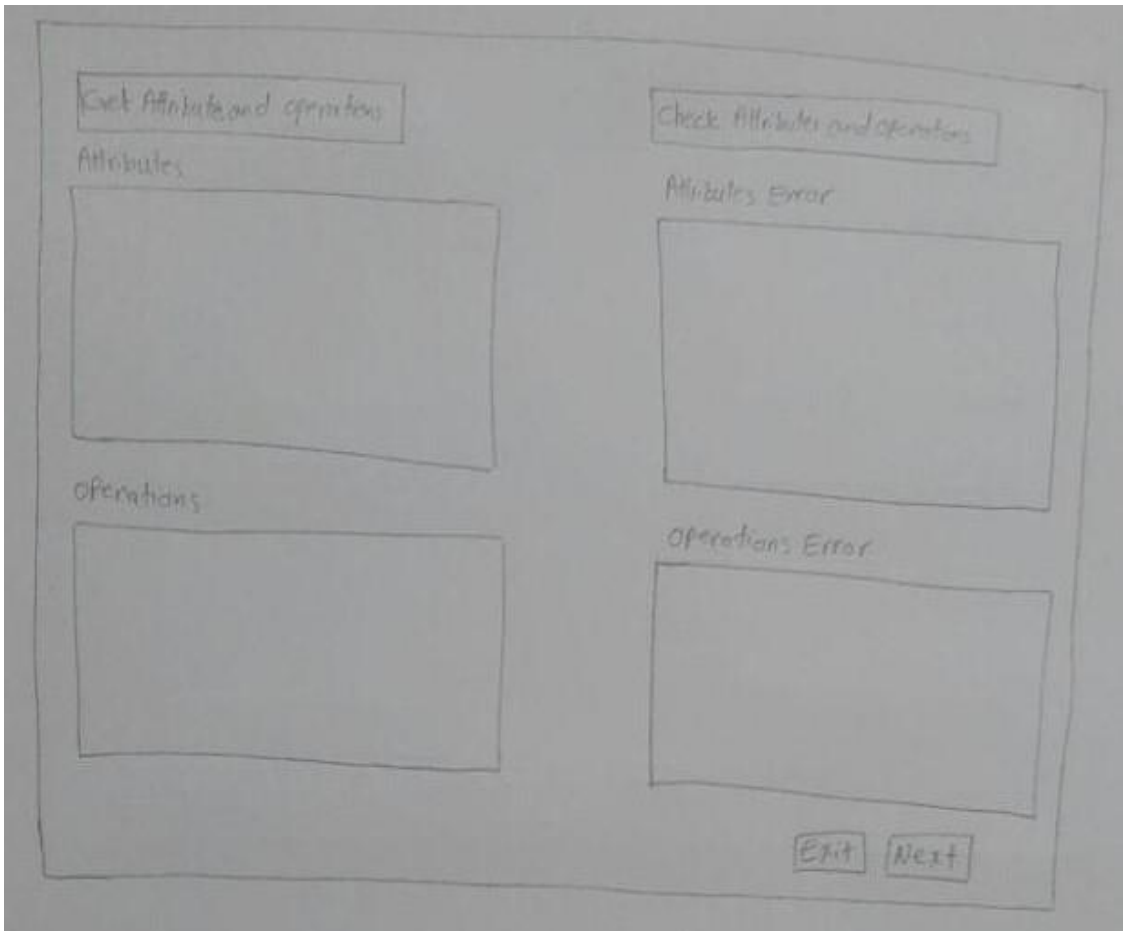


FIGURE 4: The sketch for “check Class Attribute Operation” interface.

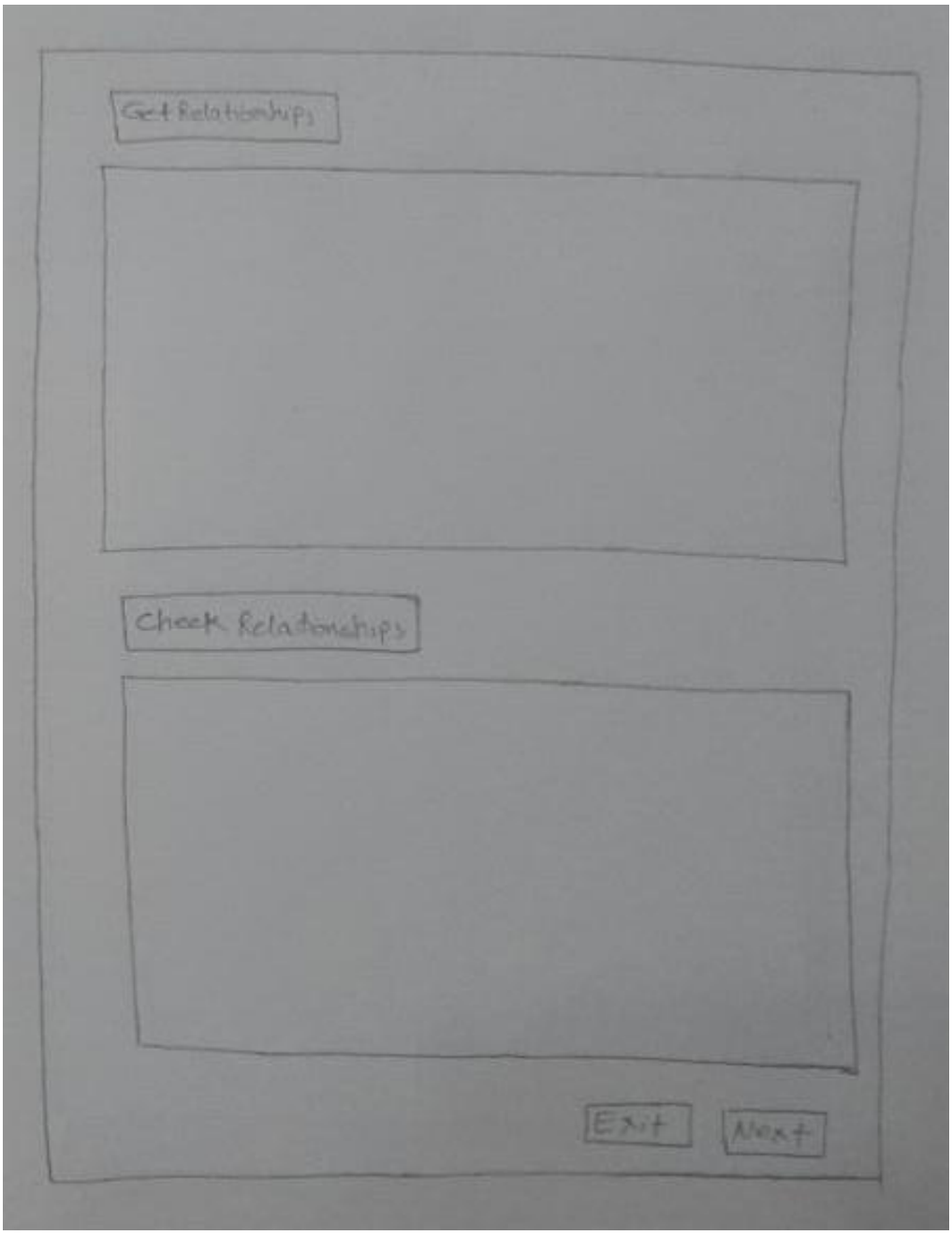


FIGURE 5: The interface for “check Class Relationship” interface.

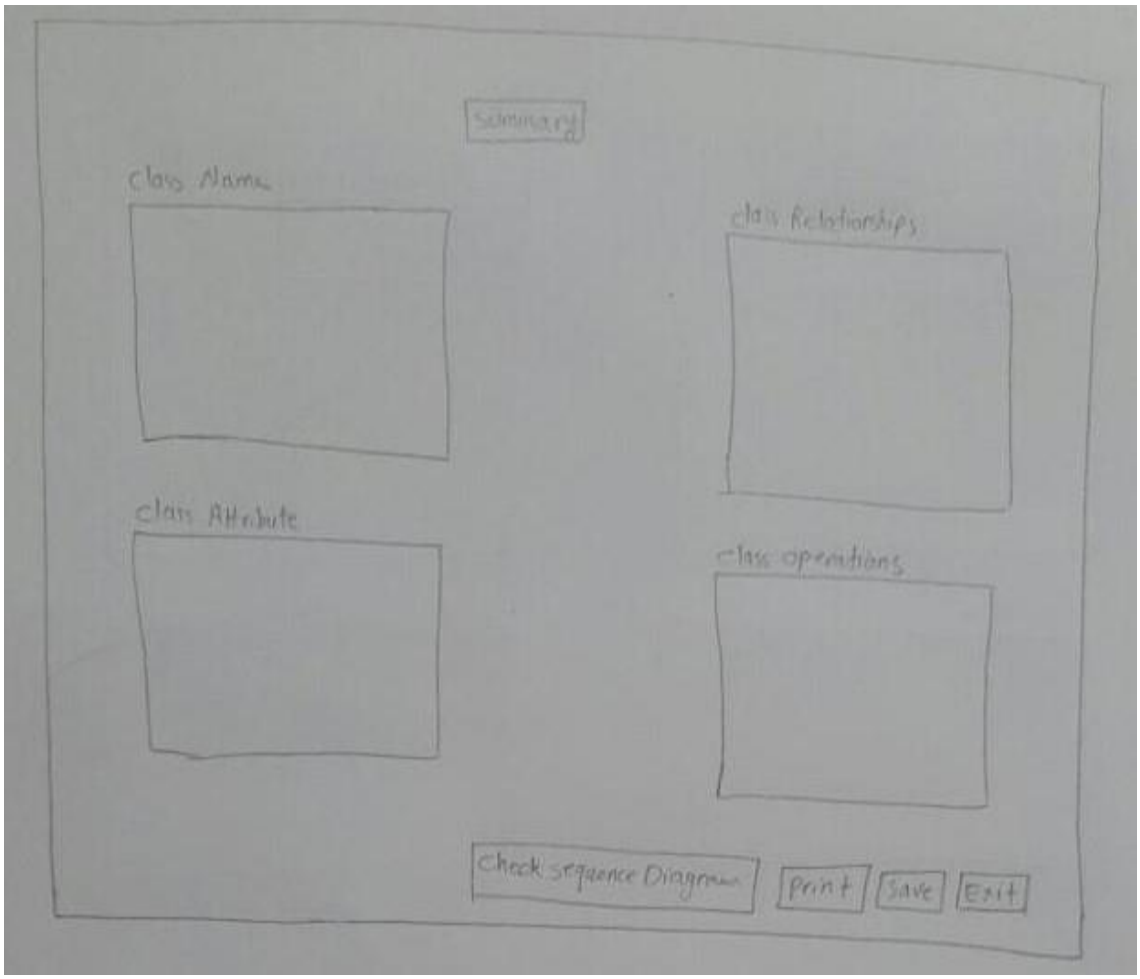


FIGURE 6: The sketch for “Class Diagram Summary” interface.

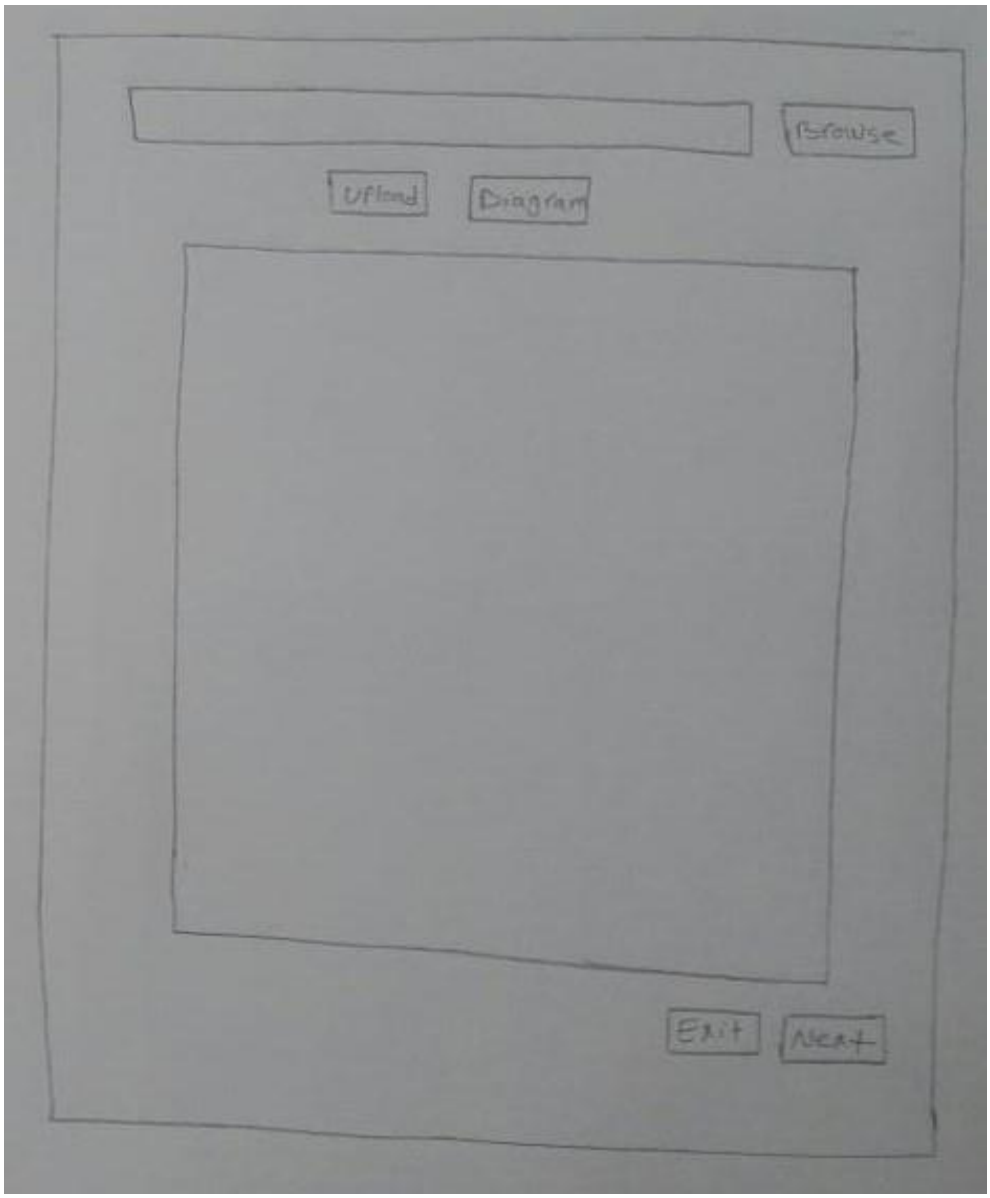


FIGURE 7: The sketch for “Sequence Diagram Upload” interface.

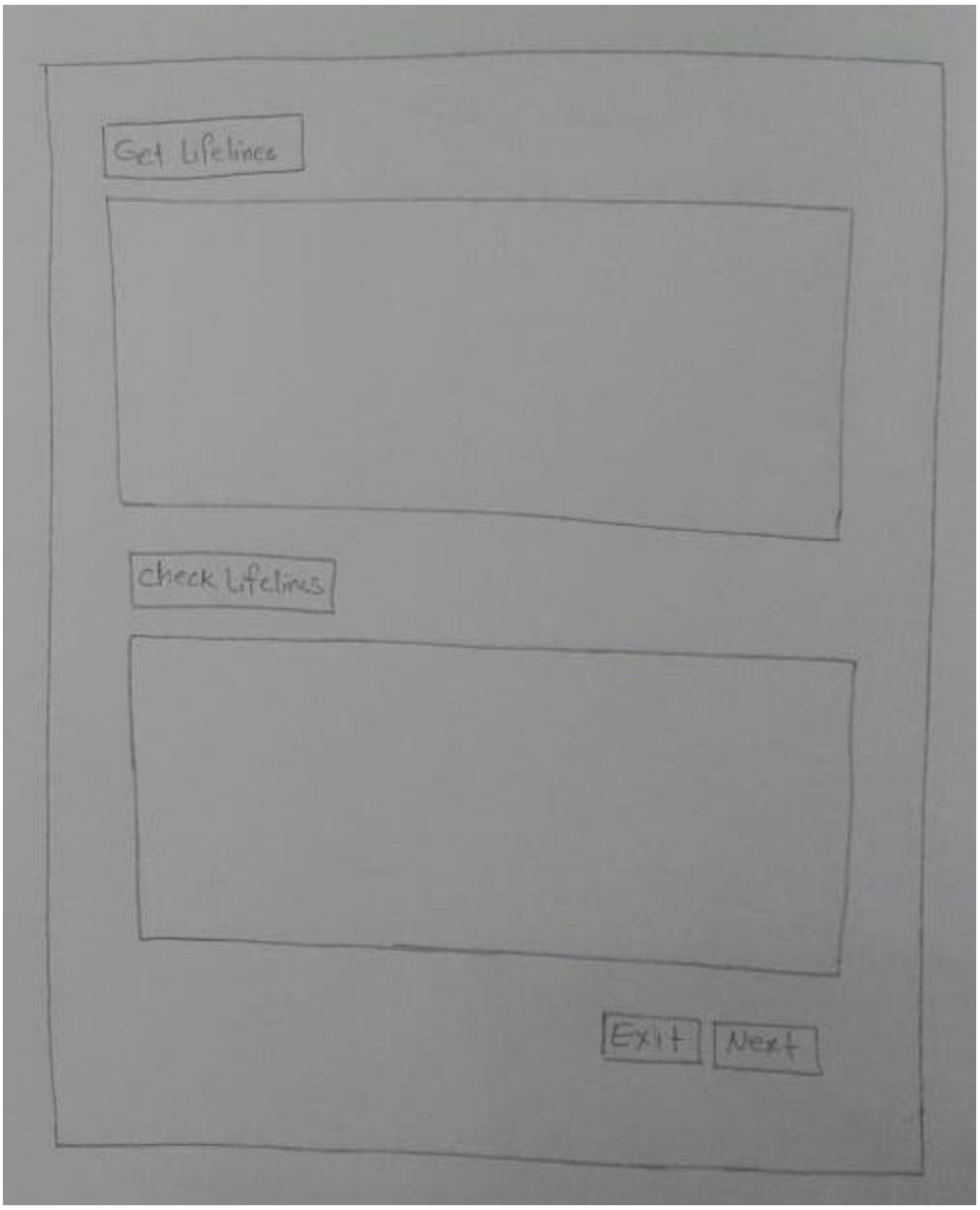


FIGURE 8: The sketch for “Sequence Diagram Lifeline” interface.

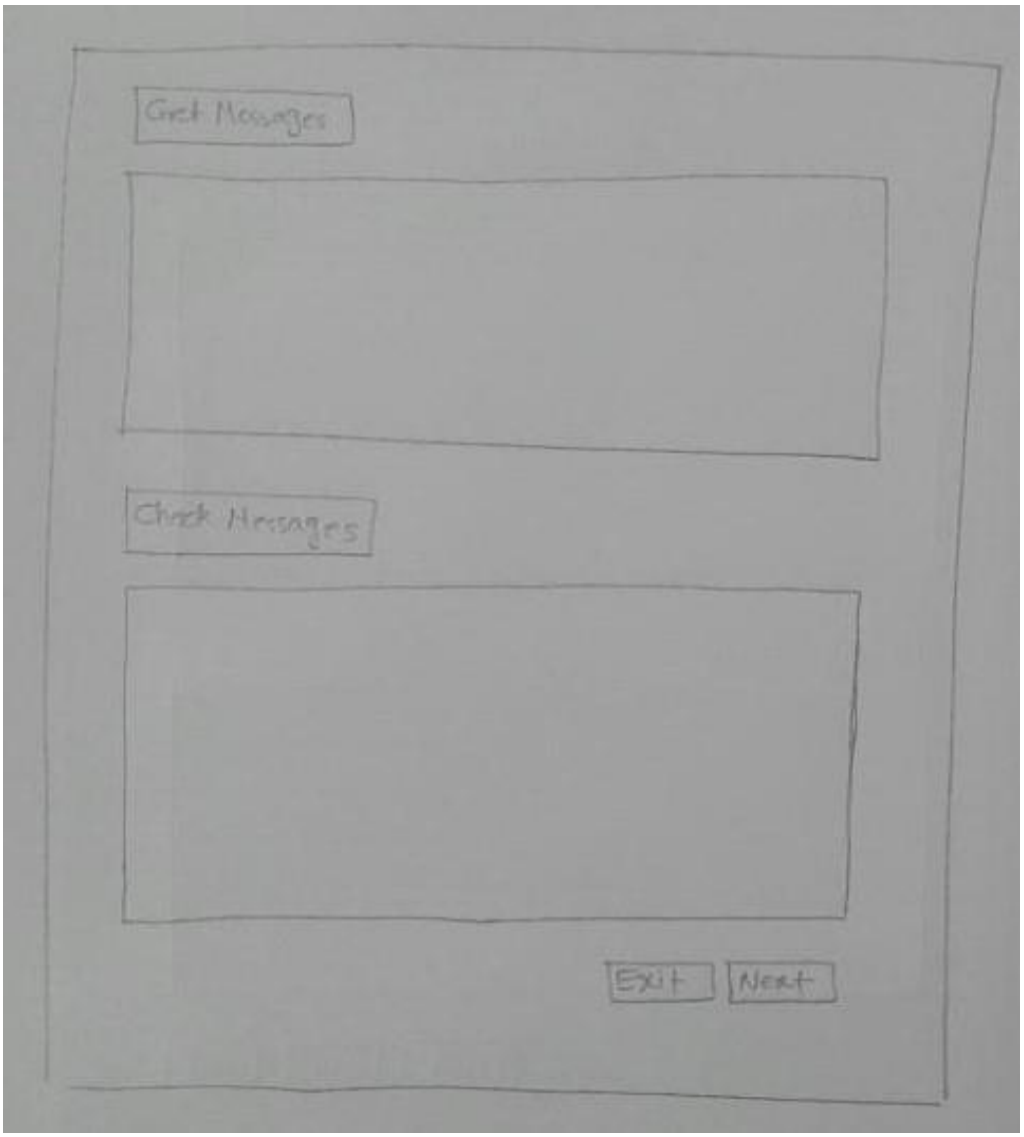


FIGURE 9: The sketch for “Sequence Diagram Message” interface.

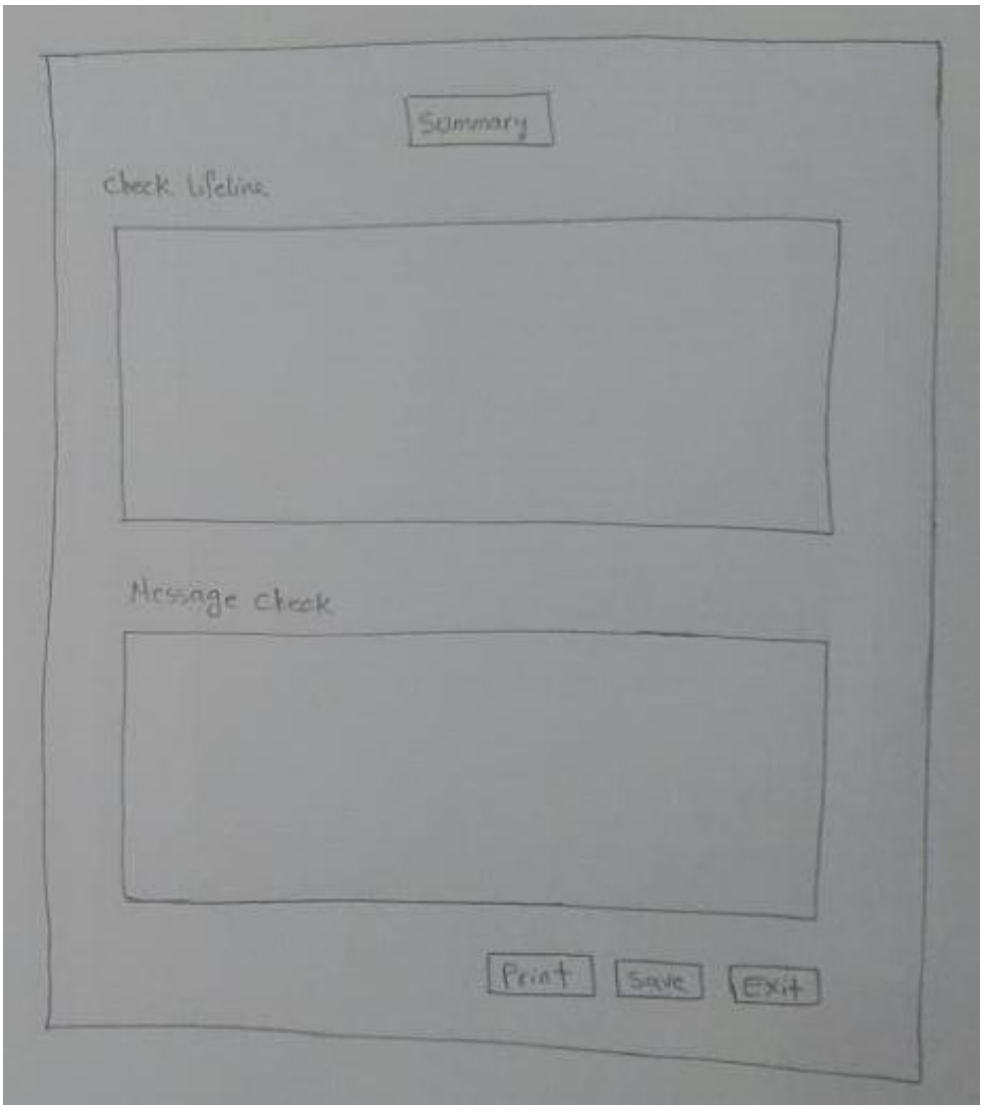


FIGURE 10: The sketch for “Sequence Diagram Summary” interface.

FIGURE 2 to FIGURE 10 represent the main interfaces that should be implemented for the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD). End-user will face to the FIGURE 2 interface as long as start to use the tool, this page is the Home page which end-users are able to choose whether they want to assess class diagram or

sequence diagram. From the FIGURE 3 to end-users can check the correctness of their class diagram functionality. By these pages it is possible for them to check the class diagram's name, attributes, operations and relationships correctness. As for FIGURE 6 end-users can see the summary of all the class diagram errors detected and they can save or print the results. From this page they can choose to exit the tool or continue for checking a sequence diagram. From FIGURE 7 to FIGURE 10 assessment of the sequence diagram is available for end-users. By FIGURE 7 they can upload the (.simp) format of their diagram and in 3 next pages they are able to check the correctness of their sequence diagram lifeline and message. In FIGURE 10 sequence diagram summary page will display and it is possible for end-users to visit the sequence diagram errors and directly save or print the summary of their sequence diagram error detection. But, this sketches may not be exactly implemented as it is shown. During the implementation phase due to technical issues such as difficulties in programming where iteration cycle needs to take place. These sketches were being used as the references in designing the interfaces during the implementation phase.

4.1.2 ARCHITECTURAL DIAGRAMS

Due to achieve the better understanding of the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) architectural design, I decided to continue with a few UML diagrams. In this phase use-case diagram, sequence diagram, activity diagram and also the deployment diagram will discuss. Use-case diagram had been discussed before in the previous chapter.

