العنوان:	Automatic Generation of Descriptive Summary for Source Code Artifact
المؤلف الرئيسي:	البطوش، أماني عبدالسلام
مؤلفين آخرين:	حماد، مصطفى محمد(مشرف)
التاريخ الميلادي:	2016
موقع:	مؤتة
الصفحات:	1 - 86
رقم MD:	951089
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة مؤتة
الكلية:	عمادة الدراسات العليا
الدولة:	الاردن
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، برمجية جافا، برمجية سي
رابط:	https://search.mandumah.com/Record/951089

Chapter 1
Introduction

Overview

Keeping up with an enormous amount of source code that you need to read and understand and the lack of summary commits that are made by programmers, are the main challenges faced by today's developers. So in order to help developers deal with this problem and in order to reduce the cost, one solution is to use a simple text description, or simple graphical representation view of the source code features that developers can easily understand. This can also help developers to understand and validate changes, trace changes to other software artifacts, and locate and re (assign) bug reports.

In fact, automatic summarization is one of the oldest research areas dating back to the late 1950s, which is noted in all programming languages starting from FORTRAN that have provided a facility to write comments. However, in recent years there has been an increasing attention to this field from academia, government and industry. The reason is the rapid growth of accessible information resources, mostly the World Wide Web, which has resulted in a well-known problem of information overload (Mani, 1999). The need for automated source code summary represents a main source for system documentation and it is the core for source code understanding with respect to maintenance, development and reducing reuse cost.

Software systems are developed in a number of different phases. The first stage is the analysis of requirements followed by the design of the system in order to meet the requirements. The next step involves writing code in a programming language to implement the design specifications. Finally, the system is tested before it is released for use by an end user. Once the product has been shipped, the system enters a phase known as maintenance. Software maintenance is one of the most time and effort consuming. In software engineering, it means the modification of a software product after delivery to correct faults, in order to improve the performance or other features (Eddy, 2013). Developers during maintenance need quick understand to the source code entities such as (packages, classes or methods), since they cannot read the entire code of large systems. So the identifying will occur efficiently and then they just focus on the ones related to the task at hand. And since the most common two activities to deal with software systems are searching and browsing, the source code with thousands or millions lines of code, source code documentation becomes important.

Also, modifications source code documentation takes place, which are often documented with long messages. Those messages are a key

component of software maintenance; they can help developers locate and triage defects, validate changes, and understand modifications (Haiduc, 2010, & Haiduc S. J., 2010). In maintenance stage software change may occur, so it affects another part of the source code. This requires spending more effort and time from developers to find the affected lines of the source code in order to understand the software.

Software changes are the basic and essential building blocks and characteristic of software evolution in software development since the software systems must respond to evolving platforms, requirements, and other environmental pressures, and after the first version has shipped the software continues to evolve, software evolution offers a different point of view on the traditional about software maintenance: it indicates the idea of essential change within an environment (Godfrey, 2008). Software evolution appeared as an unexpected and unplanned phenomenon that was observed in the original case study, in the evolution step, developers add new features, correct previous mistakes and misunderstandings, and react to the requirements, technologies, and knowledge volatility as it plays out through time. And each change introduces a new feature or some other new properties into software. During evolution, the programmers must comprehend the existing program to be able to add new functionalities or new properties to it (Rajlich, 2014).

In software development, similar problems are solved again and again, so the best career is not to repeat solving of what has been already solved. The best solution here is to reuse the same solution. Software reuse is the use of software knowledge or the existing software in order to build new software. It is also means the reuse of the code (Frakes, 2005). The importance of software reuse comes because the need to reduce effort in software maintenance and development. It also improves the quality of software and decreases time to market (Poulin, 1993). So a good software reuse process facilitates the increase of productivity, reliability, quality, and the decrease of costs and implementation Time. Software systems and components are specific reusable entities, mathematical function or an object class. According to (Selby, 2005) found that a set of programs consist of 32% reused code (not including libraries), so in order to reuse the existing software it is important to understand and document source code.

Software comprehension is the main activity that simplifies maintenance, reuse, code understanding and many other activities in software engineering. (Storey, 2005), so the summary can be one of the techniques that simplify software comprehension , which produce a text that contains a large amount of the information, contained in the original text, and do not exceed half of the original text. Program-comprehension can be categorized into three models: top-down models, bottom-up models, and integrated models. Comprehension according to the top-down model is

working on deriving and formulating hypotheses about program purpose while ignoring details, in order to evaluate them by the developers. Bottom-up comprehension describes how a program is understood when a programmer doesn't have a knowledge about a program's domain, here the programmer checks the statements of a program and groups them into semantic chunks. This then can be combined further until the developer has an understanding of the general purpose of a program. The third model is the integrated models combine top-down and bottom-up program comprehension.

The developer typically uses top-down comprehension ever possible. If a programmer has some knowledge about the domain, he/she will start with top-down comprehension. When he encounters code fragments he/she cannot explain using his domain knowledge, he/she will switch to the bottom-up comprehension (Feigenspan, 2011). A better code understanding by programmers and what is most efficient and effective can lead to many kinds of improvements such as better tools, better maintenance processes and guidelines, and documentation that support the cognitive process.

Static analysis is one of the most important areas that focus on understanding the source code; it has the ability to analyze large amounts of source code in considerably shorter amount of time than a human could. Static analysis aims to statically test the text of a program, without attempting to execute it; static analysis tools generate a first pass of the code base and highlight areas that require more attention from a senior developer.

Software metrics are one of the important aspects of software engineering. Which acts as an indicator for software attribute. It also plays an important role in the management of software projects. Software metric is defined in the IEEE 1061 standard as a function that has an input software data, and the output from these data is a single numerical value, that can be explained as the degree to which software possesses a given attribute that affects its quality. The goal is gaining objective, quantifiable measurements and reproducible, which may have valuable applications in budget planning, cost estimation, software debugging, quality assurance testing, and optimizing personnel task assignments, so analyzing software metric provide another way to understand the software from the produced numerical value.

1.1 Source Code Summarization:

Source code, is a description of a computer program which can be textual, readable, human readable, static, and fully executable that can be compiled automatically into an executable form (Binkley, 2007). Source code also can be defined as a mixed artifact that contains information that enables the communication between the developers and the compiler. So

the Automatic Source code summarization is the process of producing an illustrative subset of the data, with a computer program that contains an information of the entire source code. So in order to summarize the source code there is a need to understand the source code.

When any software product has been developed, not only the executable file or the source code is developed, but also a different kind of documents are developed as a part of software engineering process such as software requirement document, design document, test document, etc. Good documents are very useful and they serve many purposes. The documents that are produced in order to understand the source code may be included within the source code, so here the software or the source code have an Internal Documentation, or included outside the source code which is called