In this part I am going to introduce the overview of the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) in general. As mentioned before, this tool is designed to assess the diagrams which is created by modeling tool that known as Software Ideas Modeler. The interaction between Software Ideas Modeler modeling tool and Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) is provided in FIGURE 11 as follow:

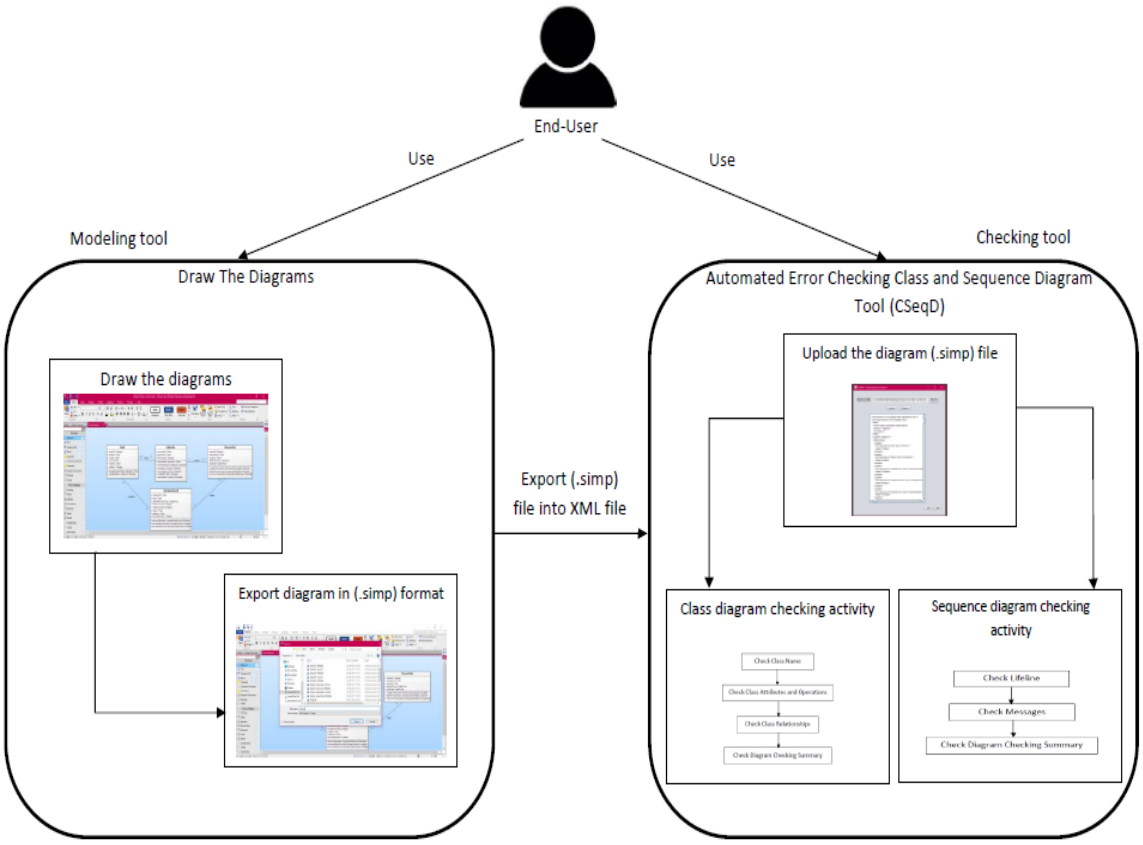


Figure 11: General overview of the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD)

Now it is the time to proceed in discussing about the diagrams that can shows the architecture of the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) in details.

4.1.2.1 SEQUENCE DIAGRAMS

Sequence diagram usually used to represent the interactions between system objects in a sequential order. Sequential diagram contributes user to achieve a better understanding about how the system works. For every functionality of the system there should be a sequence diagram. In order to have understanding of sequence diagrams, Entity-Control-Boundary (ECB) pattern approach is used for modeling the diagrams. The ECB pattern approach is the simplification of the Model-View-Controller pattern. There are three elements for this pattern which are entity, control and boundary. Entity is the object which represents the system data. The object that act as the interface between the system called boundary and the object that is act as the connector within he entities and boundaries is the control. For the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) would have around 9 sequences diagrams that will show the how does each of the functionalities works on the system.

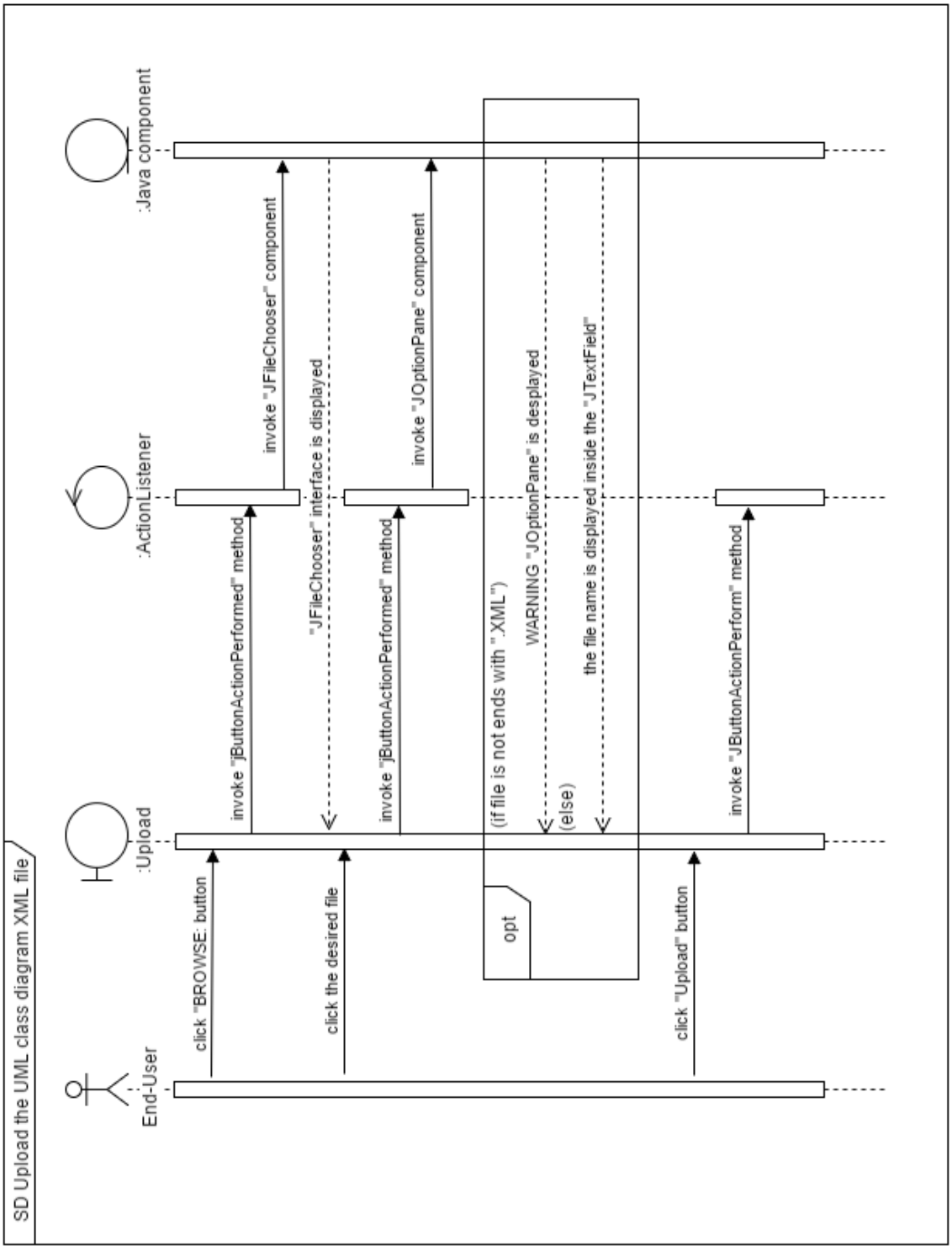


FIGURE 12: Sequence diagram for the upload the UML class diagram XML file

As for FIGURE 12, it represents the uploading the UML class diagram XML file functionality of the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) sequence diagram. It can be seen that for this functionality, “upload” interface is a boundary. The “ActionListener” acts as the control and the “Java component” as the entity. The “ActionListener” is one of the Java methods that can be invoked through the Java interface component such as the JButton component. In this sequence diagram functionality , there are two times that the student is asked to click on the button in the “upload” interface where it is allowing the interaction between the boundary and control elements were took placed. After that, the interaction between control and entity elements were took placed once the “ActionListener” invoked the element from the “Java component” entity. The elements that were invoked from the “Java component” were JFileChooser and the JOptionPane. The JFileChooser is a Java component that was allowing the user to choose file from the tool through a simple interface while the JOptionPane is a Java component that allowing the implementation of the interface that consist of a certain message, such as the warning message.

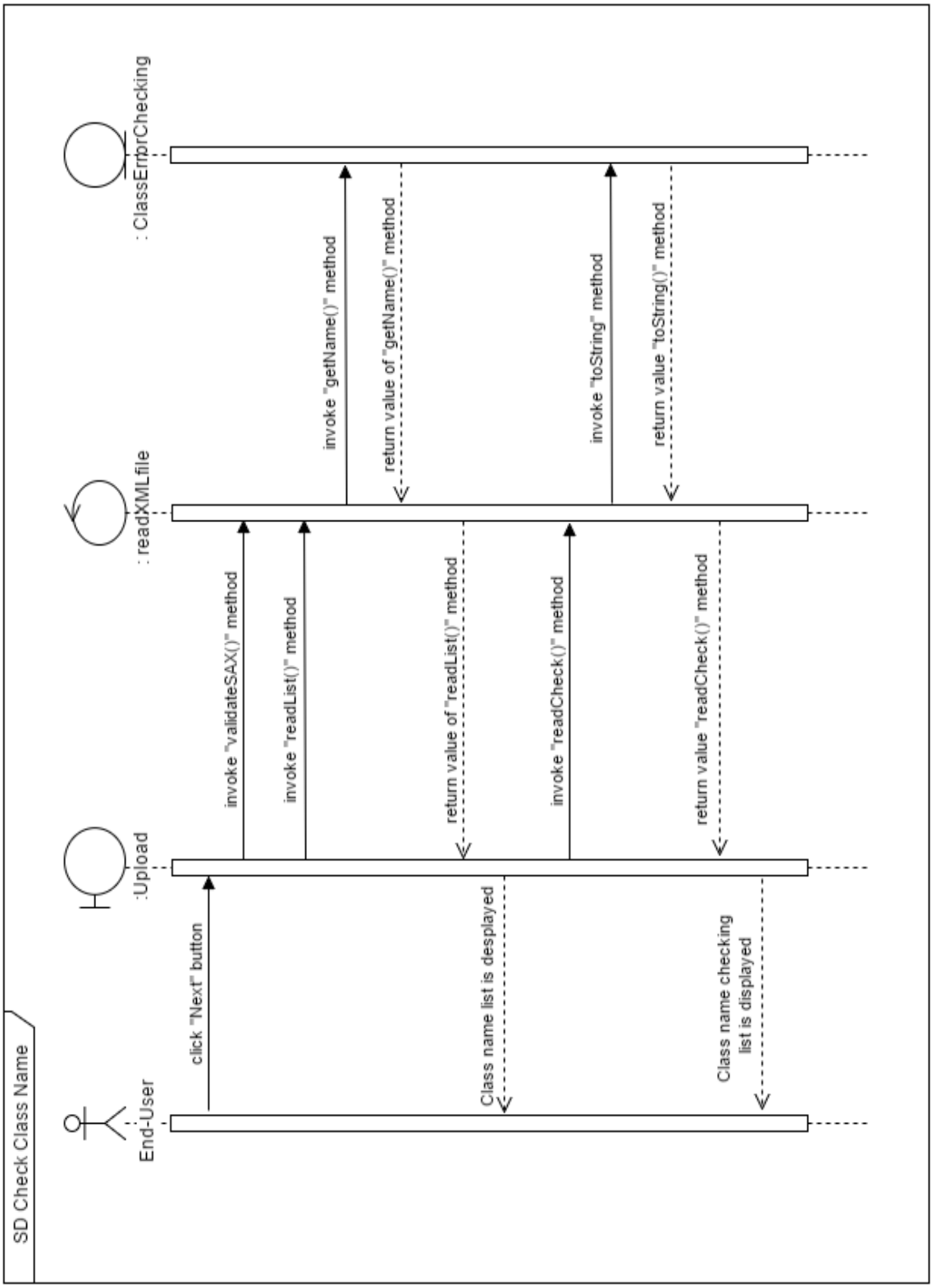


FIGURE 13: Sequence diagram for the check class's name.

FIGURE 13 represents the sequence diagram for the functionality of class name check. In this sequence diagram functionality one boundary element, one control and one entity element are used. Boundary element acts as the “upload” interface, control element which is “readXMLfile” and entity element which is “ClassErrorChecking”. The “readXMLfile” and “ClassErrorChechking” is the Java classes. For this functionality, the interactions that were occurred among the objects were involved the invoked the methods from the Java class. For example, for the first action of the student that is “click the GET CLASS NAME” button, the “upload” interface were invoking two methods inside the “readXMLfile” class concurrently and from this invoked activities, the “readXMLfile” class was invoked a method from the “ClassErrorChecking” class. After that, the “ClassErrorChecking” class was sending back the value (information) needed in the invocation activity to the “readXMLfile” class and so on. The processes were in the progress until the functionality is done.

The process of three other functionalities, “check the class’s attributes”, “check the class’s operations” and “check the class’s relationship” are similar to the process of “check the class’s name” functionality. Whereas all the three of these functionalities were used various entity elements, control elements and also the boundary elements. the functionality of the “check the class’s attributes” that is shown in FIGURE 14, this functionality involving one boundary element which is the “checkAttribute” but it consist of

one control element, which is the “readAttributeXML” class. And an entity element which is contain the “AttributeErrorChecking” class. The functionality of the “check the class’s operations” which is shown in FIGURE 15, this functionality involving one boundary element which is the “checkOperation” but it consist of one control element, which is the “readOperationXML” class. And an entity element which is contain the “OperationErrorChecking” class. For the functionality of “check the class’s relationship”, it includes just one boundary element, which is the “checkRelationship” interface, one control element that is the “readRelationshipXML” class and one entity element which is the “RelationshipErrorChecking” class.

All of the processes for these functionalities were being shown in the FIGURE 14, FIGURE 15 and FIGURE 16.

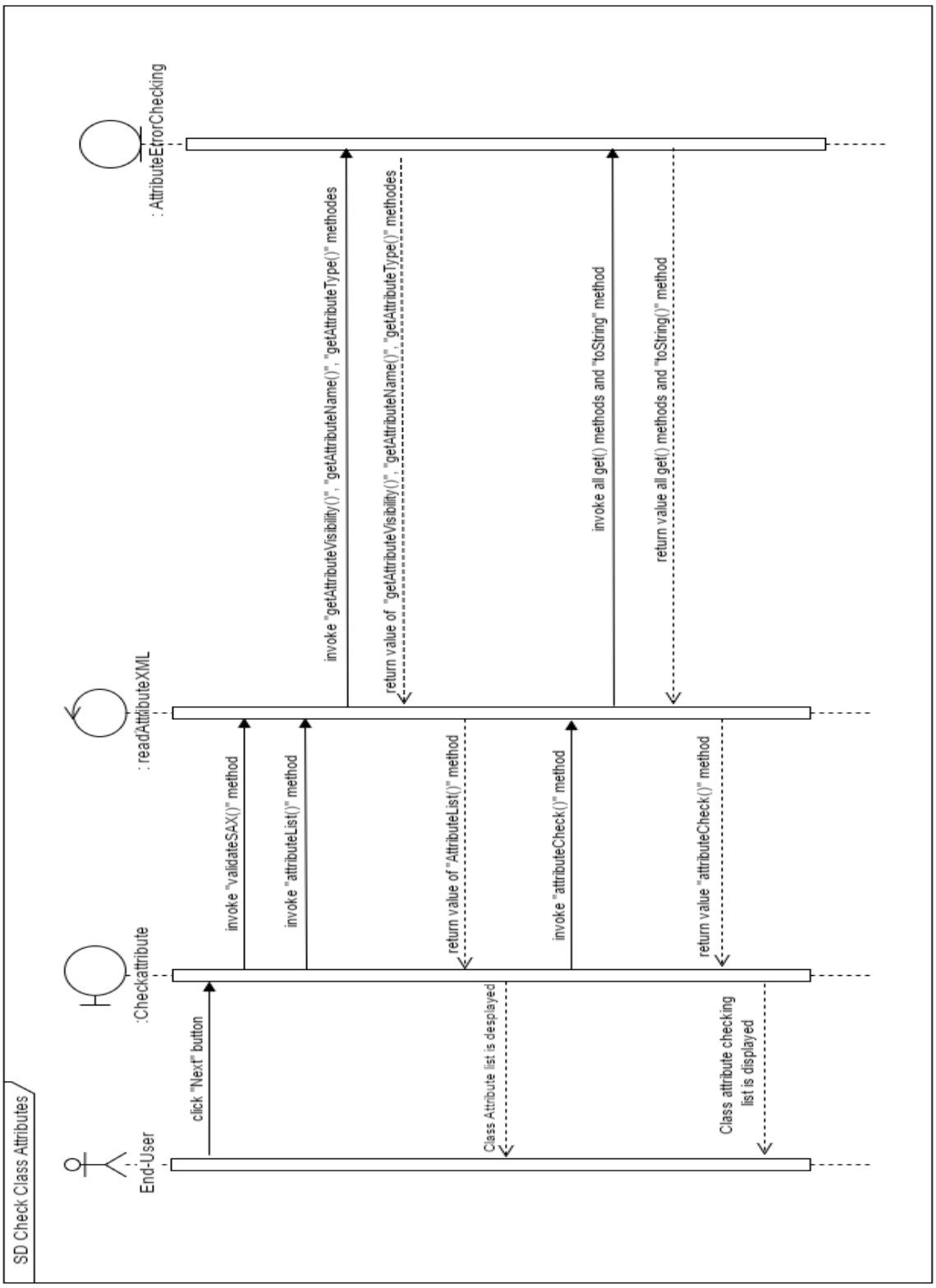


FIGURE 14: Sequence diagram for the check class's attributes.

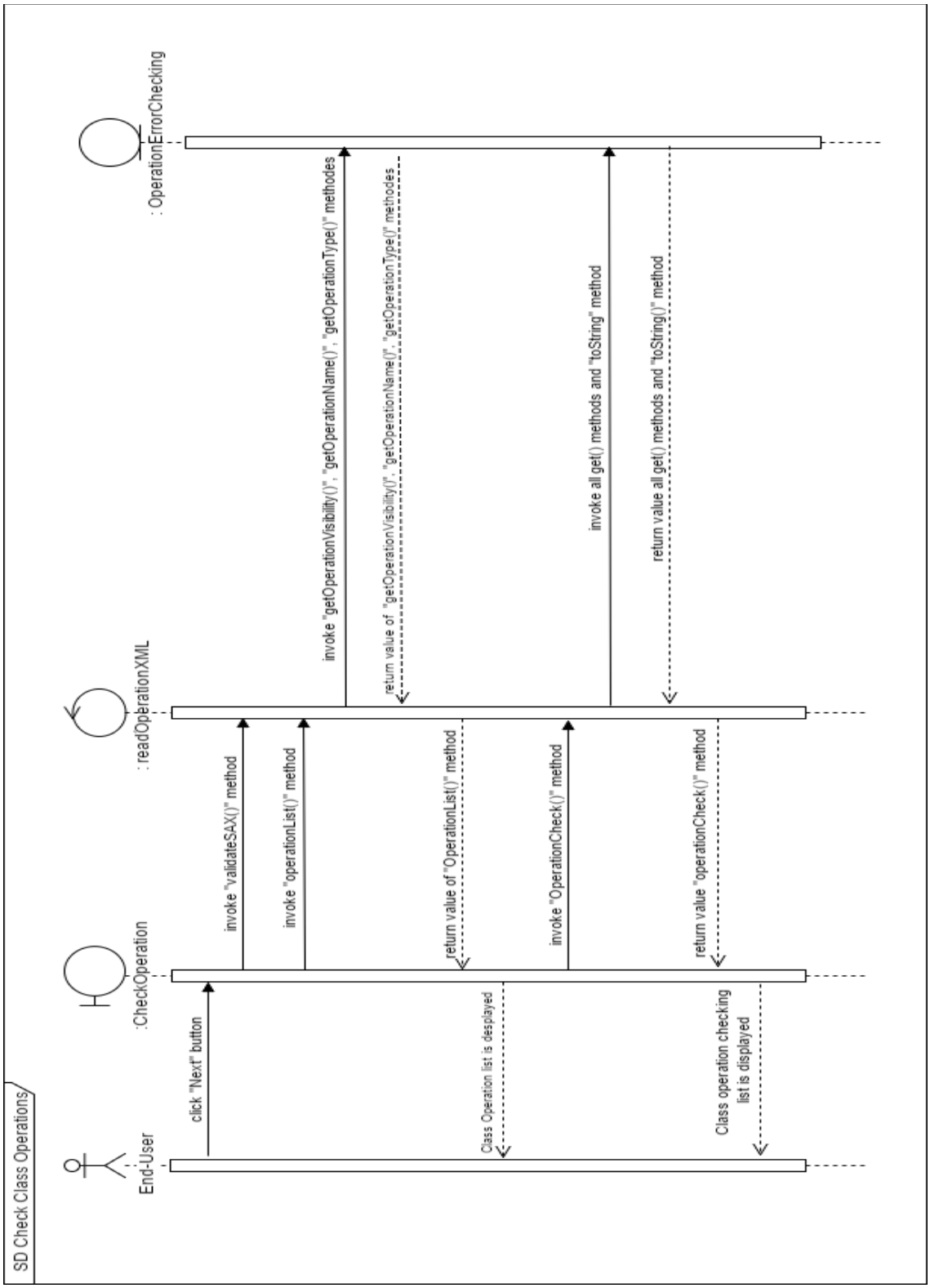


FIGURE 15: Sequence diagram for the check class's operations.

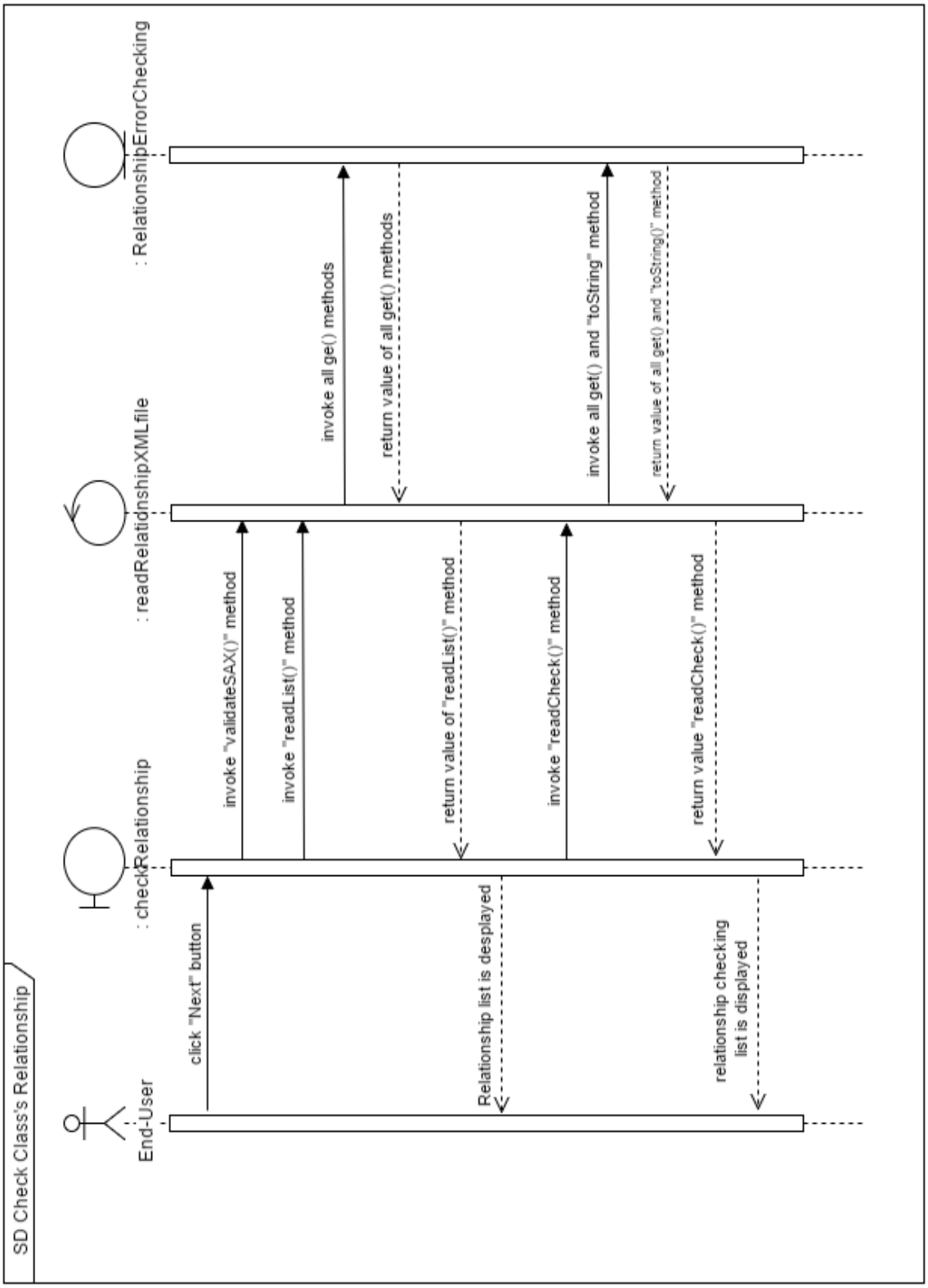


FIGURE 16: Sequence diagram for the check class's relationships.

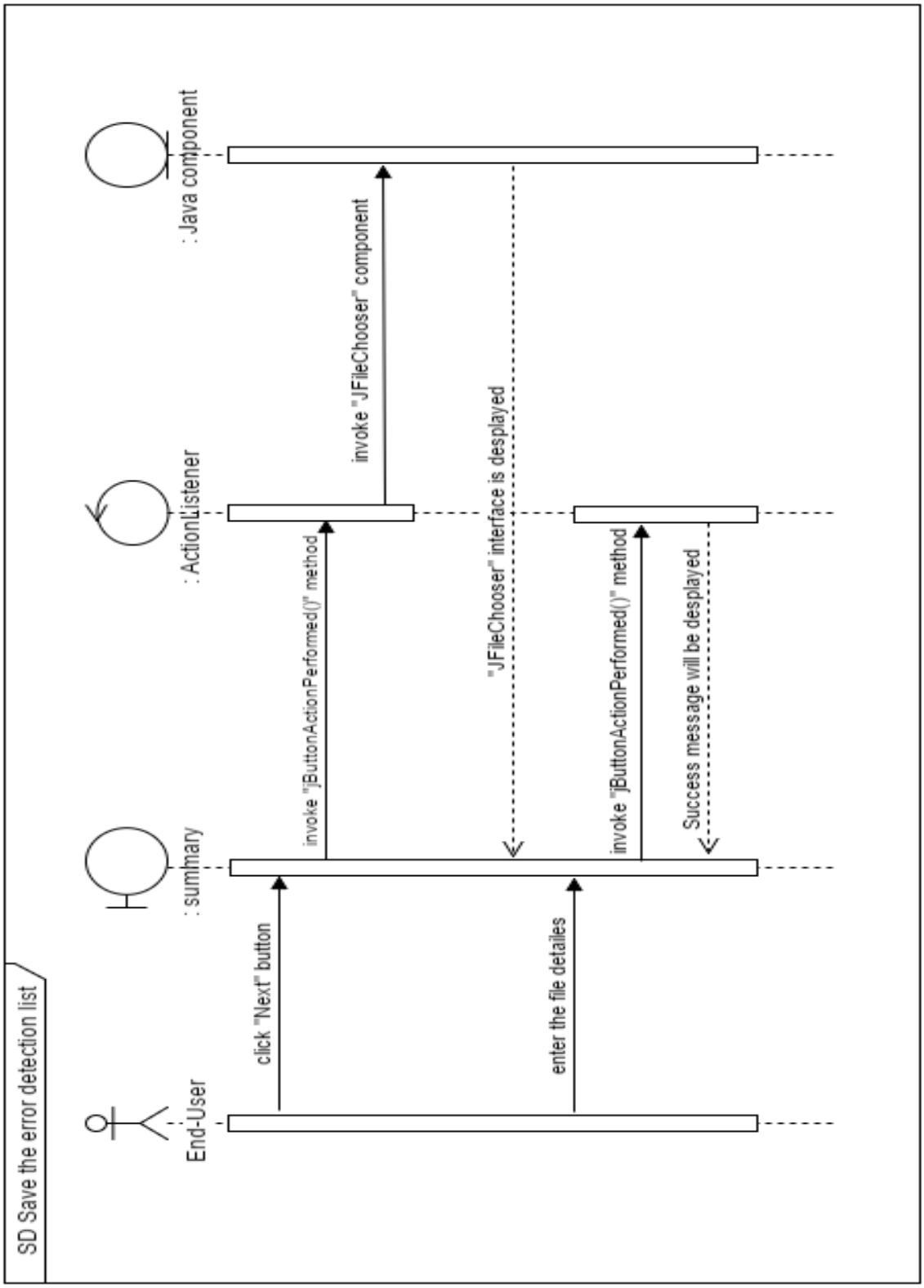


FIGURE 17: Sequence diagram for the save the error detection lists.

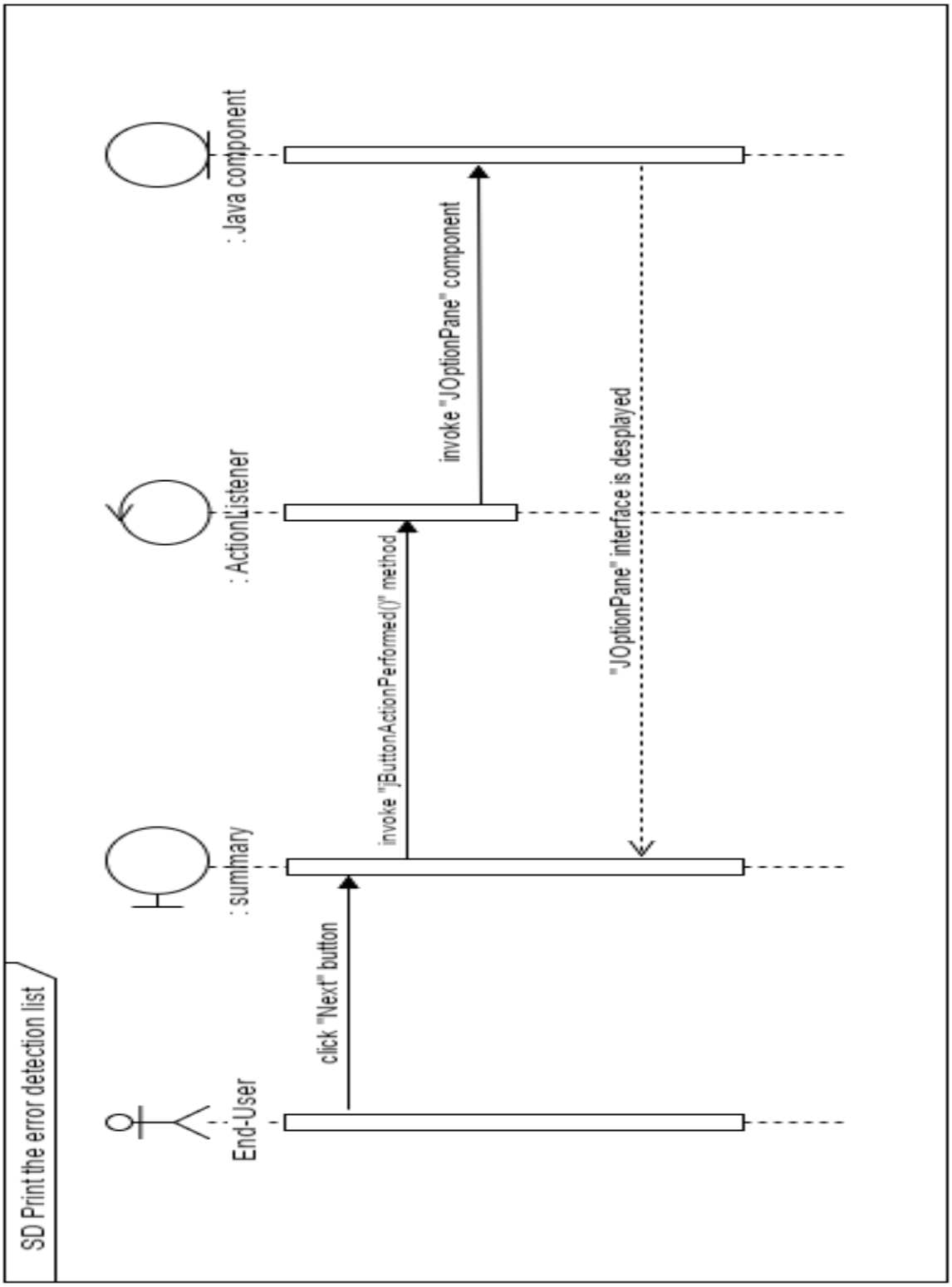


FIGURE 18: Sequence diagram for the print the error detection lists.

FIGURE 17 and FIGURE 18 are the sequence diagrams for the "save the error detection lists" and "print the error detection lists" functionalities. For these two functionalities, they were involving the same control element which is the "ActionListener" and the same entity element which is the "Java component". Both of these functionalities also are having the same boundary element which is the "summary" interface. All the processes of interactions that occurred for these two functionalities can referred to the FIGURE 17 and FIGURE 18.

Next phase of project of is related to the sequence diagram error checking. The Sequence diagram for the upload the UML sequence diagram XML file is similar to the Sequence diagram for the upload the UML class diagram XML file as shown in FIGURE 12. Two other functionality of sequence diagram error checking are "CheckLifeline" and "CheckMessage". The sequence diagram of these two functionalities are represented as FIGURE 19 and FIGURE 20.

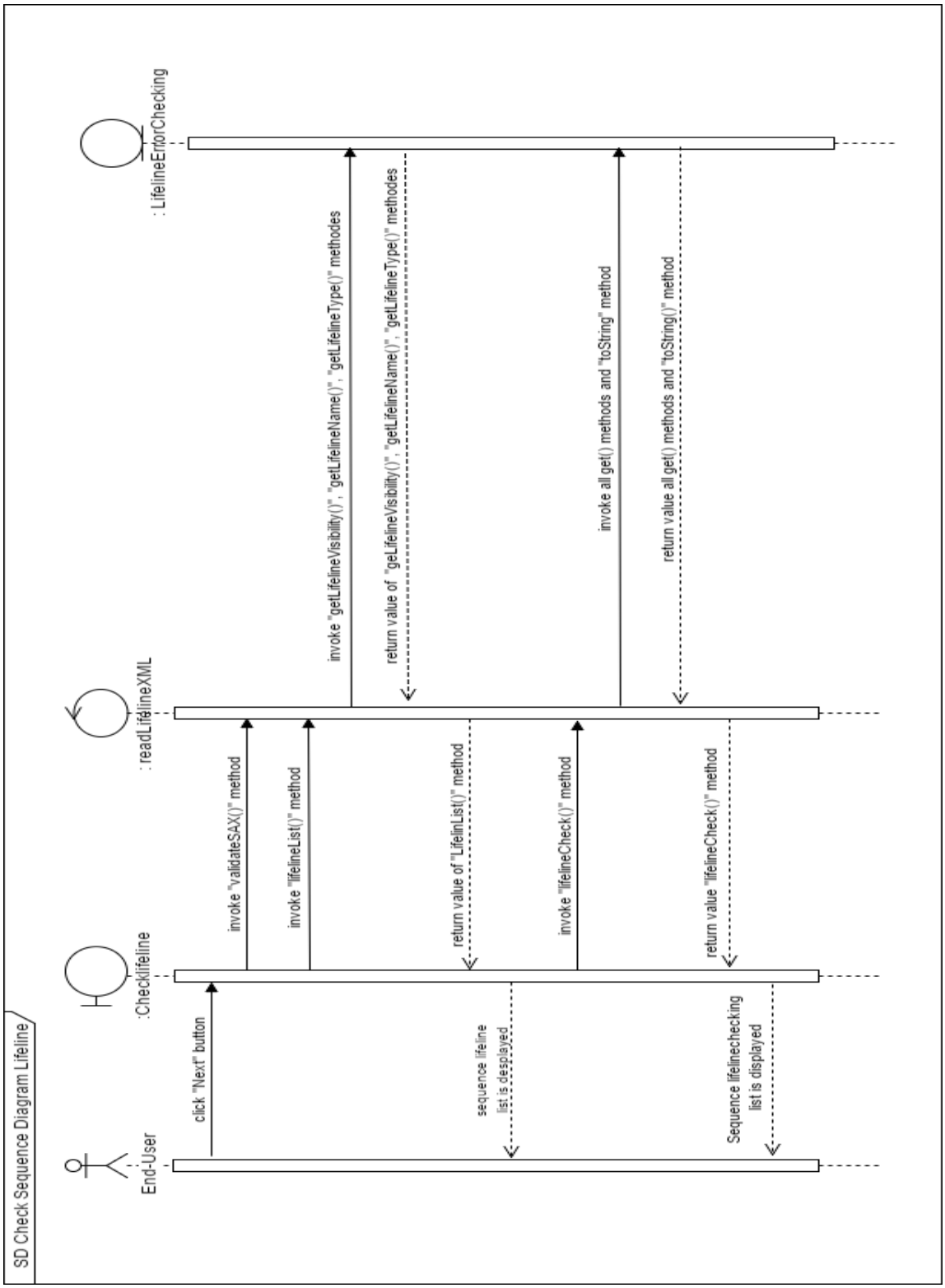


FIGURE **19**: Sequence diagram for the check sequence diagram lifeline.

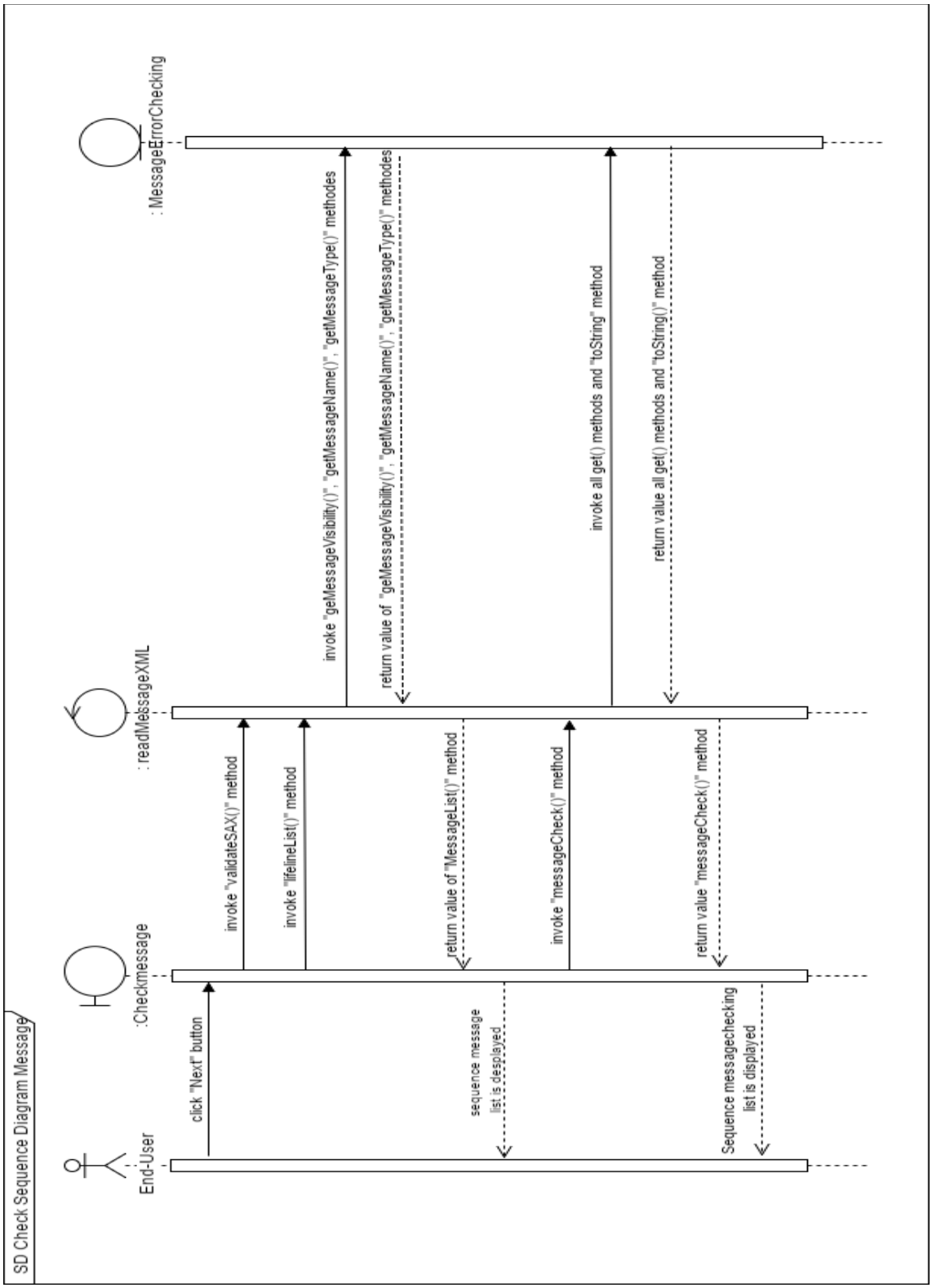


FIGURE 20: Sequence diagram for the check sequence diagram message.

The process of two other functionalities of sequence diagram error checking, “check the sequence lifeline” and “check the sequence message” are similar to each other. Whereas both of these functionalities were used various entity elements, control elements and also the boundary elements. the functionality of the “check the sequence lifeline” that is shown in FIGURE 19, this functionality involving one boundary element which is the “checklifeline” but it consist of one control element, which is the “readLifelineXML” class. And an entity element which is contain the “LifelineErrorChecking” class. The functionality of the “check the sequence message” is consist of one boundary element which is the “checkmessage” but it consist of one control element, which is the “readMessageXML” class. And an entity element which is contain the “MessageErrorChecking” class. Moreover, save and print functionality of sequence diagram error checking are similar to the class diagram error checking which are shown in FIGURE 17 and FIGURE 18.

4.1.2.2 ACTIVITY DIAGRAMS

The flow of activities which are done by the tools represented by activity diagram. This diagram provides more comprehensive understanding of the flow of the tool for end-users. Following activity diagrams are designated for each of the functionalities of the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD).

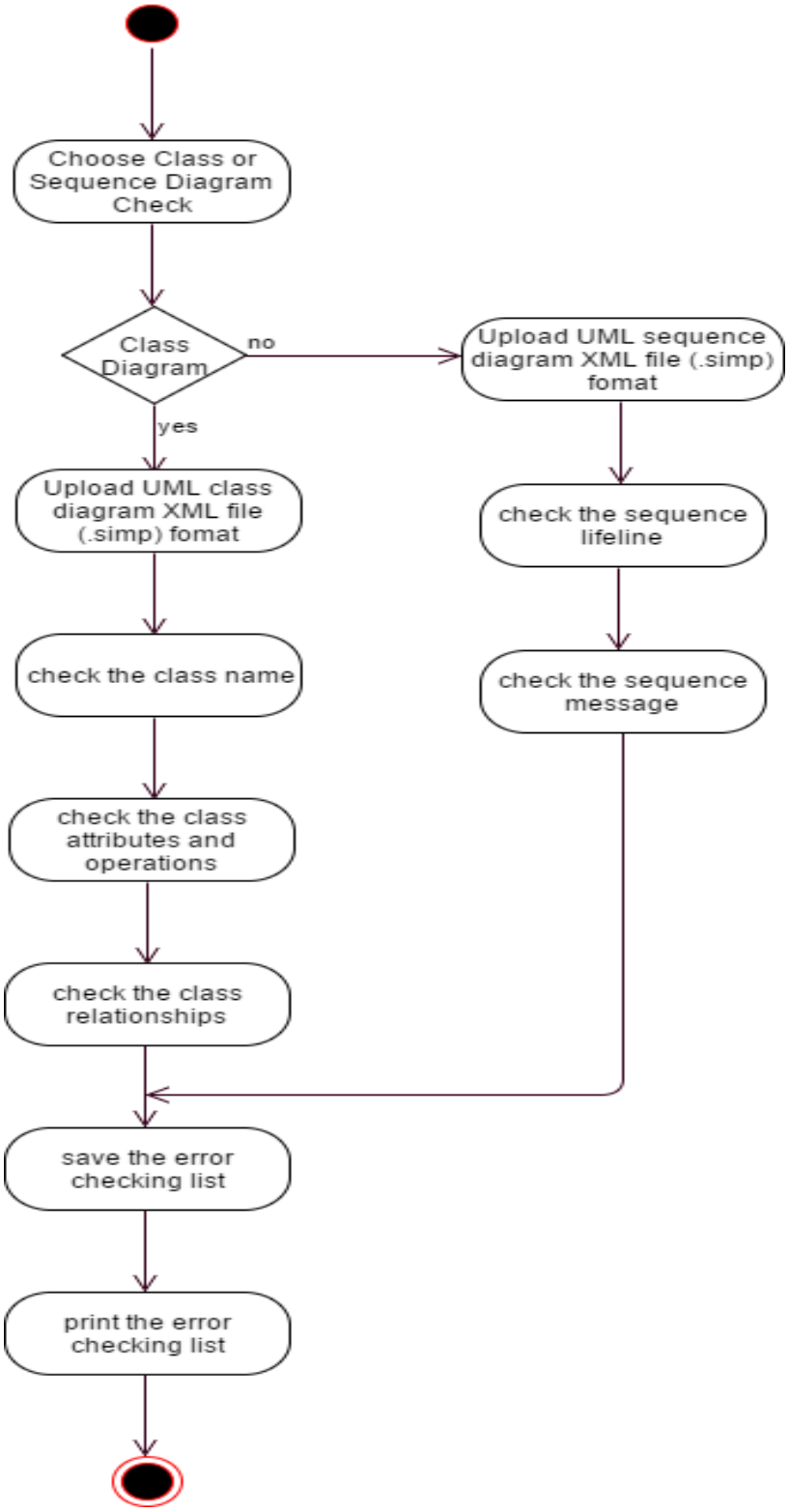


FIGURE 21: The activities of (CSeqD).

Activities of the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) is starting with uploading the UML class or sequence diagram XML file by end-users, followed with the error detection activity that involved the checking activity. If the uploaded XML file belong to the class diagram, the functionality of error checking is starting with checking class's name, followed by class's attributes and operations and finally checking the class's relationships. Otherwise, if uploaded XML file relates to the sequence diagram, the error checking activity will start with checking sequence's lifeline, followed by sequence's message. After that, end-users are able to decide either want to save the error detection lists into a ".txt" documents or not or just want to print it directly from the tool or both of them. FIGURE 21 is the activity diagram that is designated to show the flows of the activity of the (CSeqD) in generally. For FIGURE 22, it is the activity diagram for the "upload the UML class or sequence diagram XML file" functionality. As for the FIGURE 23, 24 and 25, they are the class diagrams for the "check the class's name", "check the class's attributes and operations" and "check the class's relationship" functionalities respectively. FIGURE 26 and 27 are related to the sequence diagram which are "sequence's lifeline" and "sequence's message" functionalities. For the "save the error detection lists" and "print the error detection lists" functionalities are shown in the FIGURE 28 and FIGURE 29. It can be said that most of the functionalities of the (CSeqD) are having straight- flow of the activity diagrams except for the last two functionalities which are the "save the error detection lists" and the "print

the error detection lists". Also the functionalities which are related to choose the "class" or "sequence" diagram error checking. All of these functionalities are having the diamond notation which is for the decision element.

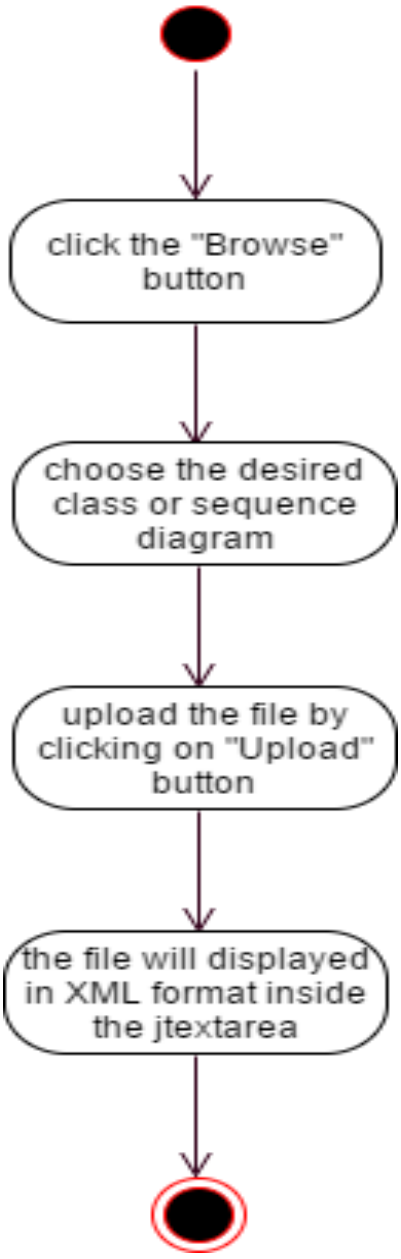


FIGURE 22: The activity diagram for "Upload UML class or sequence diagram XML file" functionality.

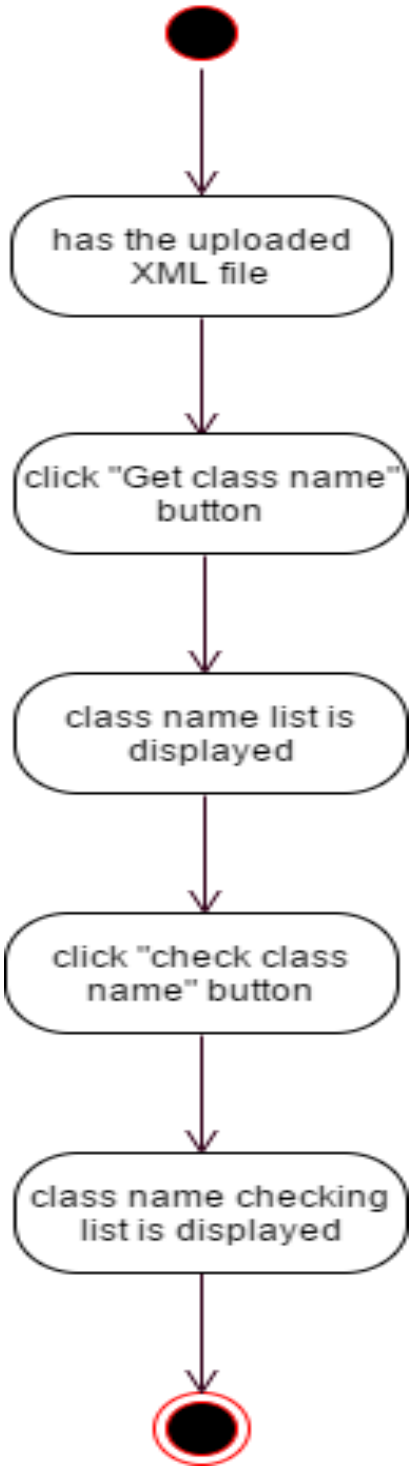


FIGURE 23: The activity diagram for “check the class’s name” functionality.

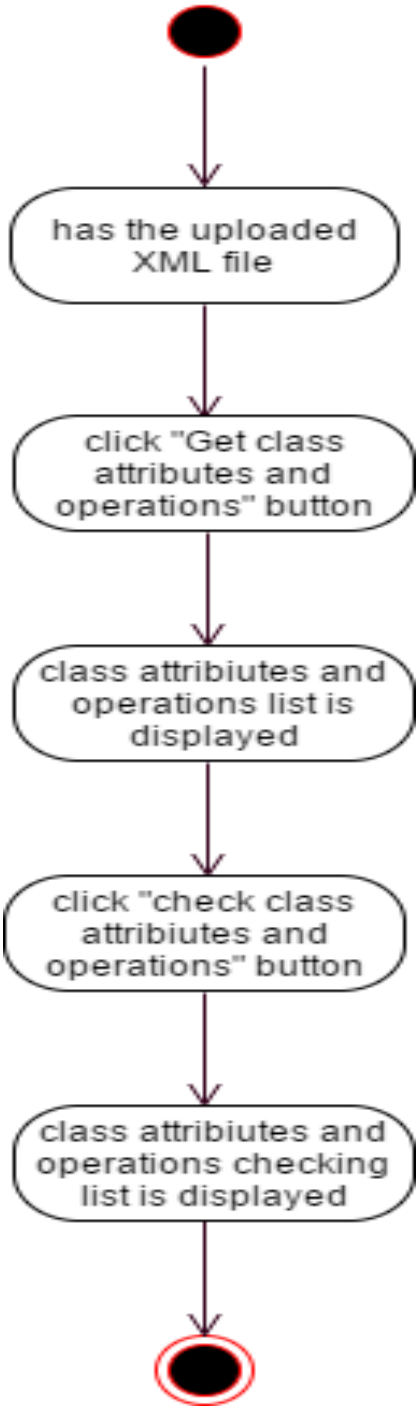


FIGURE 24: The activity diagram for “check the class’s attributes and operations” functionality.

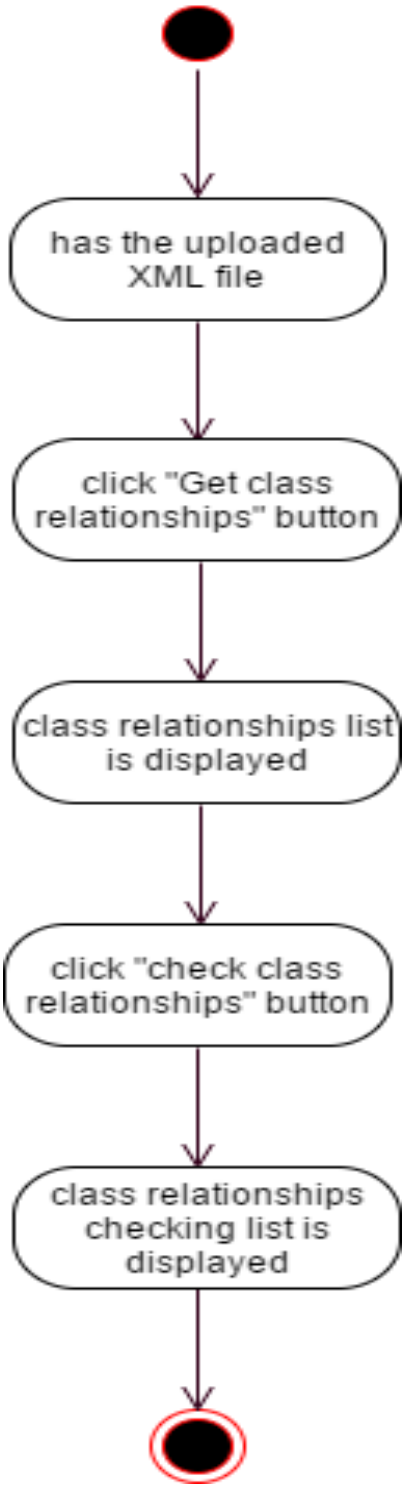


FIGURE 25: The activity diagram for “check the class’s relationship” functionality.

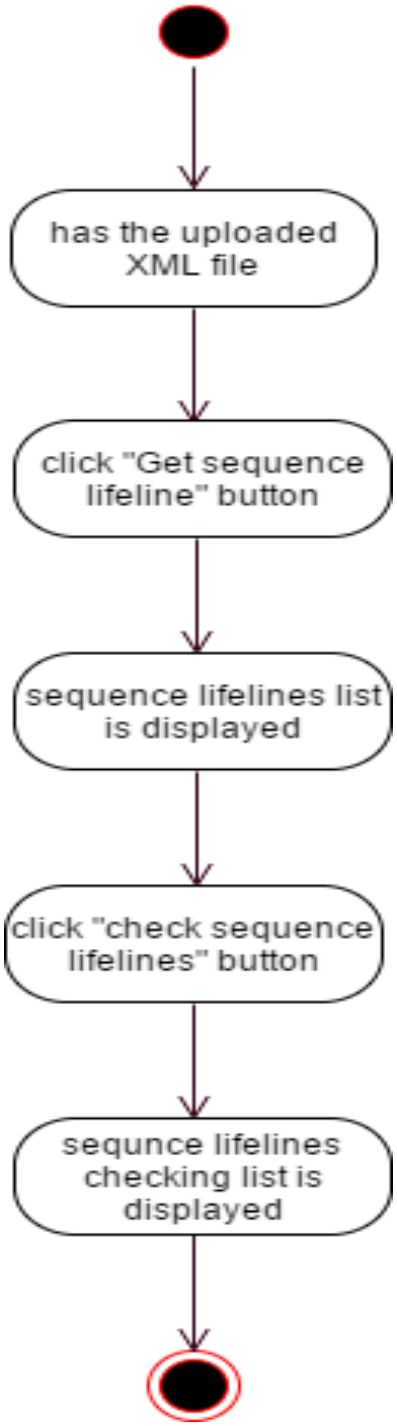


FIGURE 26: The activity diagram for “check the sequence’s lifeline” functionality.

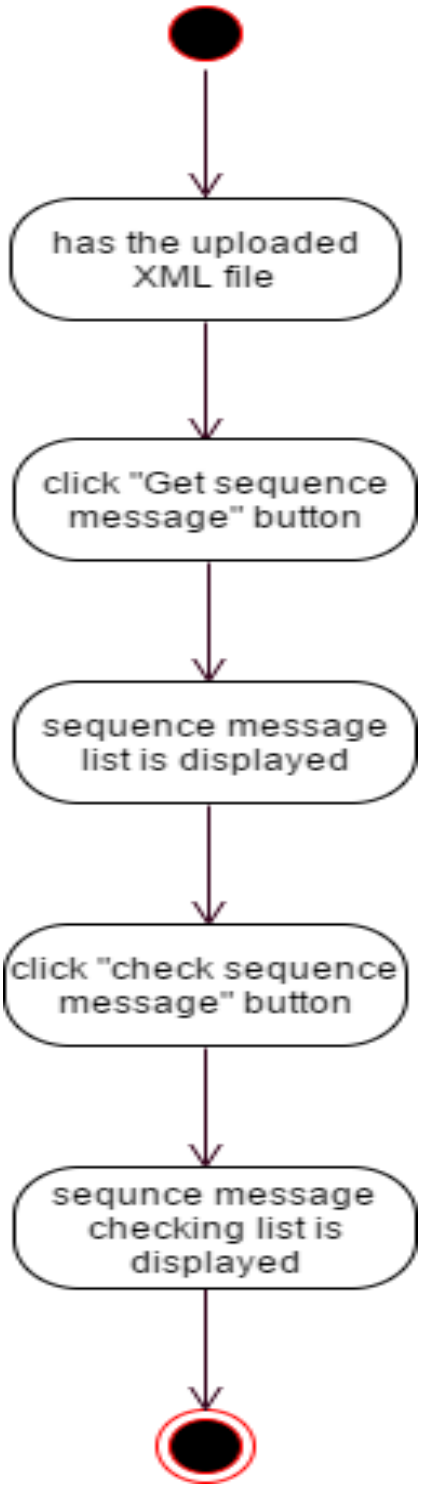


FIGURE 27: The activity diagram for “check the sequence message” functionality.

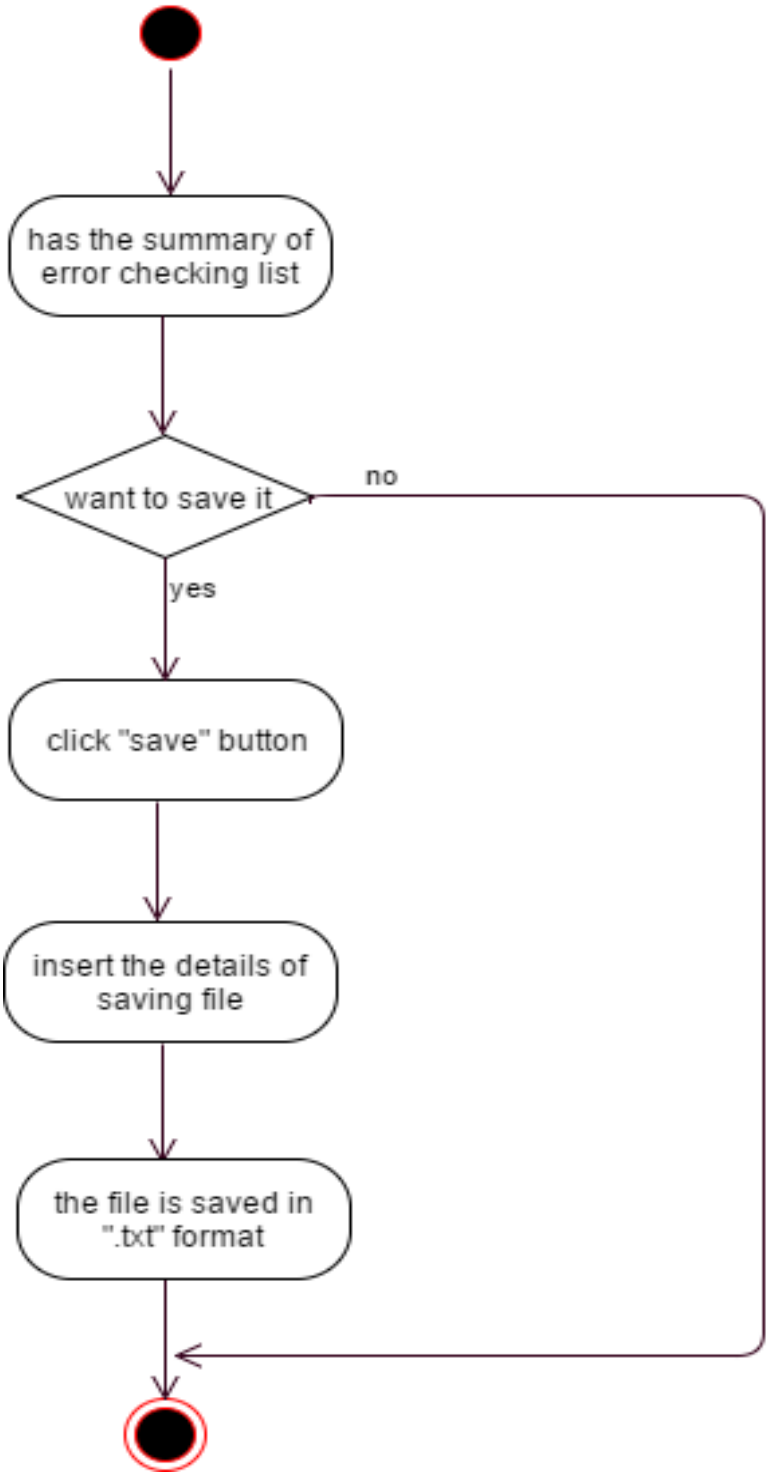


FIGURE 28: The activity diagram for “save the error detection lists” functionality.

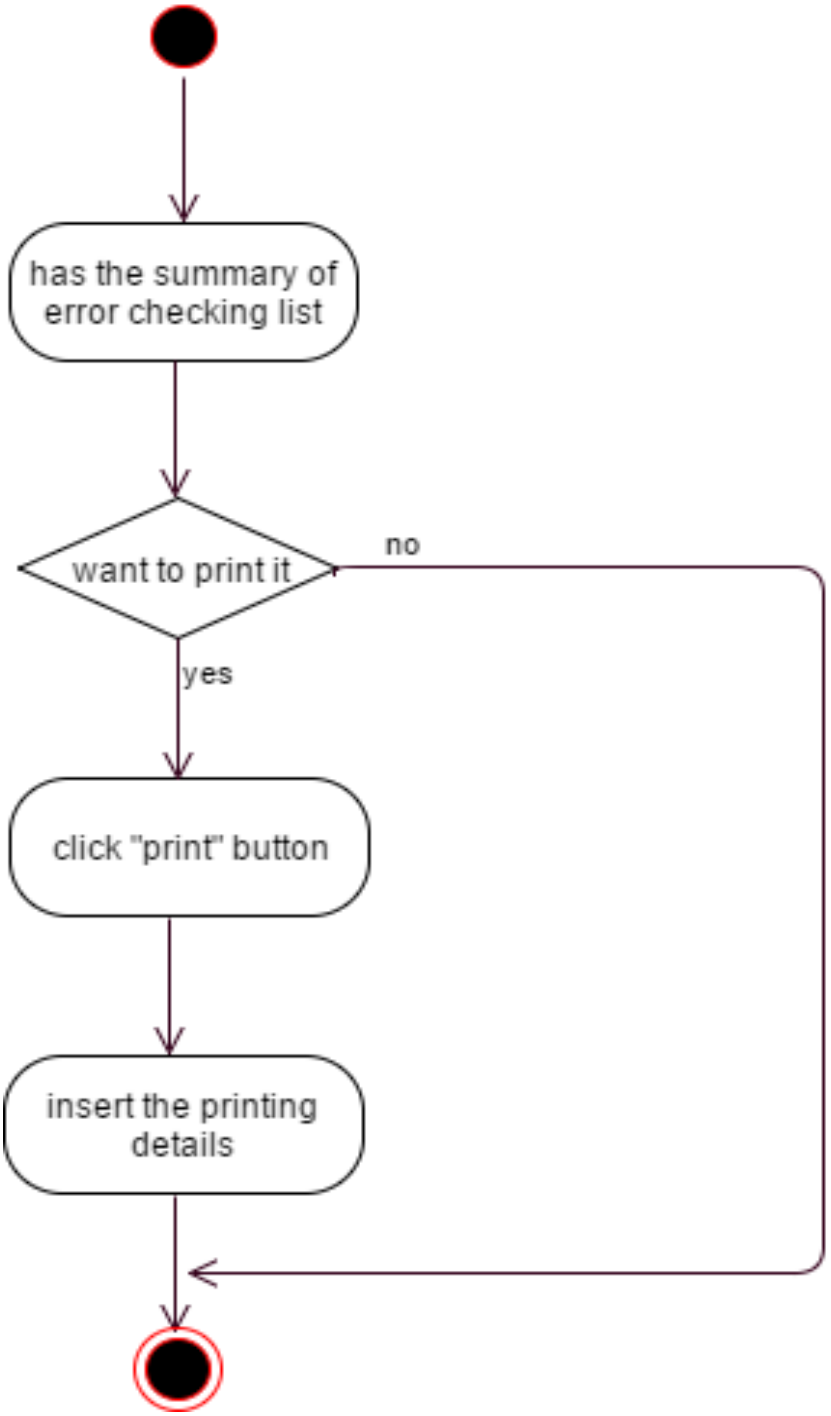


FIGURE 29: The activity diagram for “print the error detection lists” functionality.

4.1.2.4 CLASS DIAGRAM

FIGURE 30 is the class diagram of the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) that had been generated from the NetBeans IDE tool. From the diagram, it shows that all the Java classes that had been created during implementation of the (CSeqD), and how these classes are related to each other.

FIGURE 30: Class diagram for the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD).

4.1.2.5 DEPLOYMENT DIAGRAM

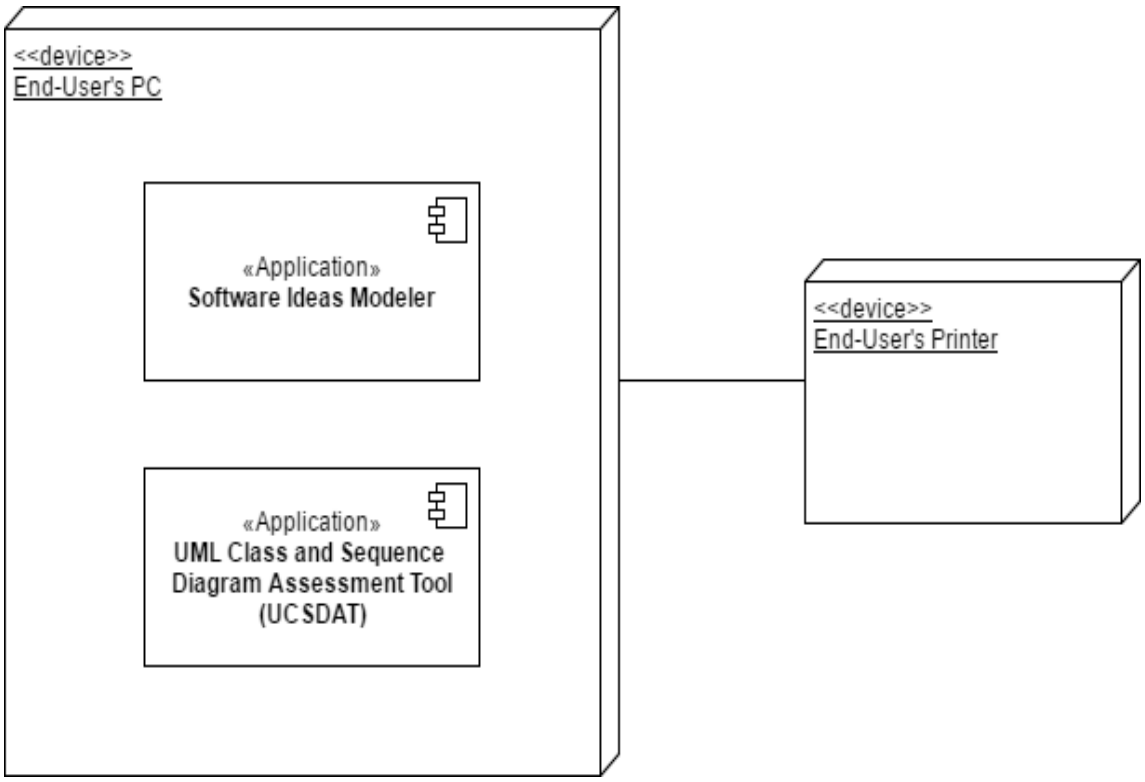


FIGURE 31: The deployment diagram of Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD).

FIGURE 31 represents the deployment diagram for the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD). Deployment diagram is a diagram that can be used to show the topology of the physical components that being while deployed this tool. Deployment diagram usually consists of the “node” that represents the hardware component that exist and “artifact” that represents the software components

being deployed in the nodes and how these nodes are related to each other. For (CSeqD) there is a node which is end-user's pc (laptop or desktop). This node contains two artifacts which are Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) used as error checking tool and Software Ideas Modeler tool used for modeling the class and sequence diagrams. In addition, this node is connected to the printer as there is function that allows end-users to print out the error detection lists that they obtained from the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD).

4.2 SYSTEM IMPLEMENTATION

Right after the designing the system, the implementation phase of Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) was begun. The chosen programming language for developing the tool was Java and NetBeans IDE as the developing software tool. The reason why Java was used because it is a programming language that has more familiar syntax or term compared to other programming language and the syntax used in this programming language was easy to understand. many tutorials or references are available for users in order to solve the problems.

4.2.1 INTERFACE IMPLEMENTATION

For the first step of implementation interface of the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) was implemented. This interface was implemented based on the low-fidelity interface prototype which were represented in chapter 4. In this phase that low-fidelity design converted to the high-fidelity interface. High-fidelity interfaces is a set of interfaces that had been developed with more detailed and can be functioning well based on the requirements that had been discussed earlier. The (CSeqD) interfaces were developed by JFrame form in NetBeans IDE developing software tool.

Since just by drag and dropping the java component from pallet in NetBeans IDE developing software tool into the form area I can develop a form and working with JFrame forms are easy and user friendly. Also, editing activities are very easy by using these JFram forms. I decided to use JFrame form in order to implement the (CSeqD). In addition, the source codes of the interfaces were auto-generated based on the components that involved during the drag-and-drop activities. These auto-generated source codes can be displayed in the “Source” tab.

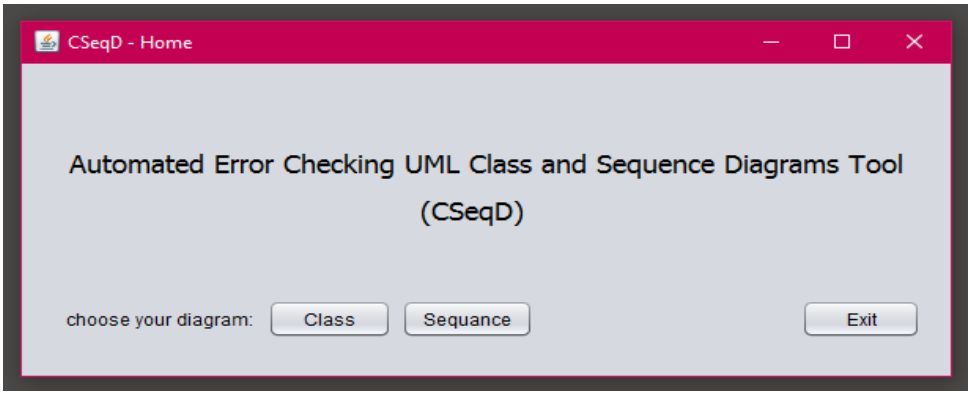


FIGURE 32: The “home” interface of (CSeqD).

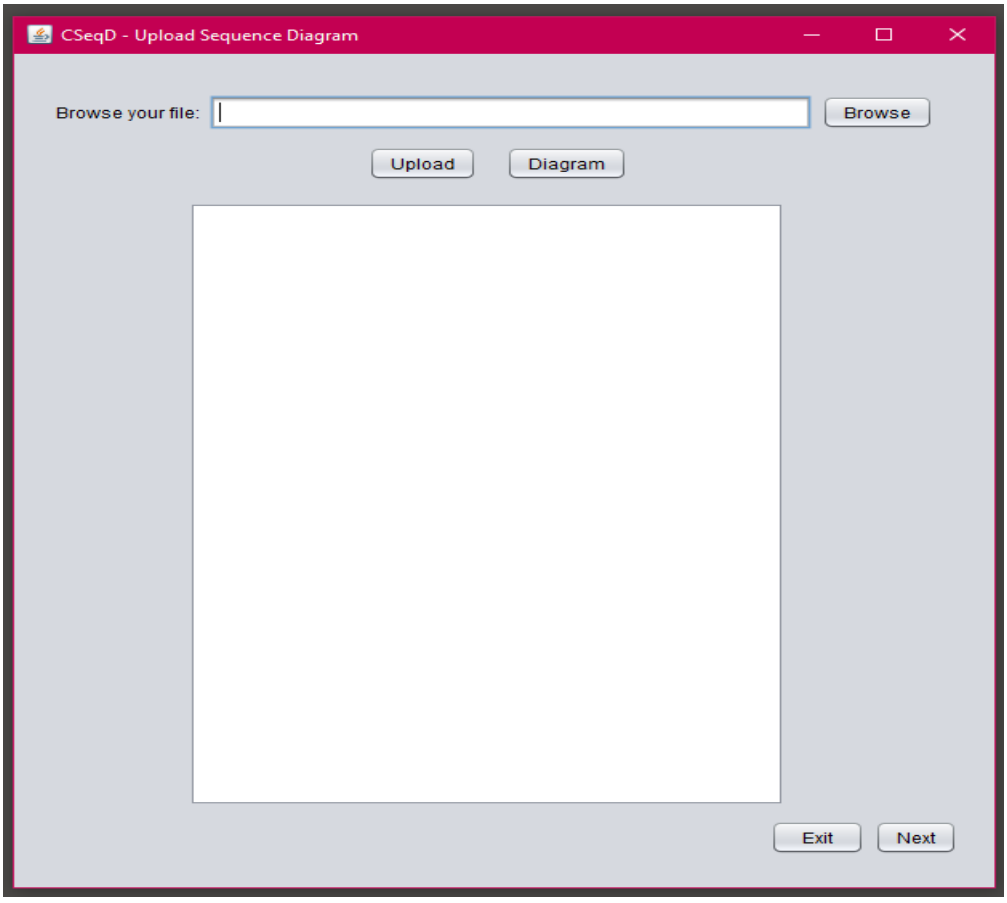


FIGURE 33: The “Upload Sequence Diagram” interface of (CSeqD).

Figure 32 shows the “Home” interface of (CSeqD) where end-users are able to choose whether they want to check the correctness of their class or sequence diagram. For example, an end-user wants to check sequence diagram, by clicking on the “sequence” button he/she will direct to the “Upload Sequence Diagram” page as shown in FIGURE 33.

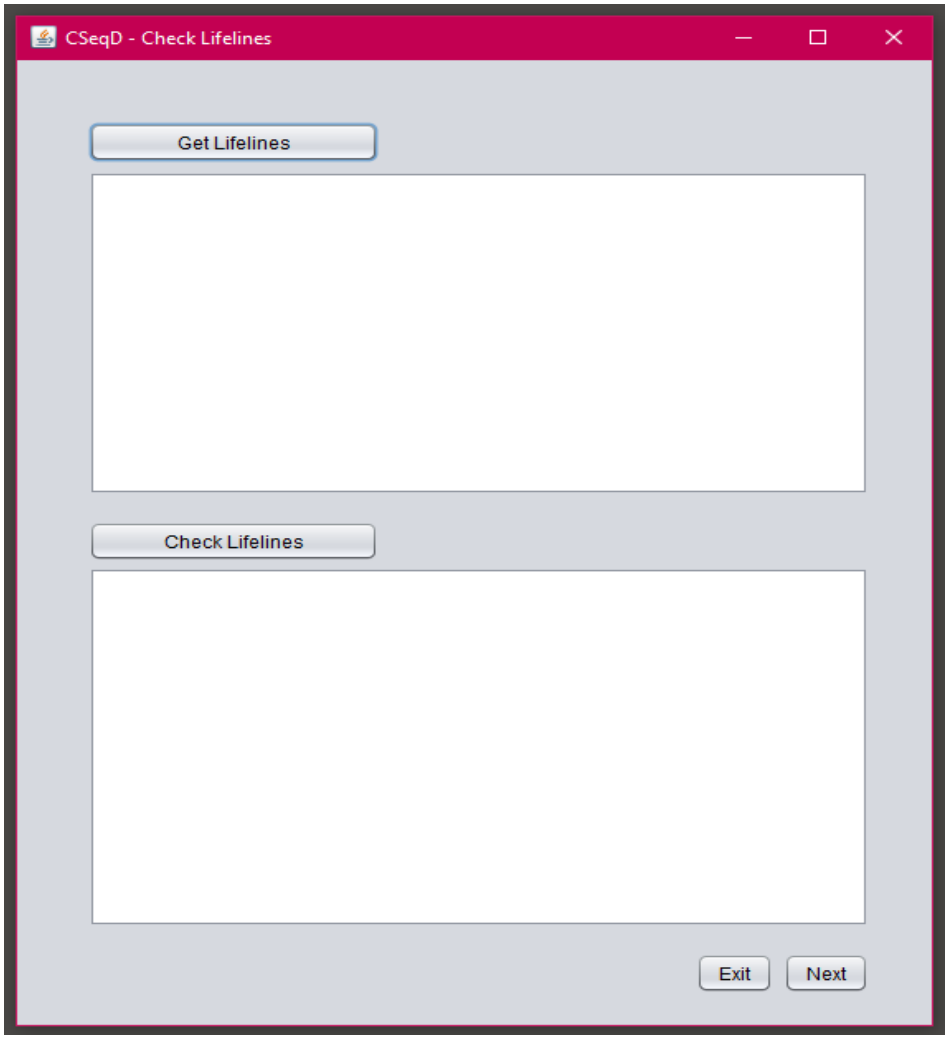


FIGURE 34: The “check Sequence Lifeline” interface of (CSeqD).

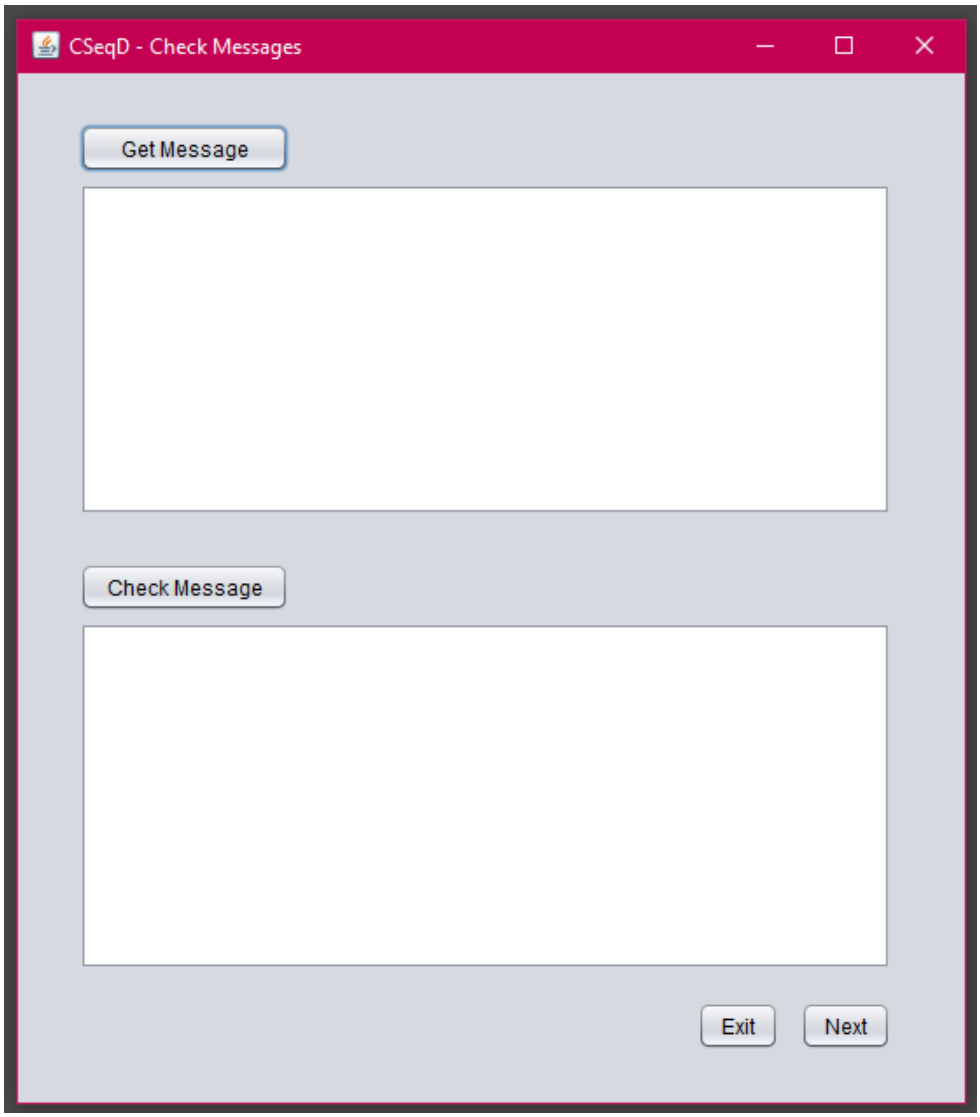


FIGURE 35: The “check Sequence Message” interface of (CSeqD).

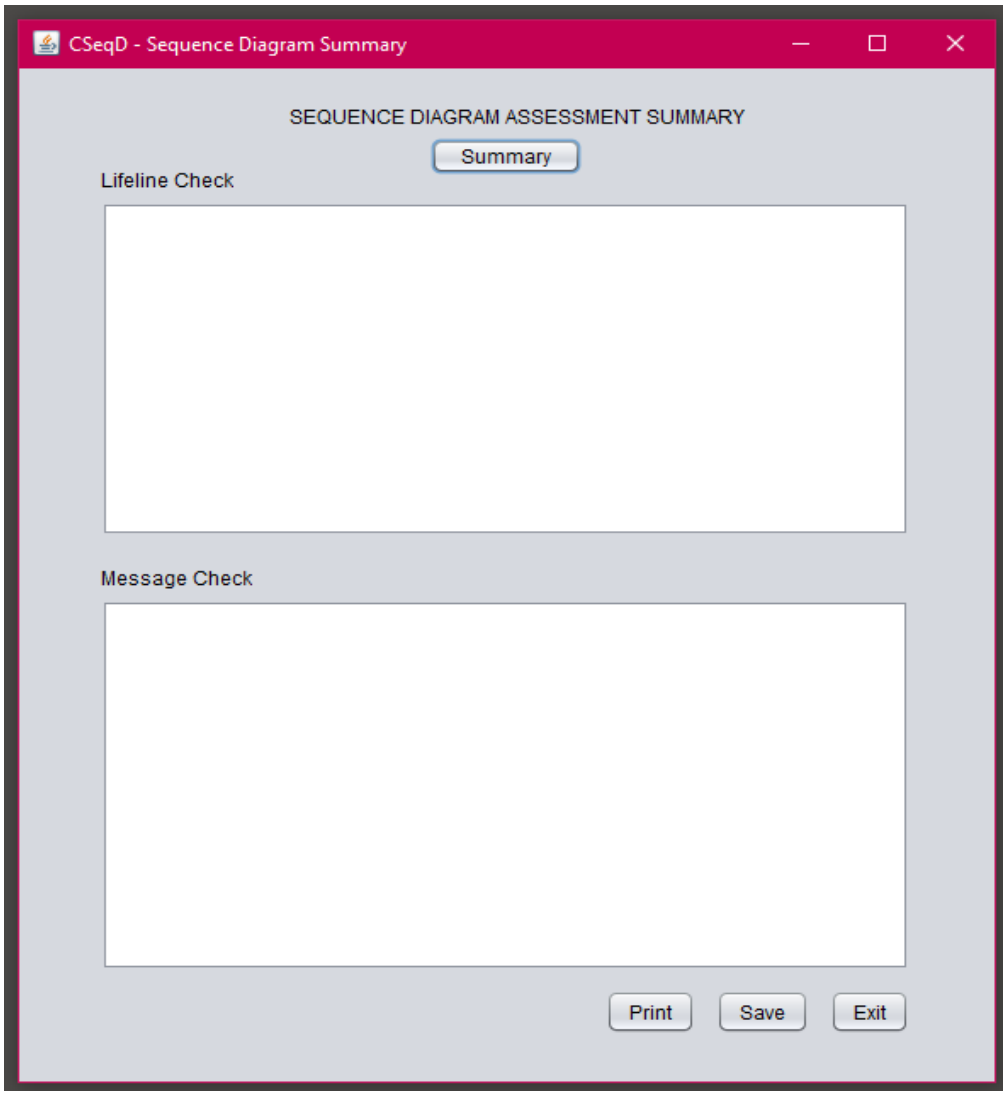


FIGURE 36: The “Sequence Diagram Error Checking Summary” interface of (CSeqD).

5.1.2 JAVA CLASS IMPLEMENTATION

After finishing the interface development phase, the project should proceeded with development of java classes. These classes are developed according to the requirements of the Automated Error Checking for UML

Class and Sequence Diagrams Tool (CSeqD) which are identified earlier. For the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD), the java classes were developed for the main requirement of the tool which is the checking activity that being involved in “assess the UML class and sequence diagram” functionality.

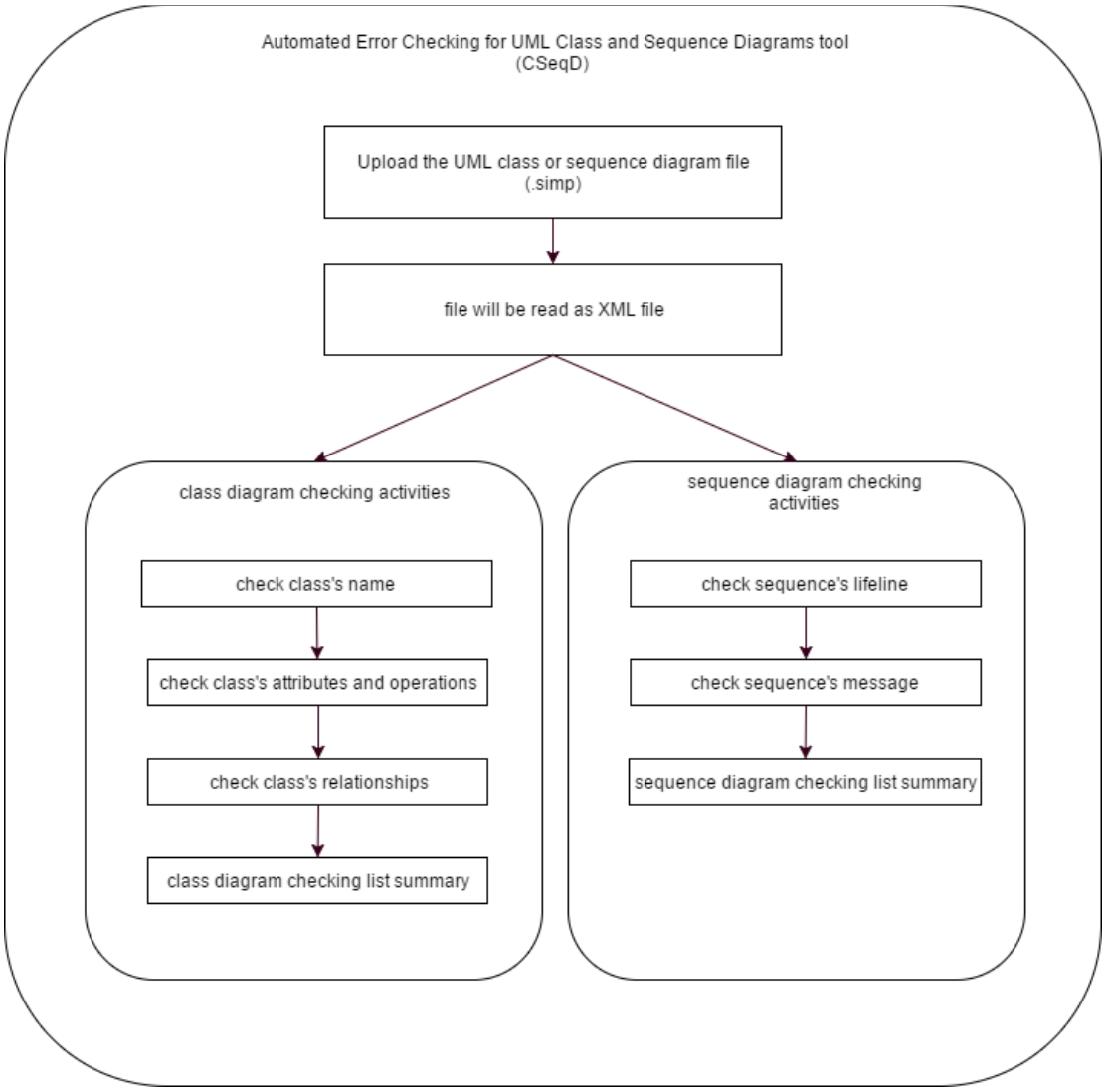


FIGURE 37: The flow of the checking activity in (CSeqD).

According to the FIGURE 37, the checking activity in the (CSeqD) begins when there is a file that is in ".simp" format which was uploaded into (CSeqD). The ".simp" format is the extension format the file that had been saved using the Software Ideas Modeler modeling tool.

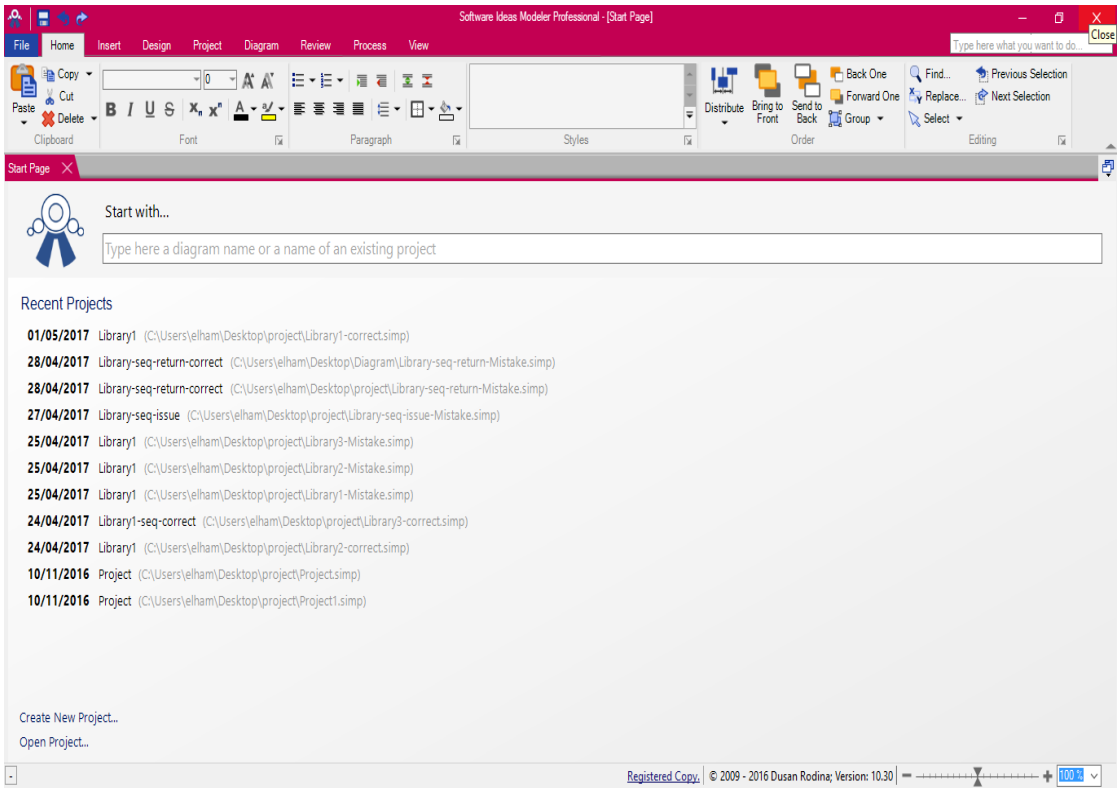


FIGURE 38: The started interface of the Software Ideas Modeler.

The exporting files will saved in (.simp) format. When this file uploaded in the (CSeqD) it will viewed as a XML format. the checking activity in (CSeqD) is conducted based on the XML structure of the class or sequence diagram. XML or Extensive Markup Language is a markup

language that defines a set of rules for encoding documents in a format which both human-readable and machine- readable and it is widely used for the representation of the arbitrary of the data structures .

The reason why I decided to use parse technique and Production-Rule based technique to do the checking activity is that, (CSeqD) is able to read the class and sequence diagrams textually in XML format. Parse technique or generally knows as parsing is the process of analyzing a string of symbols, either in natural language or in computer languages, conforming to the rules of a computer languages while the production-rule based technique that involving the IF-ELSE statement during the execution and this already being discussed earlier as the Rule-based technique.

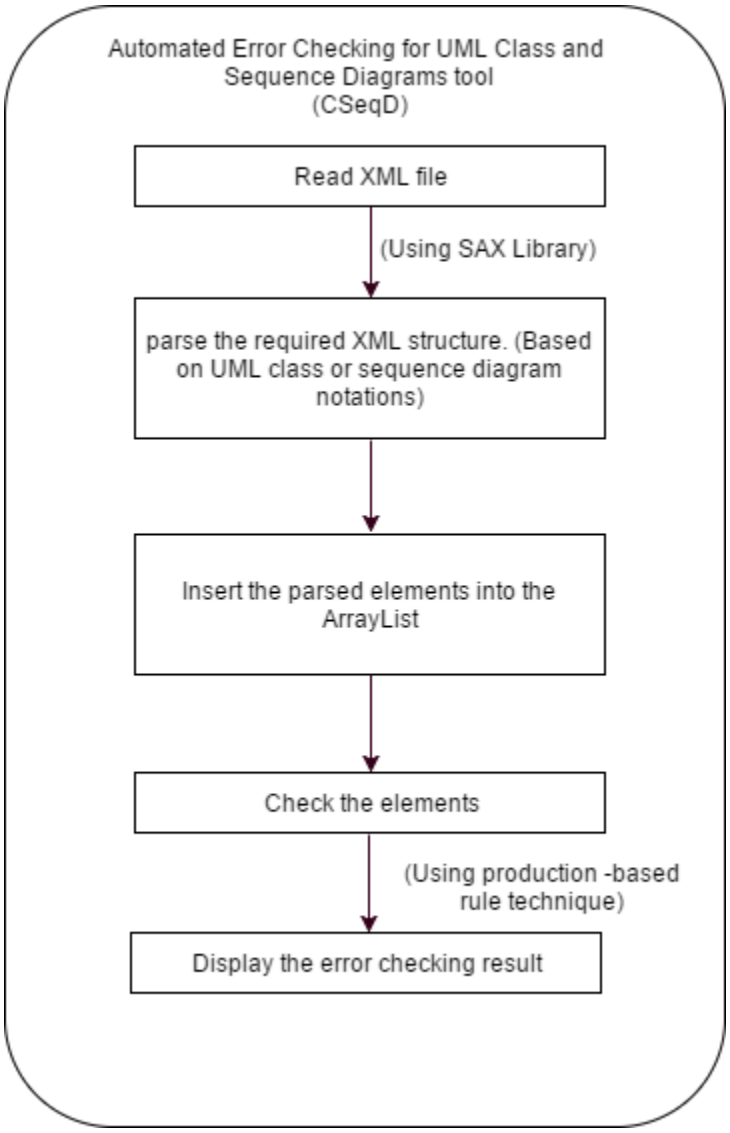


FIGURE 39: The checking activity that involved in (CSeqD).

The process of checking activity in Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) is shown by FIGURE 50. After uploading the XML file into the tool, the parsing technique will start which is used SAX library. SAX or Simple API for XML is an event-driven online algorithm for parsing XML documents with an API interface developed

by the XML-DEV mailing list. SAX parsers operate on each piece of the XML document in a sequential way. Because this SAX library only need to report each parsing event as it happens and normally discards almost all of that information once reported which means that it required minimum memory for the parses that being used in this library, I decided to use this library in order to parsing the XML file. Moreover, in terms of parsing speed, the document processing process is generally faster than other parsers like DOM-style parser.

For the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD), there are six java classes that had been created that consist of the SAX library. These classes were readXMLFile, readAttributeXML, readOperationXML, readRelationshipXML, readLifelineXML and readMessageXML (Refer to the class diagram in CHAPTER 4). As for the readXMLFile class, it was used to parse the information about the class name while the readAttributeXML and readOperationXML classes are for the class's attributes and operations, and the readRelationshipsXML class is for the class's relationships. readLifelineXML was used to parse the information of sequence diagram's lifeline and finally, readMessageXML is for sequence diagram's message parsing. An example of part of the source code that use SAX library are as follow:

```
21
22          @Override
23      public void startElement(String uri,String localName, String Ename, Attributes attributes){
24          temp = " ";
25          if(Ename.equalsIgnoreCase("item")){
26              messagecheck = new MessageErrorChecking();
27              messagecheck.setMessageName(attributes.getValue("name"));
28              messagecheck.setMessageType(attributes.getValue("type"));
29              messagecheck.setMessageVisibility(attributes.getValue("visibility"));
30          }else if (Ename.equalsIgnoreCase("sequence-action")){
31              messagecheck.setMessageType2(attributes.getValue("type"));
32              messagecheck.setMessageMessage(attributes.getValue("message"));
33              messagecheck.setMessageFrom(attributes.getValue("from"));
34              messagecheck.setMessageTo(attributes.getValue("to"));
35              messagecheck.setMessageReturn(attributes.getValue("return-value"));
36          }
37
38          }
39
40          @Override
41      public void endElement (String uri, String localName, String Ename) throws SAXException{
42          if(Ename.equalsIgnoreCase("item")){
43
44              if (messagecheck.getMessageType().equalsIgnoreCase("sequence-action"))
45              checkmessage.add(messagecheck);
46          }
47        }
48
```

FIGURE 40: A part of readMessageXML class's source code that implement SAX library.

As for FIGURE 40, it represented that the SAX library was extracted the information required from the XML structure. For instance, since this readMessageXML class was intended to get the information for the sequence diagram message, and then the SAX parser will parse all the information that needed from the XML documents. Next after the parsing activity, the extracted information was be inserted into a set of Arraylist. It same goes with other SAX library that being in other classes.

The next step after parsing activity and extracting the needed information from uploaded XML file, the checking activity will started by using the production-based rule technique. This technique was being developed in different java class in order to avoid confusion while developed this tool. The other java classes that were being created for this activity are ClassErrorChecking, AttributeErrorChecking, OperationErrorChecking, RelationshipErrorChecking, LifelineErrorChecing and MessageErrorChecking. These classes are developed for check the class's name, check the class's attributes, check the class's operations, check the class's relationships, check the sequence's lifeline and check the sequence's message. Following the example of the production-rule based technique which used to develop (CSeqD):

```
86
87              { if(checkType(attributeType))
88                    sa1 = "CALSS ATTRIBUTES SHOULD HAVE DATA TYPE - " + "\u2573";
89                else
90                    sa1 = "YOUR CLASS ATTRIBUTE HAS DATA TYPE - " + "\u2713";
91                }
92
93              { if(checkVisibility(attributeVisibility))
94                    sa2 = "CALSS ATTRIBUTES SHOULD HAVE VISIBILITY - " + "\u2573";
95                else
96                    sa2 = "YOUR CLASS ATTRIBUTE HAS VISIBILITY - " + "\u2713";
97                }
98
99              { if(spaceLetter(attributeName))
100                   sa3 = "CALSS ATTRIBUTES NAME SHOULD NOT HAVE SPACE LETTER - " + "\u2573";
101               else
102                   sa3 = "YOUR CLASS ATTRIBUTE DOES NOT HAVE SPACE LETTER - " + "\u2713";
103               }
104             { if(withAndOr(attributeName))
105                   sa4 = "ATTRIBUTE NAME SHOULD NOT INCLUDE 'AND' OR 'OR' WORDS - " + "\u2573";
106               else
107                   sa4 = "YOUR ATTRIBUTE NAME DOES NOT INCLUDE 'AND' OR 'OR' WORDS - " + "\u2713";
108               }
109
110           { if(reserved(attributeName))
111                 sa5 = "ATTRIBUTE NAME SHOULD NOT INCLUDE RESERVED WORDS - " + "\u2573";
112               else
113                   sa5 = "YOUR ATTRIBUTE NAME DOES NOT INCLUDE RESERVED WORDS - " + "\u2713";
114               }
```

FIGURE 41: A part of “AttributeErrorChecking” class’s source code that implements production-rule based technique.

Similarly for other java classes were done by using the production-rule based technique. The next step right after design and development of interfaces and java classes is testing and evaluation of the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD).

CHAPTER 5 – EVALUATION, TESTING AND DEPLOYMENT

5.1 EVALUATION

In order to evaluate the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD), two kinds of evaluation conducted. First, to figure out whether the tool is able to fill the gaps and shortcomings of other existing tool. A comparison between (CSeqD) and six other UML tools were conducted which is showed by table 6. Second, the class and sequence diagrams for project works of five groups of bachelor level computer science students was applied into the (CSeqD) tool and check how the tool is able to detect the errors of their diagrams.

5.1.1 EVALUATION BY COMPARISON

The comparison of proposed functionality between (CSeqD) and six other existing tools is available as follow:

Table 6: comparison between (CSeqD) and six other existing tools

	Error Checking Diagrams			Target User		
Tool Name	Usecase Diagram	Class Diagram	Sequence Diagram	Student	Professional	End-User
UMLGrader	X	✓	X	✓	X	X
UMLint	✓	✓	X	✓	X	X
minimUML	X	✓	X	✓	X	X
NClass	X	✓	X	X	✓	X
StudentUML	X	✓	X	✓	X	X
UCDAT	X	✓	X	✓	X	X
CSeqD	X	✓	✓	✓	X	✓

As Table 6 mentioned most of the existing tools were produced for the use of students and professionals users who have comprehensive knowledge about UML modelling and design. But, (CSeqD) is designed to give feedback to the end-users who are novice or from other fields of study on their class and sequence diagrams. In addition, none of existing tools provides feedback on sequence diagram's error while (CSeqD) checks the errors of class and sequence diagrams.

5.1.2 EVALUATION BY CHECKING STUDENTS' PROJECT REPORT

The class and sequence diagrams of five groups of bachelor level students in computer science field were checked by the (CSeqD) and check the efficiency of the tool.

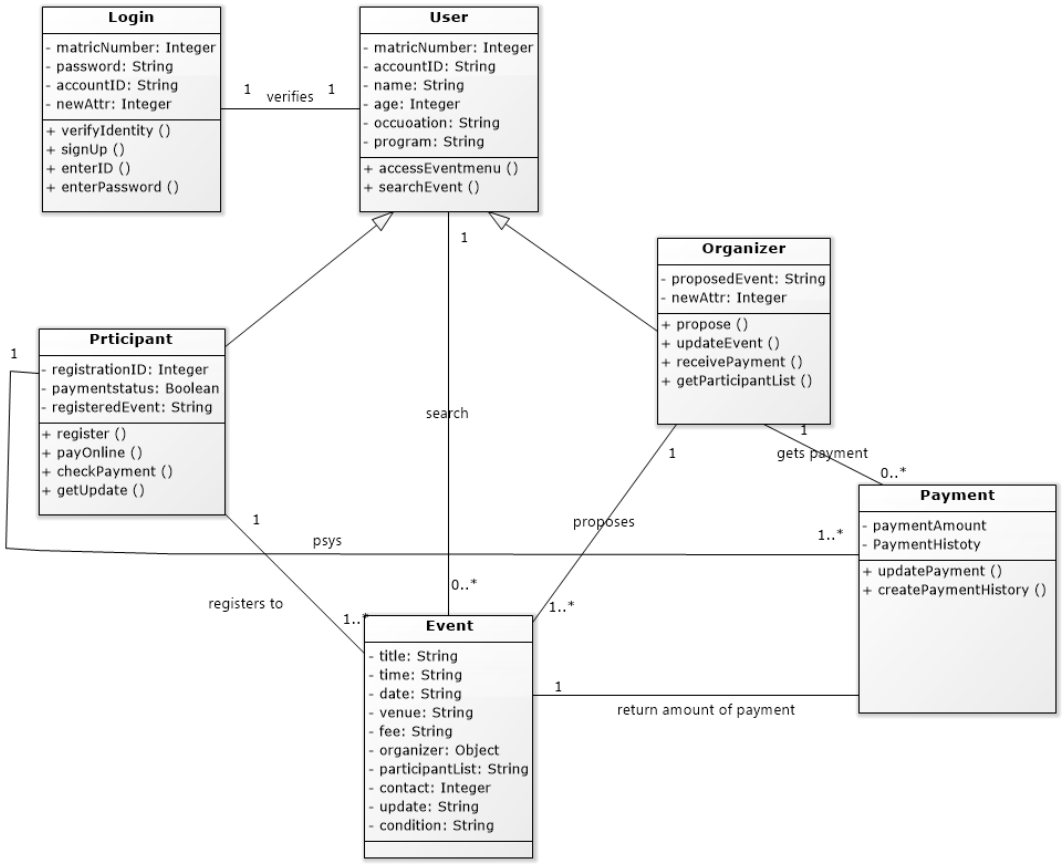


FIGURE 42: Class diagram for (event information system check) designed by one of the groups

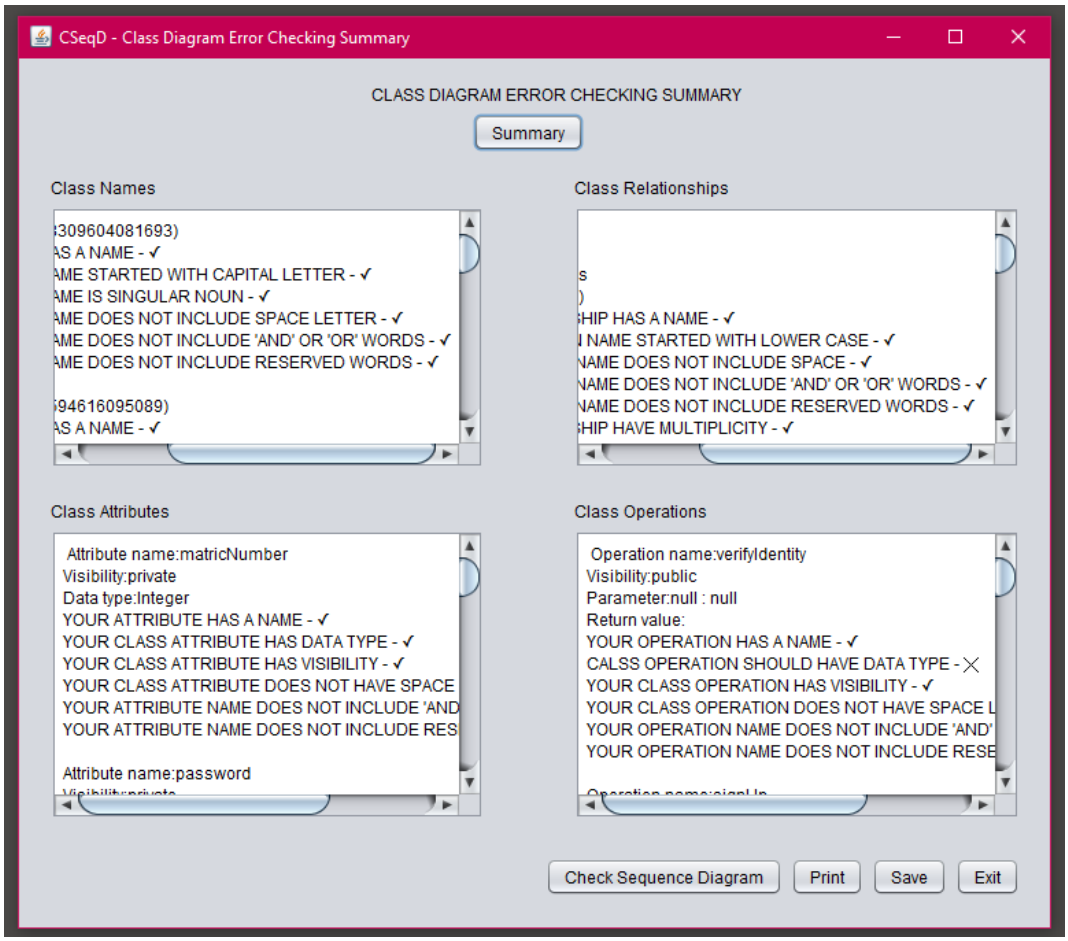


FIGURE 43: checking the class diagram for (event information system check) designed by one of the groups

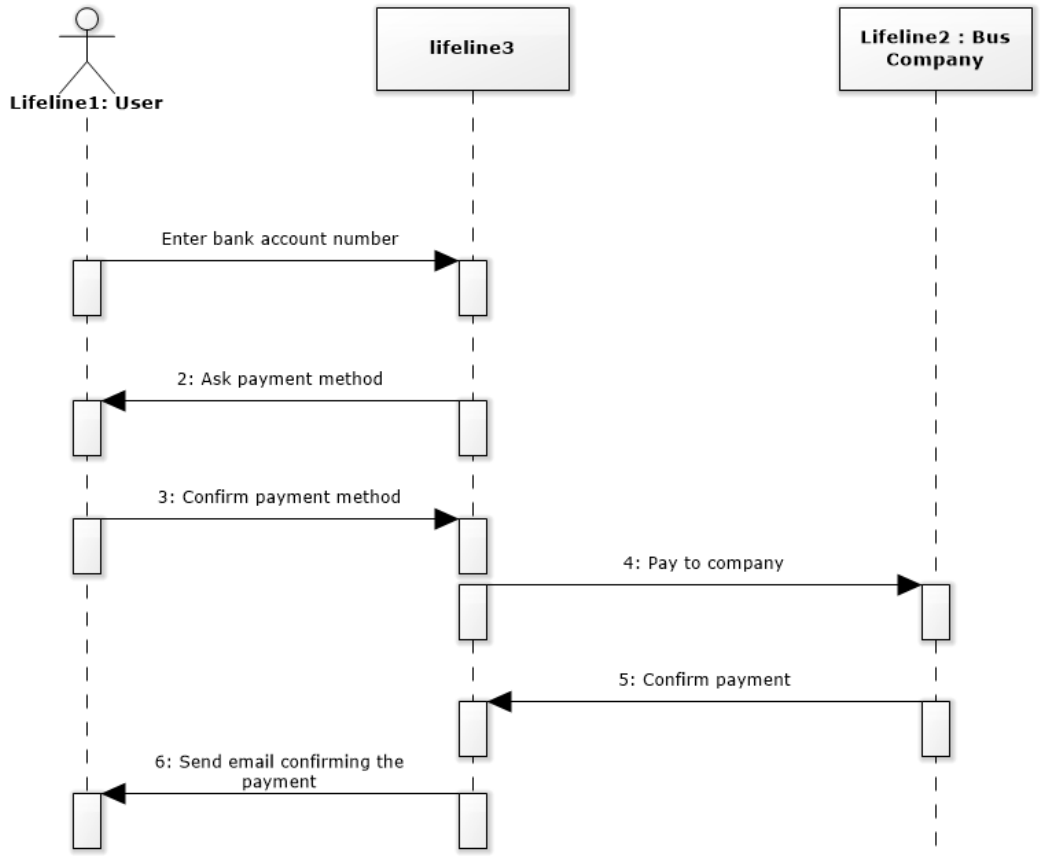


FIGURE 44: sequence diagram for (online bus ticketing system) designed by one of the groups

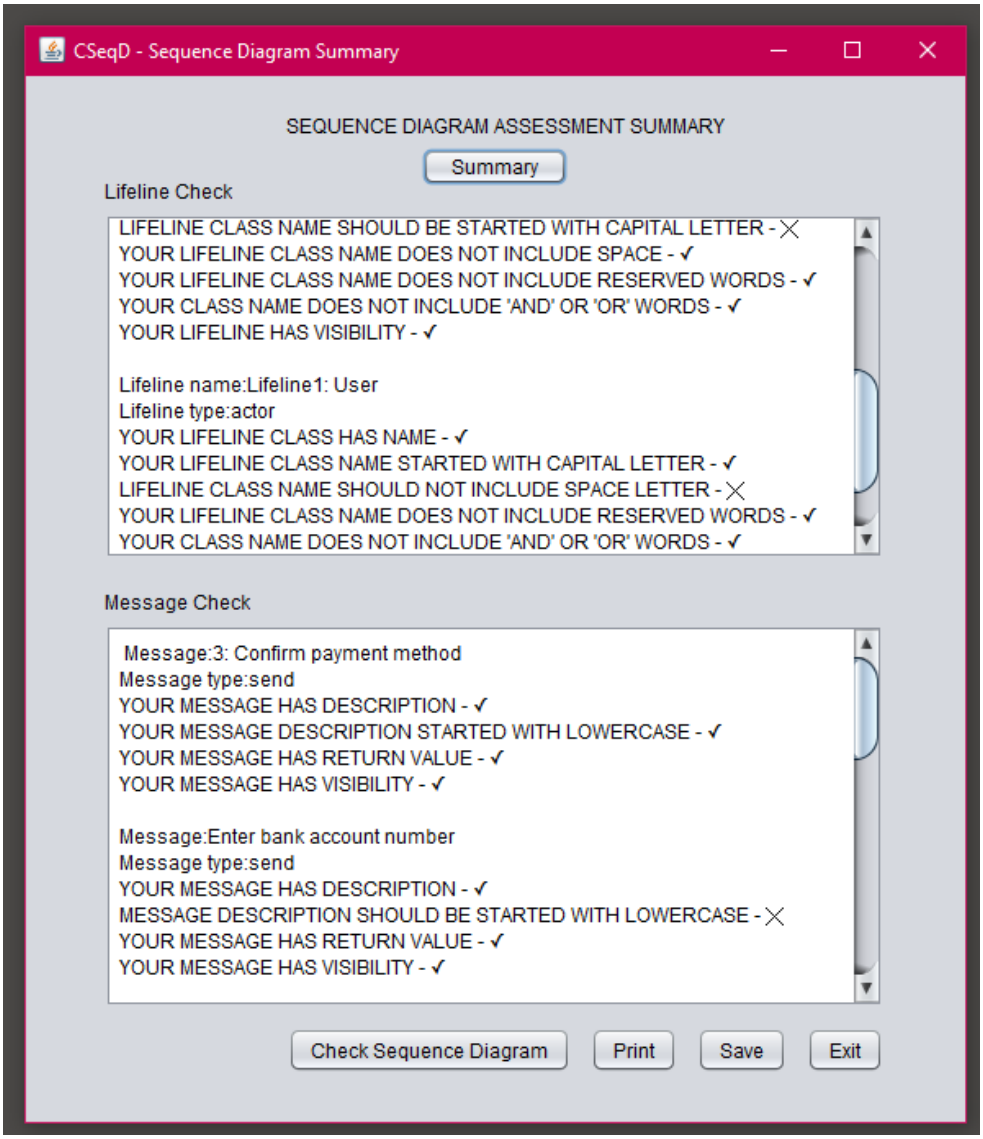


FIGURE 45: checking the sequence diagram for (online bus ticketing system) designed by one of the groups

Table 7 represents the rules checked by the (CSeqD) and the errors detected in students projects.

Table 7: detected errors in student's projects

Design Critique Rules	Event Information System	Online Bus Ticketing System	The Mobile Dictionary	Student Attendance System	Student Task Manager
CLASS NAME SHOULD BE STARTED WITH CAPITAL LETTER	✓	✓	✓	✓	✓
CLASS NAME SHOULD NOT INCLUDE SPACE LETTER	✓	X	✓	X	✓
CLASS NAME SHOULD NOT INCLUDE RESERVED WORDS	✓	✓	✓	X	✓
CALSS ATTRIBUTES SHOULD HAVE DATA TYPE	X	X	X	X	✓
ATTRIBUUTE SHOULD HAVE A NAME	✓	✓	✓	✓	✓
CALSS OPERATION SHOULD HAVE VISIBILITY	✓	✓	✓	✓	✓
CALSS OPERATION SHOULD HAVE DATA TYPE	X	✓	X	X	X
RELATIONSHIP SHOULD HAVE MULTIPLICITY	✓	X	✓	✓	✓
RELATION NAME SHOULD NOT INCLUDE SPACE LETTER	X	✓	✓	✓	X
LIFELINE CLASS SHOULD HAVE NAME	✓	✓	✓	X	✓
LIFELINE SHOULD HAVE VISIBILITY	✓	✓	✓	✓	✓
MESSAGE SHOULD HAVE DESCRIPTION	✓	✓	✓	✓	✓

Based on the efficiency formula which is :

Efficiency = (No. of defects Resolved / Total No. of Defects Submitted)* 100

The results reviled that the (CSeqD) is able to detect all the defects which is expected to find. It means maybe there are other errors and defects in the students diagrams but the (CSeqD) tool finds all the structural errors that designed to detect. The efficiency of the tool based on expected functionality is 100 percent

5.2 TESTING PHASE

After implementing phase, system should be tested to check either the tool can be functioning according to all the functionalities. For the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD), there are two testing that were conducted which are testing via diagrams and unit testing.

5.2.1 TESTING VIA UML DIAGRAMS: Class and Sequence Diagrams

In order to check whether the tool is able to meet the requirements, a library scenario is conducted and five different diagrams in small, medium and large scale are designed and checked by the tool twice. At first, three class diagram in small, medium and large scale and two sequence diagram with “Issue Book” and “Return Book” scenario which were designed with the correct notation modeled by Software Ideas Modeler tool and the (.simp) exporting format file uploaded in to the (CSeqD) and the tool check the diagrams hundred percent correctly. In the next step the same diagrams were modeled with some mistakes and check the system with not correct systems. The tool was able to find and report the ninety percent of the mistakes. The Figures of the checking process are provided as follow:

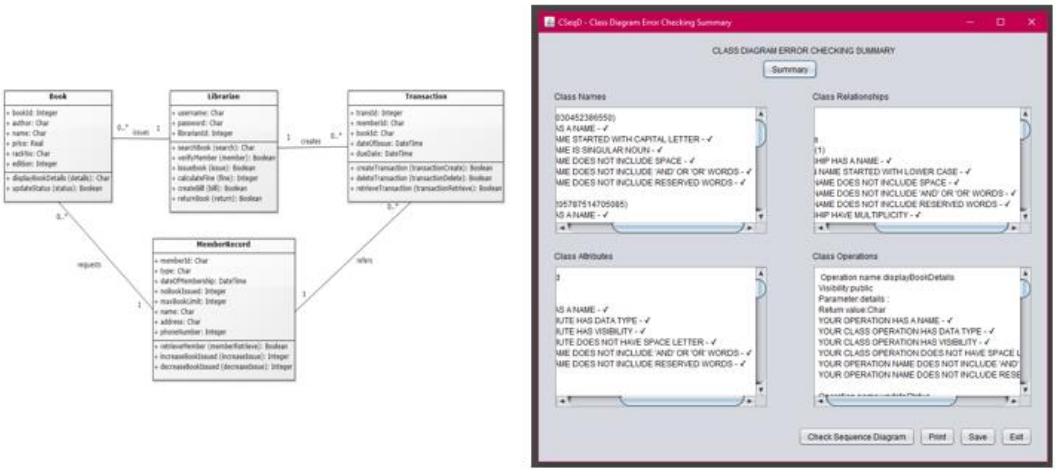


FIGURE 46: small scale Library scenario class diagram with correct notations and the checking result which shows all the checking notations as correct.

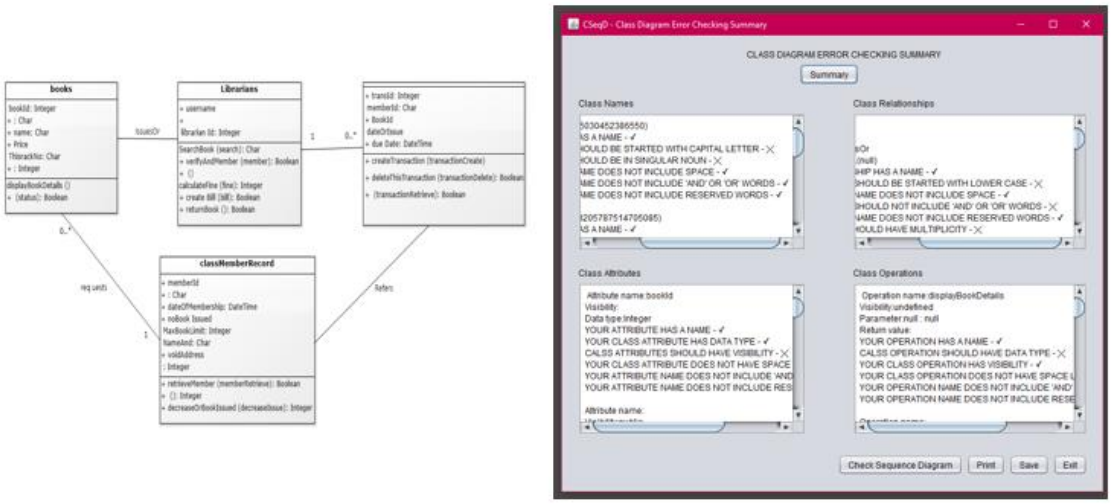


FIGURE 47: small scale Library scenario class diagram with some wrong notations and the checking result which shows the detected errors by the tool.

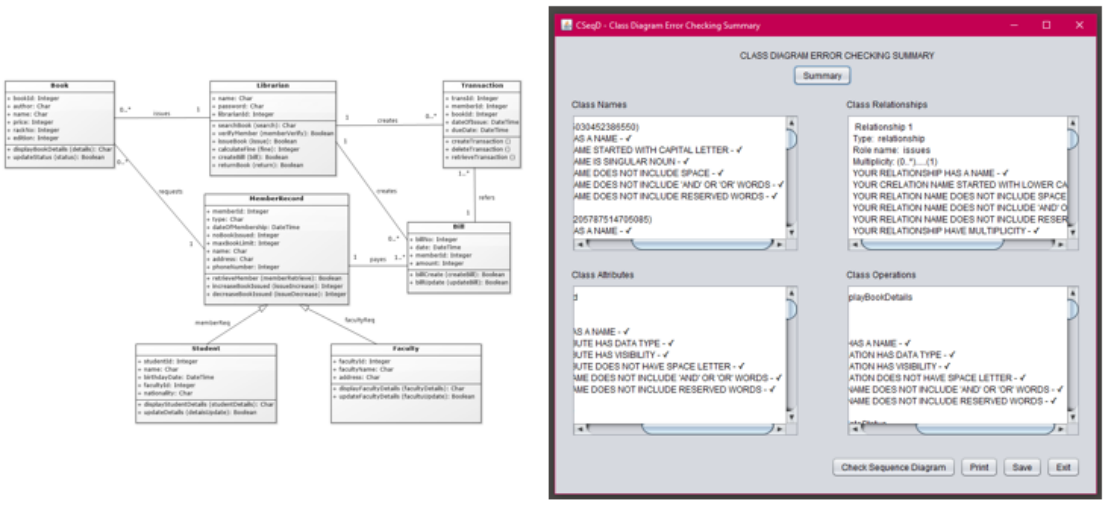


FIGURE 48: medium scale Library scenario class diagram with correct notations and the checking result which shows all the checking notations as correct.

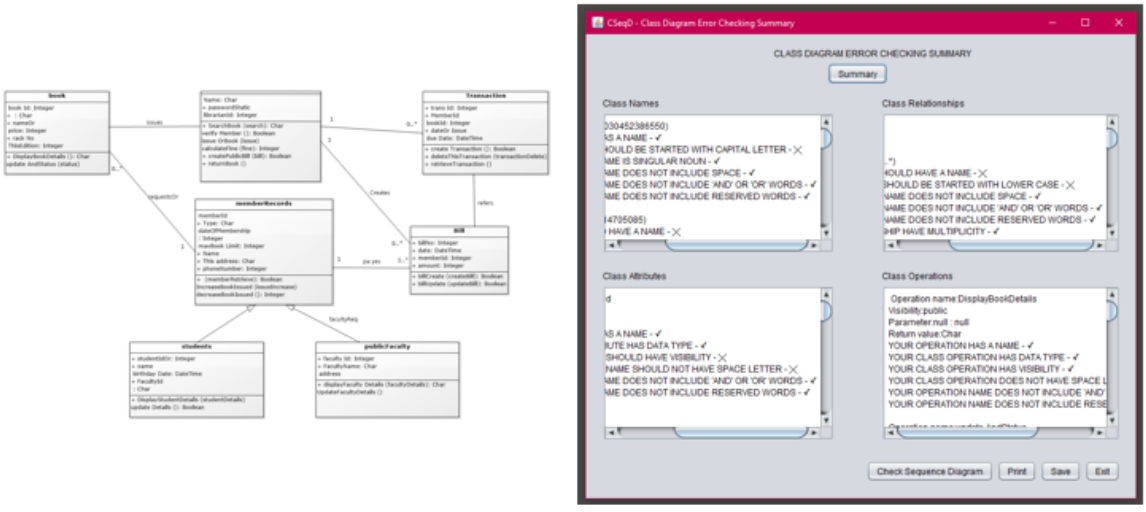


FIGURE 49: medium scale Library scenario class diagram with some wrong notations and the checking result which shows the detected errors by the tool.

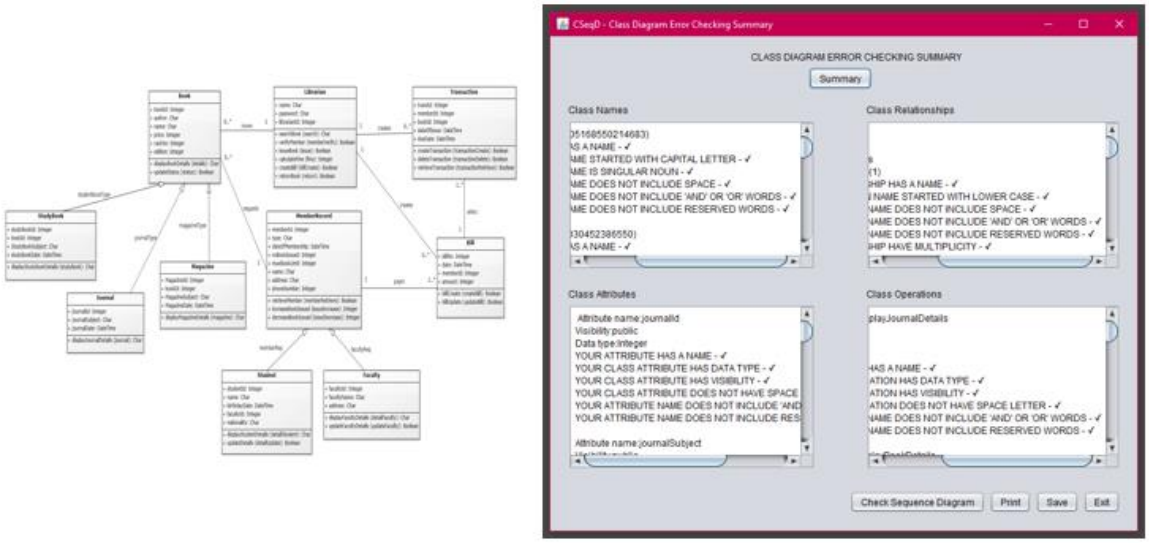


FIGURE 50: large scale Library scenario class diagram with correct notations and the checking result which shows all the checking notations as correct.

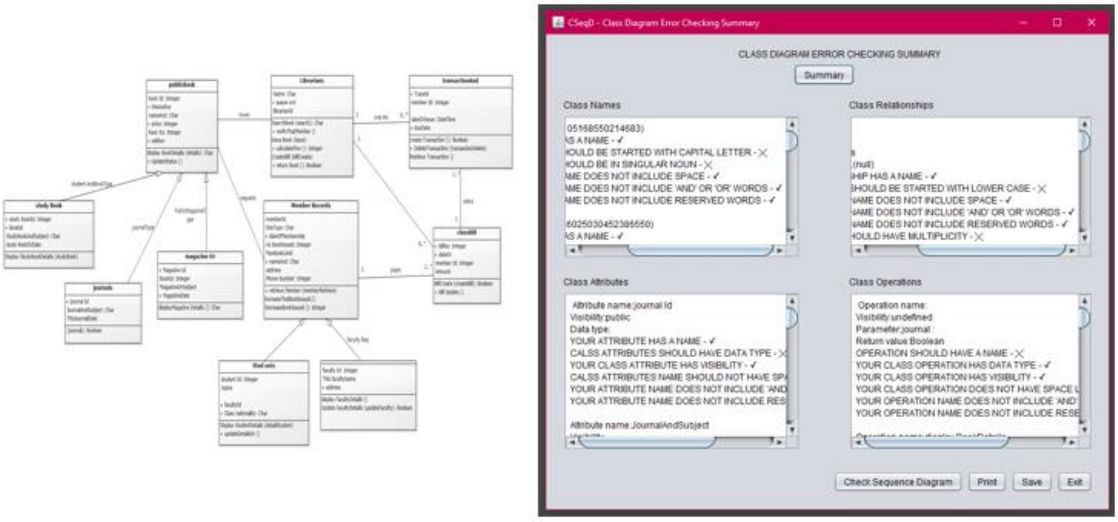


FIGURE 51: large scale Library scenario class diagram with some wrong notations and the checking result which shows the detected errors by the tool.

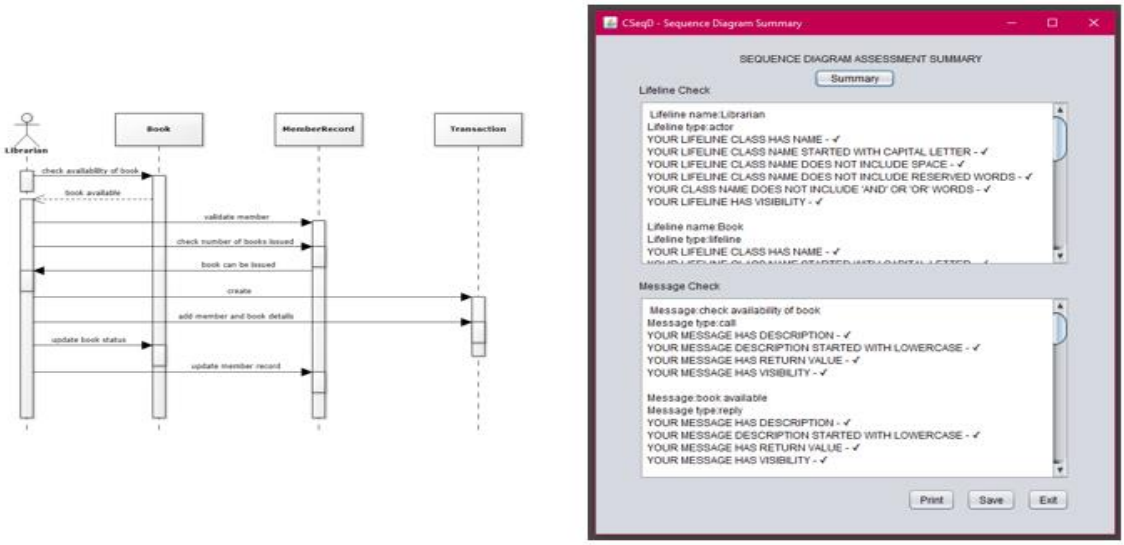


FIGURE 52: Library scenario (Issue Book) sequence diagram with correct notations and the checking result which shows all the checking notations as correct.

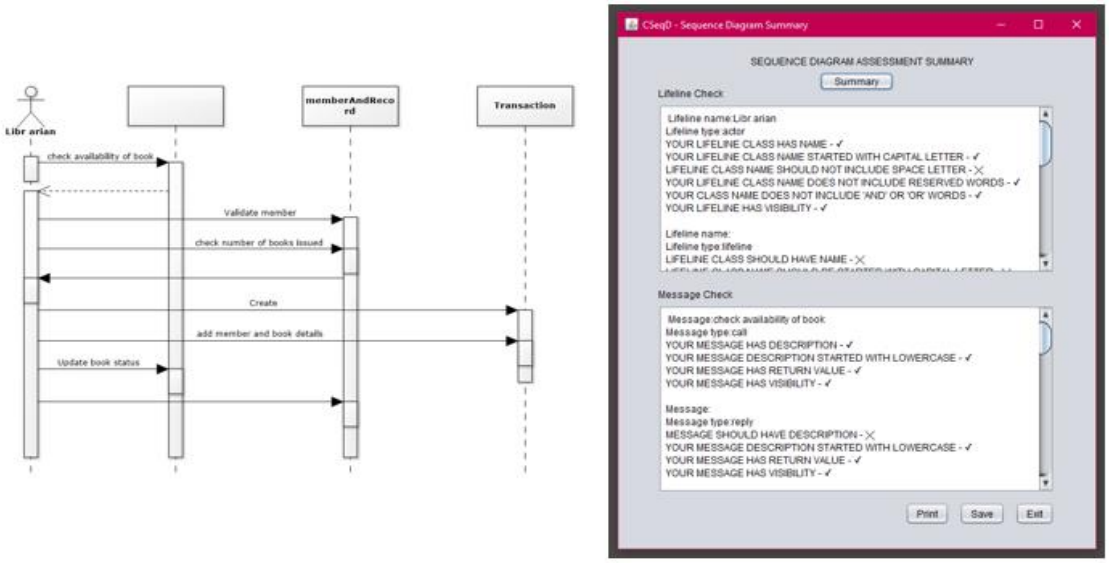


FIGURE 53: Library scenario (Issue Book) sequence diagram with some wrong notations and the checking result which shows the detected errors by the tool.

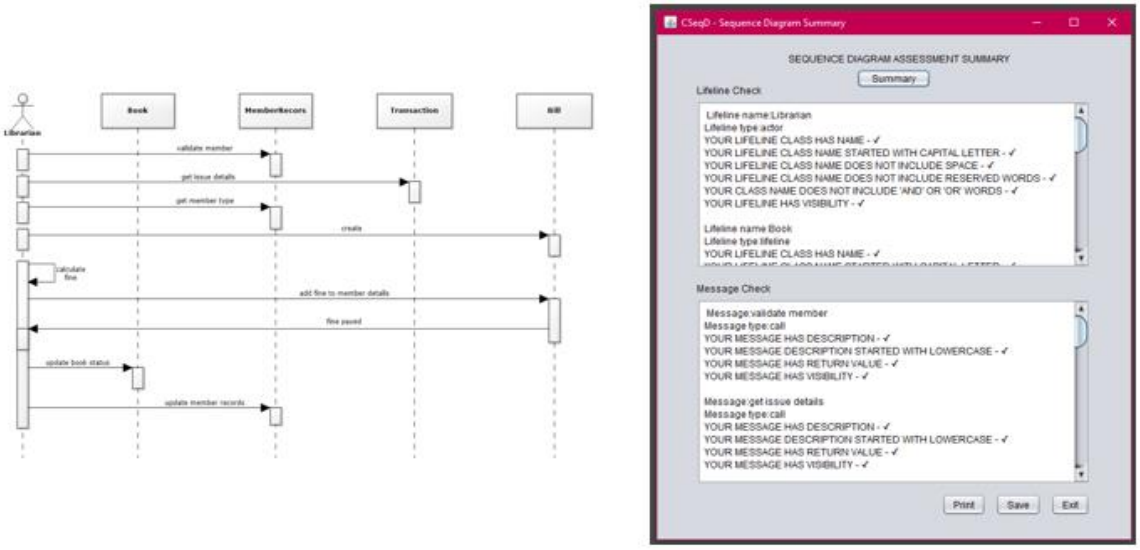


FIGURE 54: Library scenario (Return Book) sequence diagram with correct notations and the checking result which shows all the checking notations as correct.

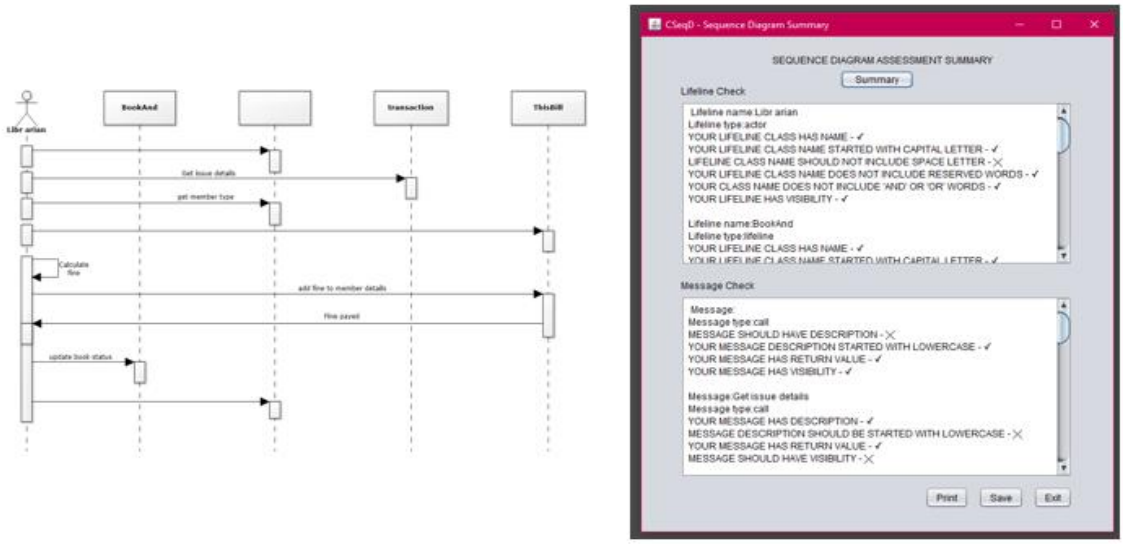


FIGURE 55: Library scenario (Return Book) sequence diagram with some wrong notations and the checking result which shows the detected errors by the tool.

According to the FIGURE 46 to 55, it can be seen that all the class naming, attributes, operations and relationships errors in class diagram and lifeline and message errors in sequence diagram were detected. It means that using this tool would be helpful for end-users who do not have comprehensive knowledge in UML diagrams and software design.

5.2.2 UNIT TESTING

Unit testing was conducted after the implementation phase of development process. This testing was conducted by using white-box testing where it was used test-cases to test on the IF-ELSE statement that it had used during the checking activity. The reason why this testing was used is because this testing is easy to handle if there are problems arise that related to the programming. It is because this test was conducted by using the test-cases and there are the expected results that it must obtain according to the test-cases. This testing was conducted to all the functionalities of the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD). Test-cases used for this testing are represented as follow:

Test Case for Uploading the UML class diagram XML file.

Purpose: to test the effective of the tool to upload the file.
Conditions:
1.The file uploaded is a ".simp" file.
2.There is a file that being uploaded into the tool.

Test Case ID	Condition 1	Condition 2	Expected Result
TC1	True	-	The file name will displayed in JtextField.
TC2	False	-	"The file uploaded should be in .simp format."
TC3	-	True	The file will displayed in JTextArea.
TC4	-	False	"There are no file uploaded."

Test Case for Checking class's name.

Purpose: to check the effective of class's name checking rules.
Conditions:
1.Class's name should not include space letter.
2.Class's name should not include programming reserved words.

Test Case ID	Condition 1	Condition 2	Expected Result
TC1	True	-	"YOUR CLASS NAME DOES NOT INCLUDE SPACE LETTER."
TC2	False	-	"CLASS NAME SHOULD NOT INCLUDE SPACE LETTER."
TC3	-	True	"YOUR CLASS NAME DOES NOT INCLUDE RESERVED WORDS."
TC4	-	False	"CLASS NAME SHOULD NOT INCLUDE RESERVED WORDS."

Test case for checking class's attributes.

Purpose: to check the effective of class's attributes checking rules.
Conditions:
1.Class's attributes should have data type.
2. Class's attributes should have name.

Test Case ID	Condition 1	Condition 2	Expected Result
TC1	True	-	"YOUR CLASS ATTRIBUTE HAS DATA TYPE."
TC2	False	-	"CALSS ATTRIBUTES SHOULD HAVE DATA TYPE."
TC3	-	True	"YOUR ATTRIBUTE HAS A NAME."
TC4	-	False	"ATTRIBUUTE SHOULD HAVE A NAME."

Test case for checking class's operations.

Purpose: to check the effective of class's operations checking rules.
Conditions:
1.Class's attributes should have visibility.
2. Class's operations name should not have space letter.

Test Case ID	Condition 1	Condition 2	Expected Result
TC1	True	-	"YOUR CLASS ATTRIBUTE HAS VISIBILITY."
TC2	False	-	"CALSS ATTRIBUTES SHOULD HAVE VISIBILITY."
TC3	-	True	"YOUR CLASS OPERATION DOES NOT HAVE SPACE LETTER."
TC4	-	False	"CALSS OPERATION NAME SHOULD NOT HAVE SPACE LETTER."

Test case for checking class's relationships.

Purpose: to check the effective of class's relationships checking rules.
Conditions:
1.Class's relationships should have multiplicity.
2.Class's relationships should have name.

Test Case ID	Condition 1	Condition 2	Expected Result
TC1	True	-	"YOUR CLASS OPERATION HAS MULTIPLICITY."
TC2	False	-	"CALSS OPERATION SHOULD HAVE MULTIPLICITY."
TC3	-	True	"YOUR RELATIONSHIP HAS A NAME."
TC4	-	False	"RELATIONSHIP SHOULD HAVE A NAME."

Test case for checking sequence's lifeline.

Purpose: to check the effective of sequence's lifeline checking rules.
Conditions:
1.lifeline should have visibility.
2.lifeline's class should have name.

Test Case ID	Condition 1	Condition 2	Expected Result
TC1	True	-	"YOUR LIFELINE HAS VISIBILITY."
TC2	False	-	"LIFELINE SHOULD HAVE VISIBILITY."
TC3	-	True	"YOUR LIFELINE CLASS HAS NAME."
TC4	-	False	"LIFELINE CLASS SHOULD HAVE NAME."

Test case for checking sequence's message.

Purpose: to check the effective of sequence's message checking rules.
Conditions:
1.message should have description.
2.mesage should have visibility.

Test Case ID	Condition 1	Condition 2	Expected Result
TC1	True	-	"YOUR MESSAGE HAS DESCRIPTION."
TC2	False	-	"MESSAGE SHOULD HAVE DESCRIPTION."
TC3	-	True	"YOUR MESSAGE HAS VISIBILITY."
TC4	-	False	"MESSAGE SHOULD HAVE VISIBILITY."

Test cases for save the error detection lists & print the error detection lists.

Purpose: to check the effective of save and print functionalities.
Conditions:
1.click the save button.
2.click the print button.

Test Case ID	Condition 1	Condition 2	Expected Result
TC1	True	-	The error detection lists are successfully saved in ".txt" format.
TC2		True	The error detection lists are successfully printed.

Even though the tool is not a complete tool, it still achieve the objective of the project which is to design a supporting tool that suitable for the educational purposes where it can be used by the end-users or the beginner in the software engineering fields since the evaluating and testing results state that the tool is quite useful to them. Besides that, based on the feedback given by the evaluating and testing result, it can lead to the improvement for the tool in the future.

5.3 DEPLOYMENT PHASE

The last phase of developing the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) is the deployment phase. This phase was conducted where the full-developed tool was deploy into another hardware (laptop or desktop), besides my own laptop in order to test either it can works properly or not. For the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) was developed in ".java" format. Therefore, in order to allow it can deploy into hardware easily, all of the source code files of this tool were compiled together and build into a ".jar" format. The reason why I decided to develop the tool in ".jar" instead of ".java" is that, the ".jar" format can allow the tool can be accessed easily without needs to open the NetBeans IDE tool. After building an executable ".jar" file from the whole project, by using an installer maker tool named as "Setup Factory" an executable file in ".exe" format was produced from (CseqD) ".jar" file that can be install and run on every platform.

CHAPTER 6 – CONCLUSION

Benefits and limitations of Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) will discussed in this chapter. Also, some suggestions that can contribute the improvement of the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) will come afterwards.

6.1 BENEFITS OF AUTOMATED ERROR CHECKING FOR UML CLASS AND SEQUENCE DIAGRAMS TOOL (CSeqD)

There are several benefits that the user of this Automated Error Detecting for UML Class and Sequence Diagrams Tool (CseqD) can take advantage of. One of the benefits is that tool is very useful for educational purposes where it is very beneficial for the end-users or the novice of the computer science fields because this tool is focusing in assessing the UML class and sequence diagrams that are very important in design phase of software development process. Moreover, this Automated Error Detecting for UML Class and Sequence Diagrams Tool (CseqD) also can be used to help the end-users to gain more understanding about the UML class and sequence diagrams since this able to shows the guidelines in designing class and sequence diagram. In addition, this tool also is easy to use and have a simple and friendly user interfaces.

6.2 LIMITATIONS OF AUTOMATED ERROR CHECKING FOR UML CLASS AND SEQUENCE DIAGRAMS TOOL (CSeqD)

However, this Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) still is not a perfect software tool as it still has some limitations that can give some difficulties to the users. One of the limitations of this tool is the error checking list that provided by this tool is listed textually. Also regarding the modeling tool, this Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) is developed for a specific modeling tool that is the Software Ideas Modeler which means the user only can use this assessment tool if they use the Software Ideas Modeler when designated the UML class and sequence diagram. It is the other limitation of the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD).

6.3 FUTURE WORKS FOR AUTOMATED ERROR CHECKING FOR UML CLASS AND SEQUENCE DIAGRAMS TOOL (CSeqD)

Thus, in order to overcome the limitations of the (CSeqD) there are several suggestions that can be used as the future works for this tool. One of the suggestions is that in addition to detect the errors of the class and sequence diagrams, the tool can be improved to check the consistency between these two diagram and check whether the features of both class and sequence diagrams are consistent with each other or not and gives feedback to the user. Another suggestion that can be used to improvise the

(CSeqD) is by upgrading the tool so that it can be assessing the class and sequence diagrams that has been designated with various modeling tool, not for specific modeling tool.

6.4 CONCLUSION

In conclusion, the Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) is quite beneficial to the end-users, students or the novice of the computer science fields since this tool is aiming for the educational purposes. The result of this project is to propose an automated error checking tool to detect defects in UML class and sequence diagrams. With this tool, the object-oriented models developed by end-users will improved by providing feedback on the diagram's correctness. Besides, common mistakes made by end-users and students regarding to their UML class and sequence diagrams will also be identified. The main contribution of this project is to help end-users and students to design a good and valid UML diagrams via the Automated Error Checking UML Class and Sequence Diagrams Tool (CSeqD).

REFERENCES

[1] Zaretska, I., Kulankhina, O. and Mykhailenko, H. 2013. Cross-Diagram UML Design Verification from *Springer-Verlag Berlin Heidelberg*.

[2] Herout, P., Brada, P. 2016. UML-test Application for Automated Validation of Students' UML Class Diagram from *2016 IEEE 29th International Conference on Software Engineering Education and Training*.

[3] Noda, K., Kobayashi, T. and Agusa, K. 2009. Sequence Diagram Slicing from *16th Asia-Pacific Software Engineering Conference*.

[4] Burnet, M., 2009. What Is End-User Software Engineering and Why Does It Matter? from *Springer-Verlag Berlin Heidelberg*.

[5] Akayama, S., Demuth, B. Lethbridge, T. C. Scholz, M. Stevens, P. and Stikkolorum, D. R. 2015. Tool Use in Modelling Education from *http://ceur-ws.org/Vol 1134/paper6*.

[6] Zaretska, I., Kulankhina, O. and Mykhailenko, H. 2013. Cross-Diagram UML Design Verification from *Springer-Verlag Berlin Heidelberg*.

[7] Brewer, J., and Lorenz, L. 2003. Using UML and Agile Development Methodologies to Teach Object Oriented Analysis & Design Tools and Techniques from *03 Proceedings of the 4th conference on Information technology curriculum* 54-57.

[8] Soler, J., Boada, I. Prades, F. Poch, J. and Fabregat, R. 2010. A web-based e-learning tool for UML class diagram from *IEEE Education Eng Conference* 973-979.

[9] Hasker, R. W., and Shi, Y. 2014. Teaching Basic Class Diagram Notation with UMLGrader from *121th ASEE Annual Conference & Exposition*.

[10] Hasker, R. W., Rosene, A. and Reid, J. 2012. Experiences with a UML diagram critique tool from *Midwest Instruction and Computing Symposium.*

[11] NClass. Retrieved on 12/09/2015 from http://nclass.sourceforge.net/

[12] Software Ideas Modeler. Retrieved on 17/09/2015.from https://www.softwareideas.net/

[13] Ramollari, E., and Dranidis, D. 2007. StudentUML: An Educational Tool Supporting Object-Oriented Analysis and Design from *11th Panhellenic Conference in Informatics.*

[14] Egyde, A. 2007. UML/Analyzer: A tool for the instant consistency checking of UML models from *Proceedings of the 29th International Conference on Software Engineering*.

[15] Welcome to ArgoUML. Retrieved on 17/09/2015 from http://argouml.tigris.org/

[16] Crahen, E., Alphonce, C. and Ventura, P. 2002. QuickUML: a beginner's UML tool from *OOPSLA '02 Companion of the 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*.

[17] Hasker, R. W., Rowe, M. 2001. UMLint: Identifying defects in UML diagrams from *118th ASEE Annual Conference & Exposition*.

[18] Ali, N. M., Hosking, J. and Grundy, J. 2013. A Taxonomy and Mapping of Computer-Based Critiquing Tools from *IEEE Transaction on Software Engineering,* Volume 39, Issue 11.

[19] Pazzani, M., and Brunk, C. 1991. Detecting and correcting errors in rule-based expert systems: an integration of empirical and explanation-based learning from *Knowledge Acquisition*, 3: 157-173.

[20] Elfaki, A. O., Fong, S. L. Vijayaprasad, P. Gapar Md Johar, M. D. and Fadhil, M. S. 2014. Using a Rule-based Method for Detecting Anomalies in Software Product Line from *Research Journal of Applied Sciences, Engineering and Technology*, 275-281.

[21] Raguuath, P. K., Velmourougan, S. Davachelvan, D. Kayalvizhi, S. and Ravimohan, R. 2010. Evolving A New Model (SDLC Model-2010) For Software Development Life Cycle (SDLC) from *IJCSNS International Journal of Computer Science and Network Security*, VOL.10 No.1.

[22] Brooke, J. 1996. SUS: A "quick and dirty" usability scale. In P. W. Jordan, B. Thomas, B. A. Weerdmeester, & A. L. McClelland (Eds.), *Usability Evaluation in Industry.* London: Taylor and Francis.

[23] Johazmi, N.N. 2016. *Uml Class Diagram Assessment Tool (UCDAT),* Bachelor Thesis, Universiti Putra Malaysia.

[24] Auer, M., Tschurtschenthaler, T., and Biffl, S. 2003. A flyweight UML modeling tool for software development in heterogeneous environments from *EUROMICRO'03*.

[25] Turner S. A., Perez-Quinones M. A., and Edwards S. H. 2005. minimUML: A minimalist approach to UML diagramming for early Computer Science education from *Computing Research Repository.*

APPENDIX 1:

THE USER MANUAL DOCUMENTATION FOR THE USERS

Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD)

INTRODUCTION

Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) is a supporting tool that had been developed in order to help you whose is the End-User or the UML designing tools. This tool can be used to assess the UML class and Sequence diagrams that had been designated by the End-Users using the modeling tool. This user manual documentation is created in order to show you the user on how to the interfaces of the tool so that you can get familiar with it and you can use it easily without having any difficulties.

PREREQUISITES

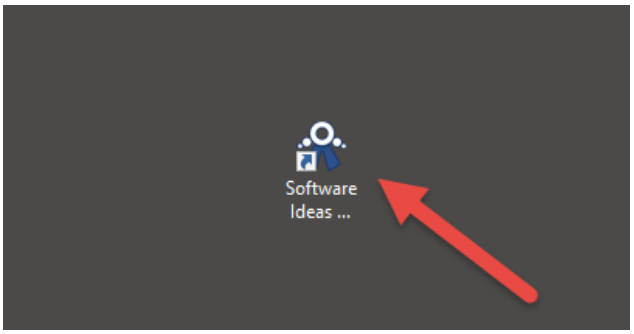
- This tool only can be used to assess the **UML class and sequence diagrams**.

- Make sure you are using **Software Ideas Modeler** modeling tool while designated your UML class and sequence diagrams.

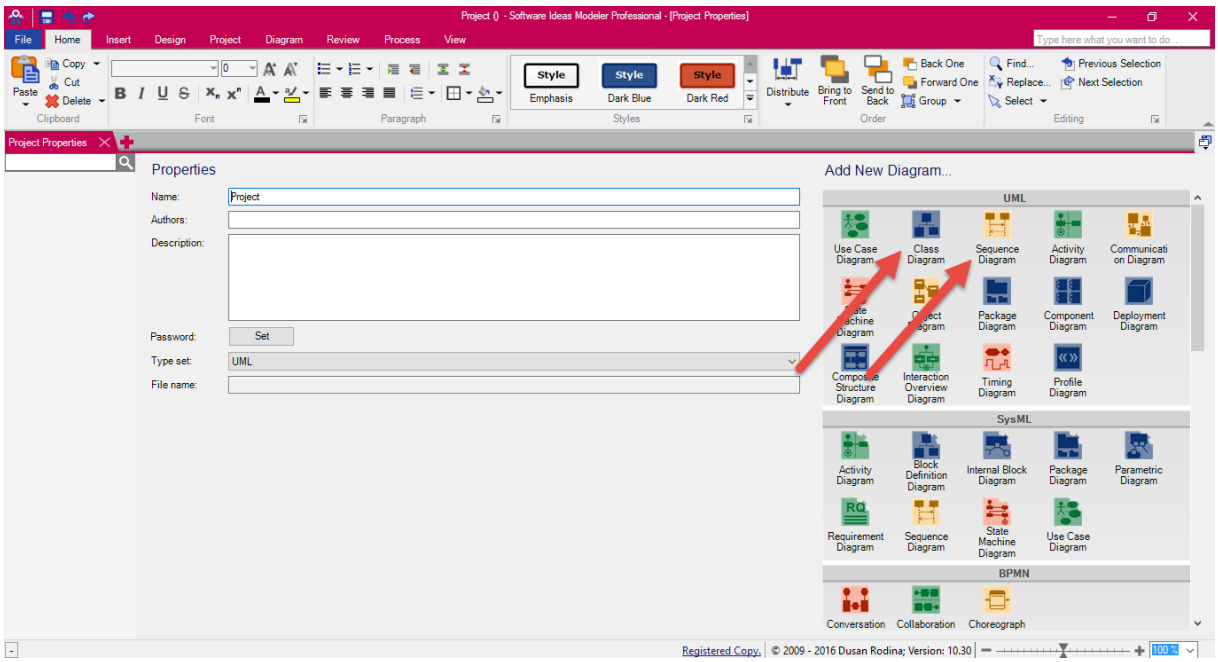
- Make sure the file that being saved from the modeling tool is in **".simp" format**.

HOW TO USE?

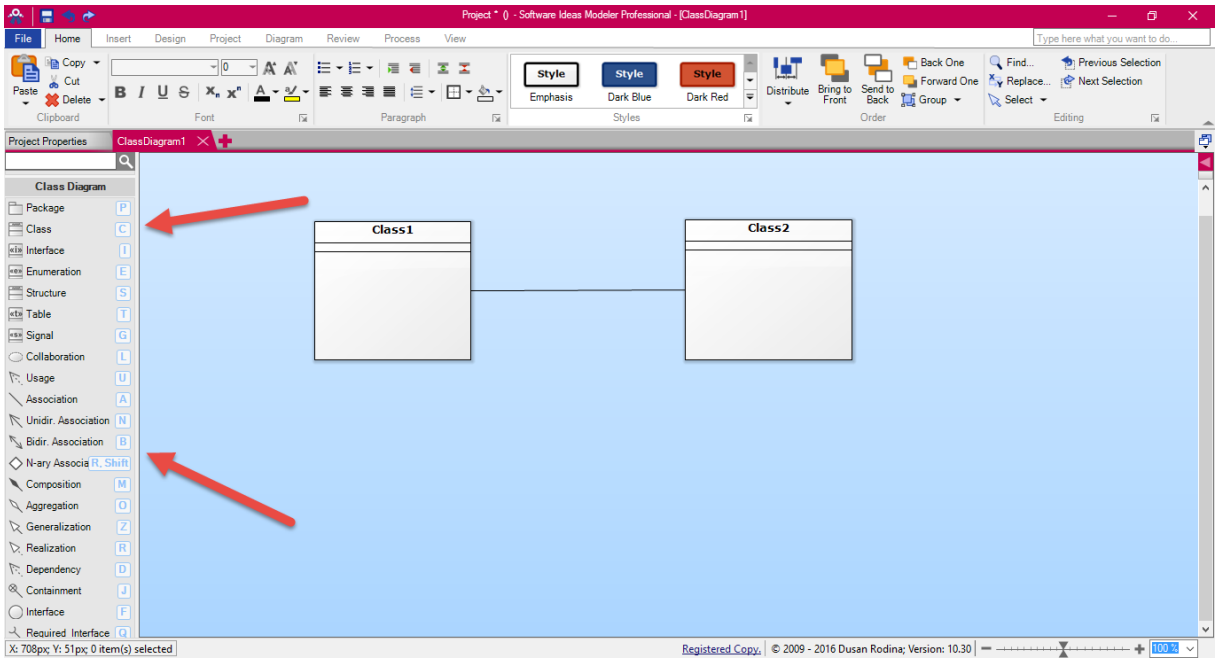
1- Firstly, you need to design your UML class or sequence diagrams by using Software Ideas Modeler modeling tool.



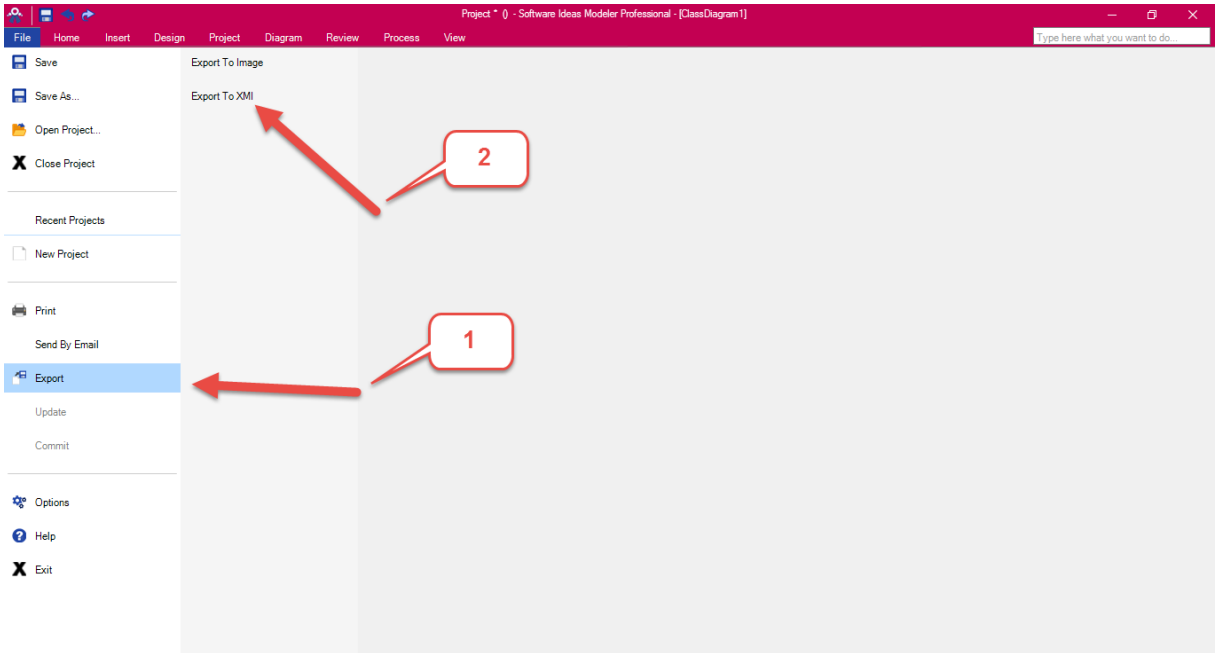
2- You should choose whether you want to draw a class diagram or a sequence diagram.



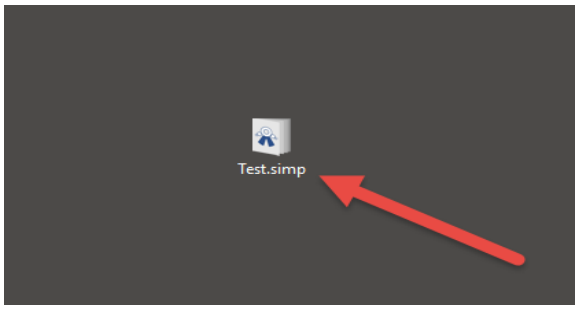
3- Your required tools in order to draw diagrams are available in the left side of the Software Ideas Modeler. You can use them by drag and drop.



4- After finishing your modeling you should export your model in XMI.

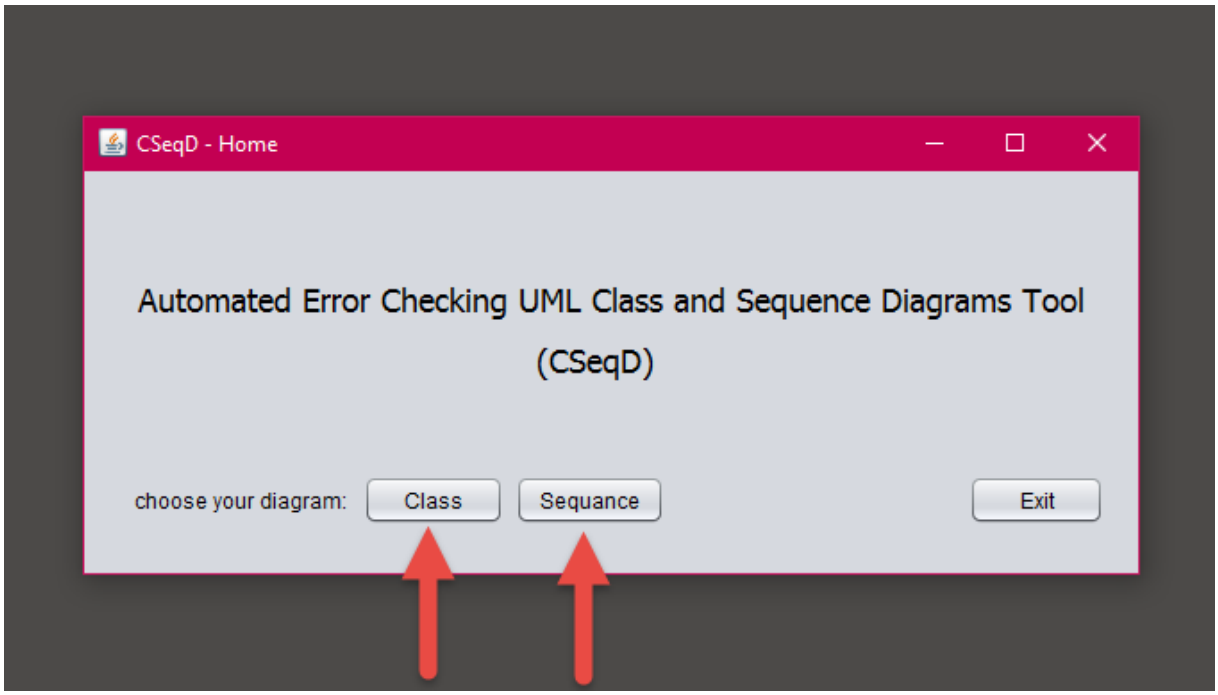


5- Make sure your file have been saved in (.simp) format.



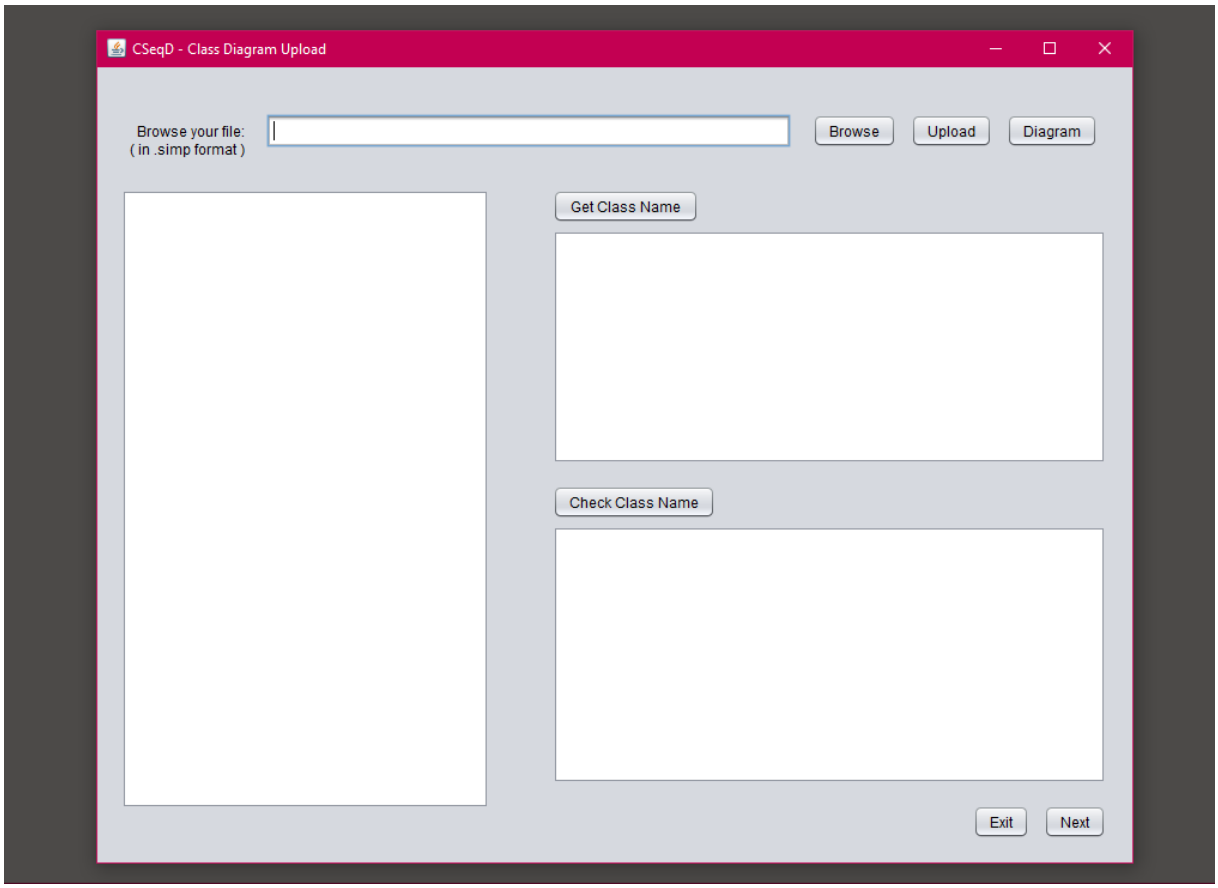
6- Now you can check your diagram correctness by importing your (.simp) file in to Automated Error Checking for UML Class and Sequence Diagrams (CSeqD).

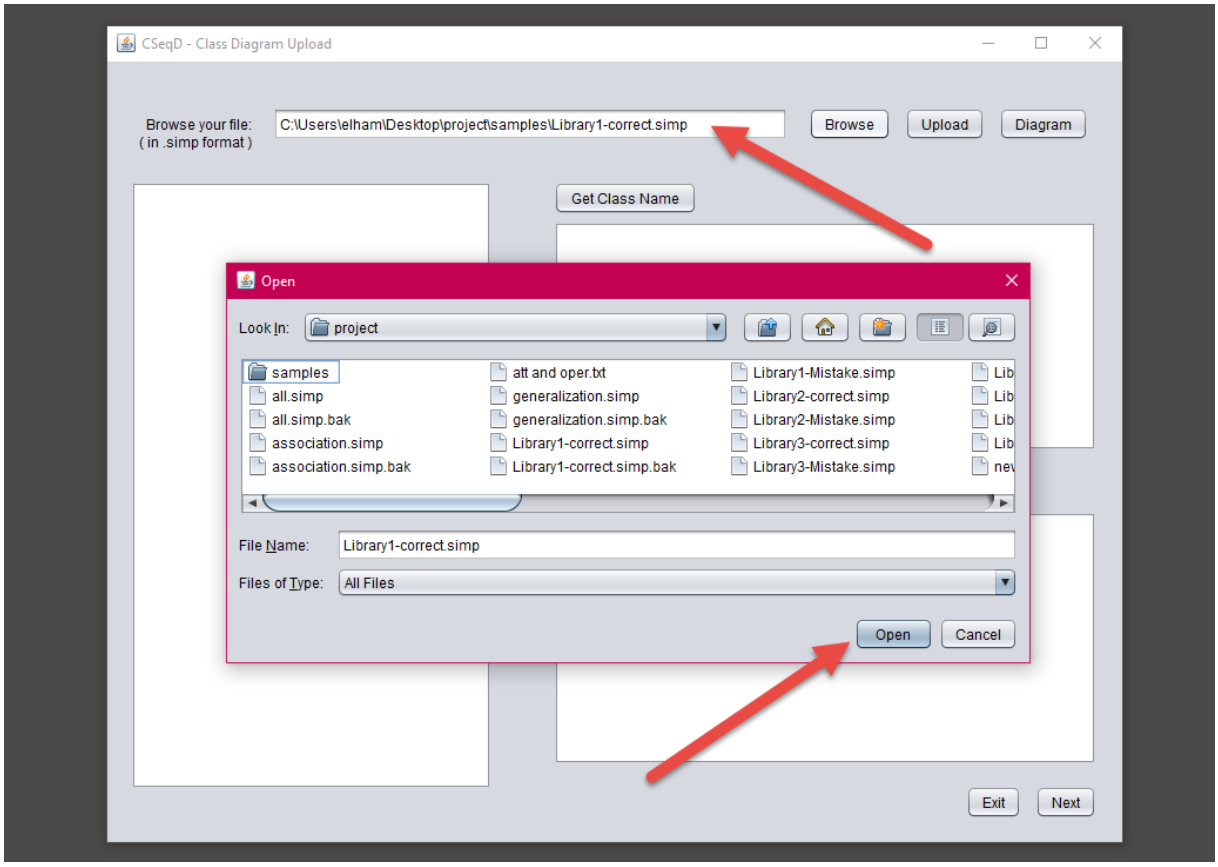
7- By clicking on the (CSeqD) icon, the home page of the tool, where you should choose if you want to assess class or sequence diagram, will appear.

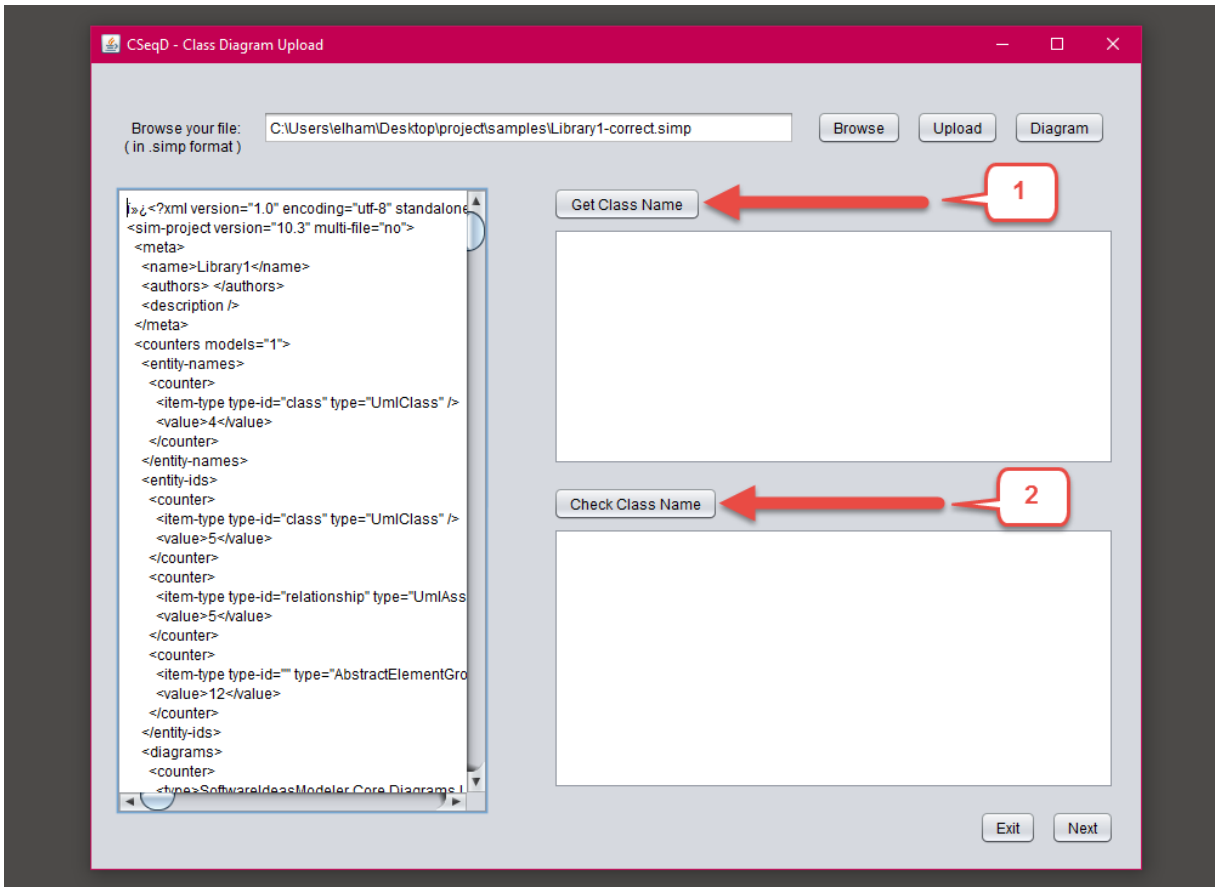


8- For example you choose class diagram, then you will see the following page. In this page you should Browse and find your (.simp) format file that you produced previously. Then, you should upload the file, then by clicking on the “Get Class Name” following by “Check Class Name” buttons, the list of your UML modelling classes and the checking result will appear. You can see the process by following pictures.

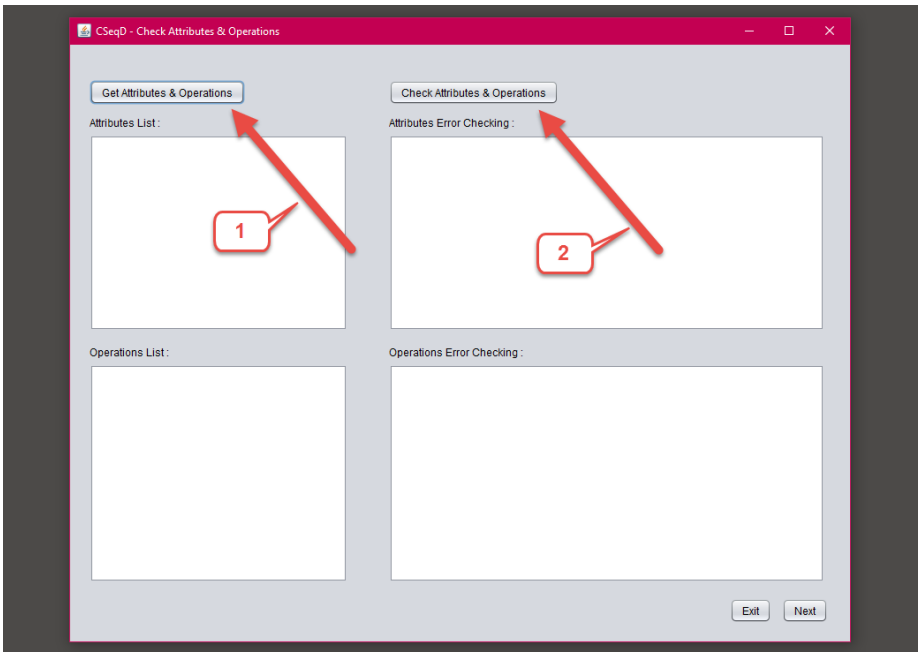
By clicking on the “Diagram” button you will see the picture of your design.


CSeqD - Class Diagram Upload
Browse your file:
(in .simp format)
Browse
Upload
Diagram
Get Class Name
Check Class Name
Exit
Next

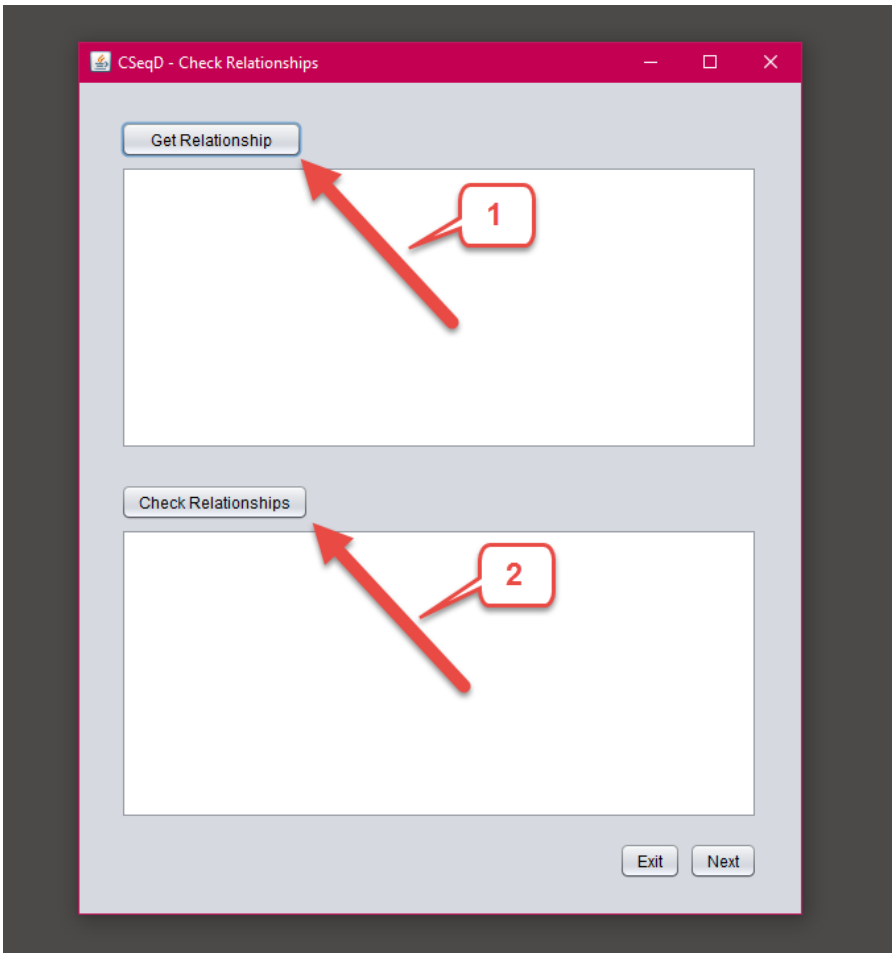

CSeqD - Class Diagram Upload
Browse your file:
(in .simp format)
C:\Users\elham\Desktop\project\samples\Library1-correct.simp
Browse
Upload
Diagram
Get Class Name
Open
Look In:
project
samples
all.simp
all.simp.bak
association.simp
association.simp.bak
att and oper.txt
generalization.simp
generalization.simp.bak
Library1-correct.simp
Library1-correct.simp.bak
Library1-Mistake.simp
Library2-correct.simp
Library2-Mistake.simp
Library3-correct.simp
Library3-Mistake.simp
File Name:
Library1-correct.simp
Files of Type:
All Files
Open
Cancel
Exit
Next



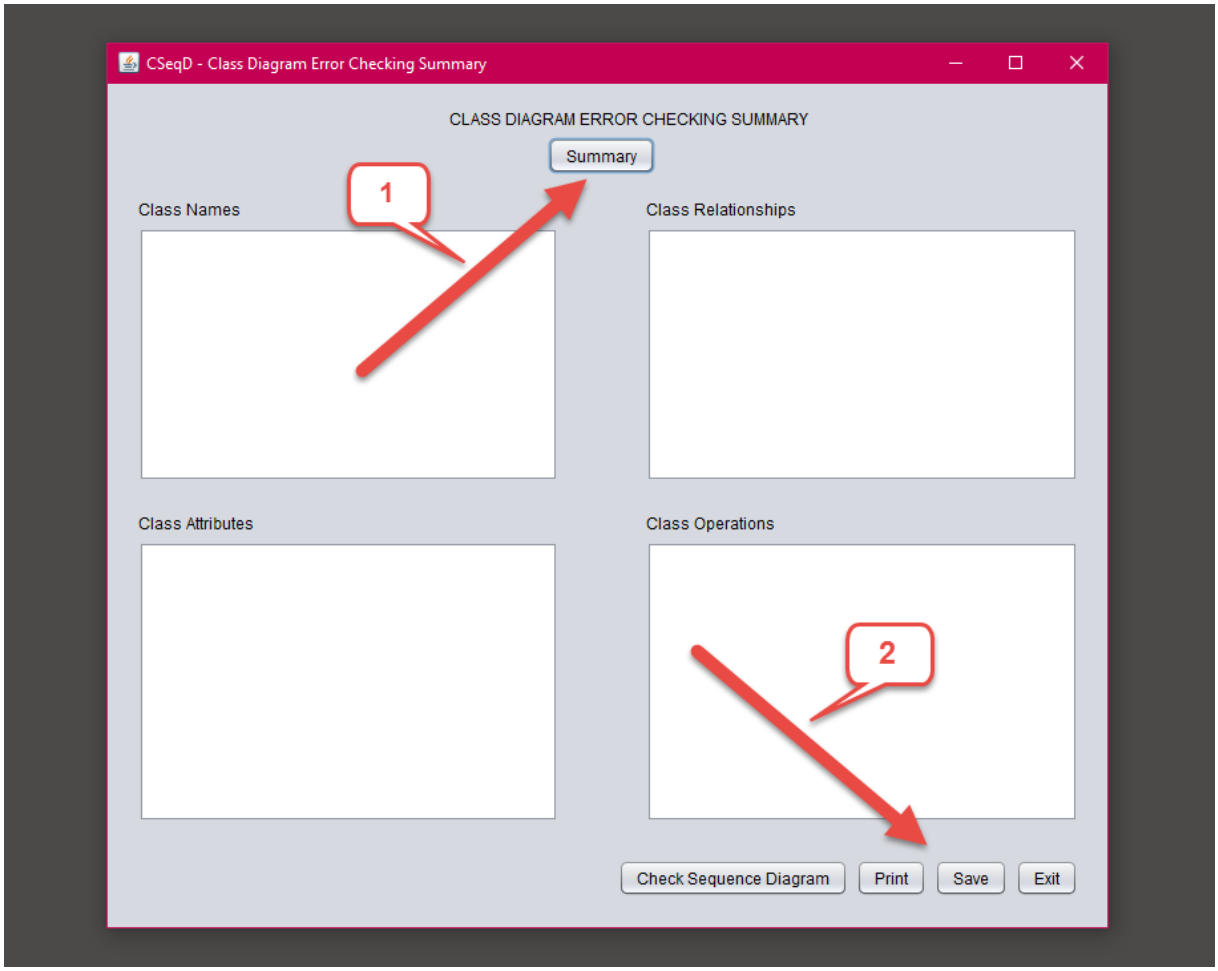
9- After checking your Class names, you can check the correctness of your attributes and operations by clicking on the “Next” button or exit the tool by clicking on the “Exit” button.



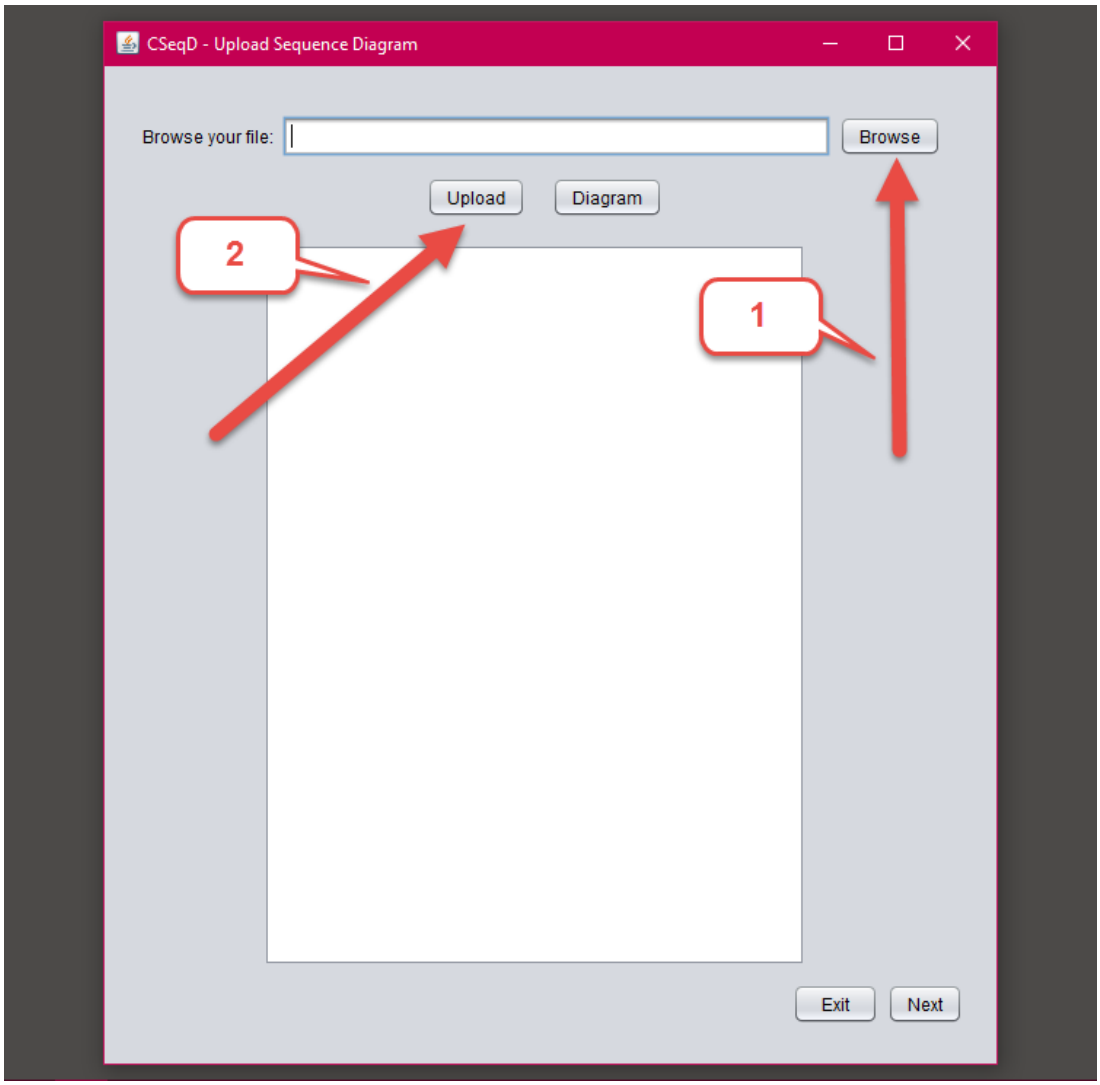
10-Like previous step, you can check you class diagram relationships.



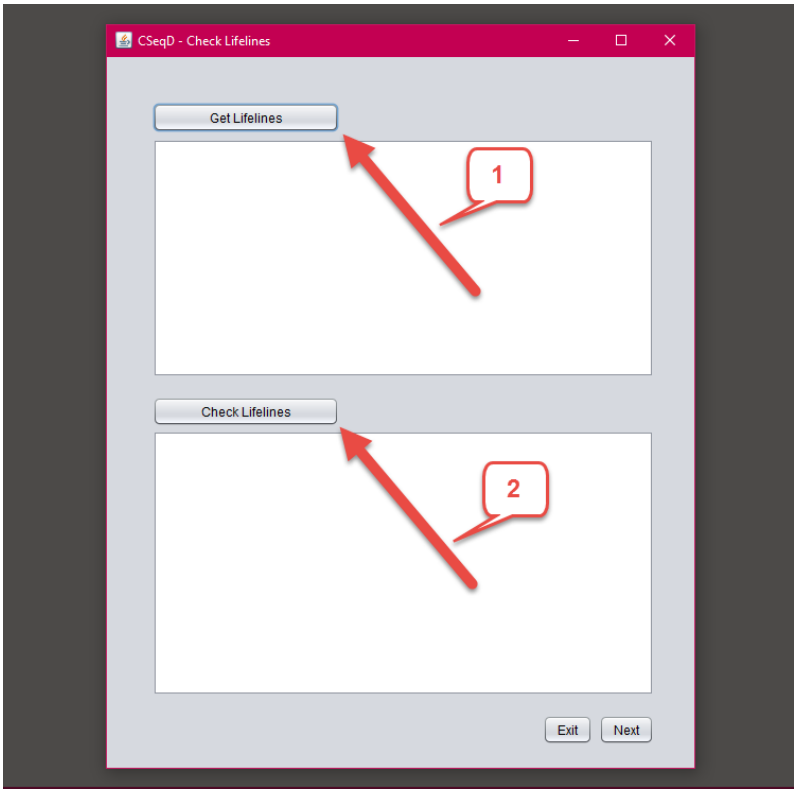
11- Right after checking your class diagram relationships, by clicking on the “Next” button, the summary of all your checking results will be available. In this page you can save or print the results and if you want to check the sequence diagram you can click on “ Check Sequence Diagram” button or exit the tool.



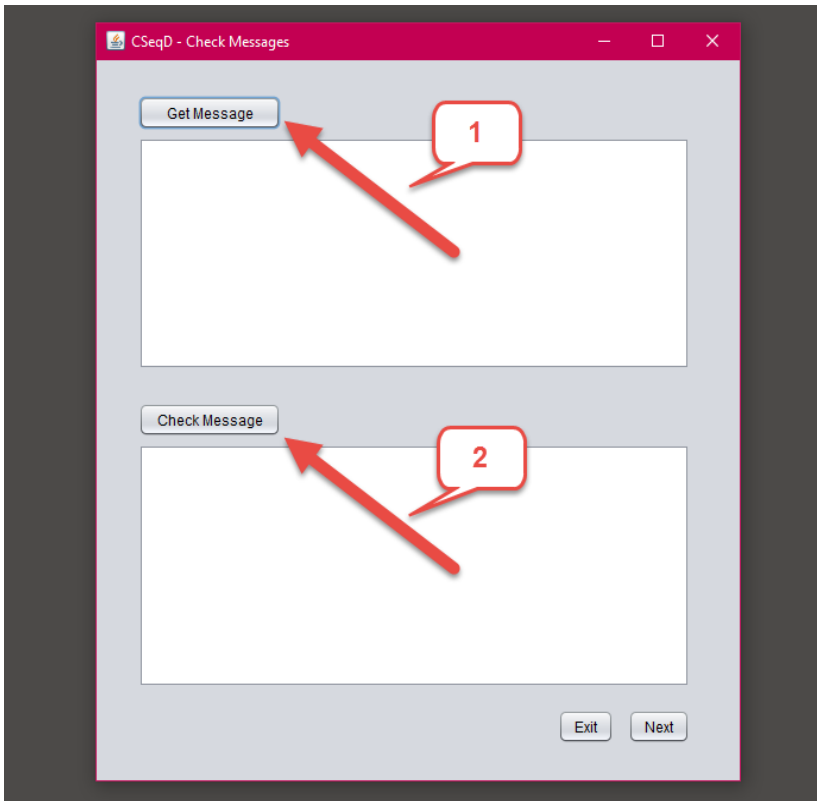
12- If you choose to check the sequence diagram, you should browse and upload your modeling (.simp) format file that you draw by Software Ideas Modeler.



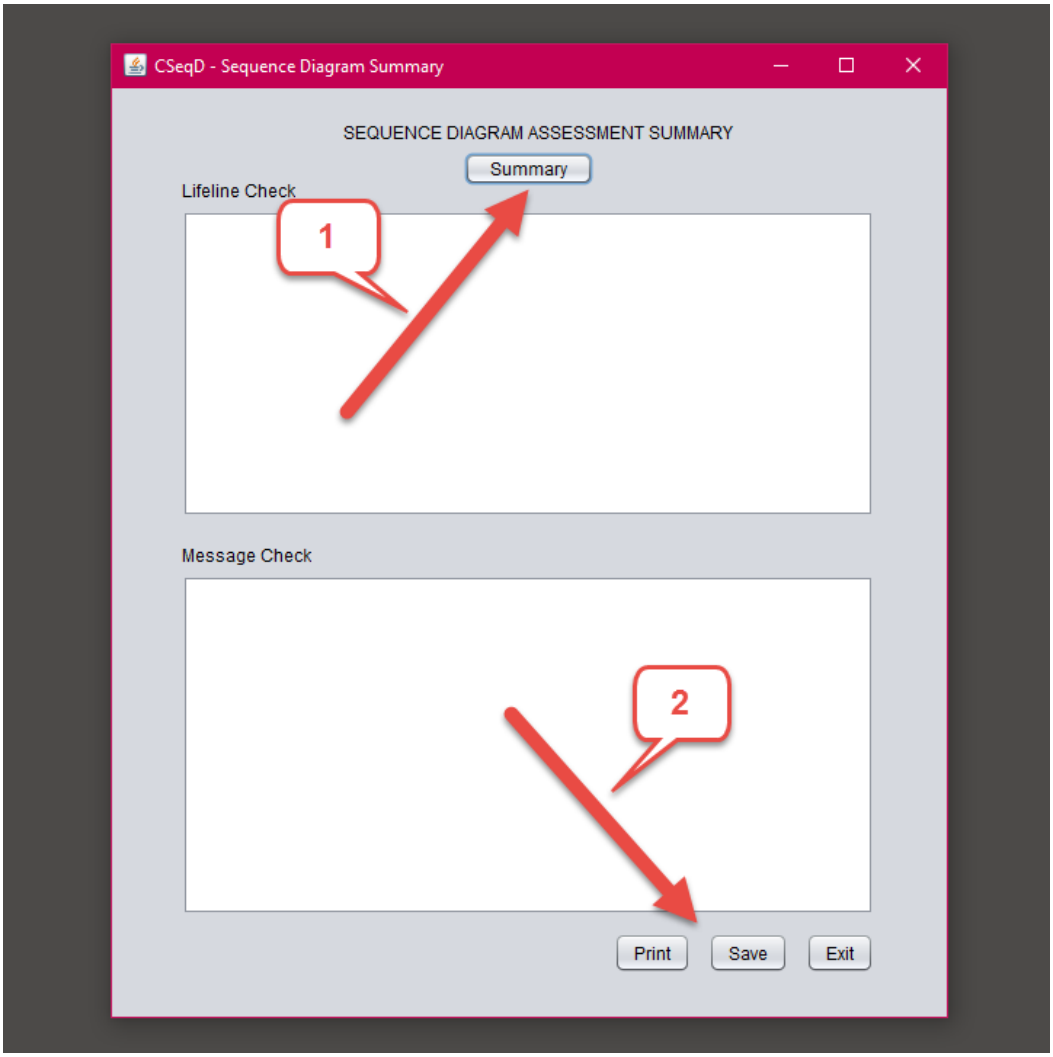
13- After uploading your file by clicking on the “Next” button you can check your sequence diagram lifeline correctness.



14- Next step is to detect the errors of your sequence diagram message.



15- Finally, by clicking on the “Next” button, the summary of your sequence diagram error checking will be available. you can save or print the results.



CONCLUSION

In conclusion, with user manual documentation provided, it hoped that this Automated Error Checking for UML Class and Sequence Diagrams Tool (CSeqD) is beneficial for all of its users and also the users are satisfy with all the functionalities that provided by this tool.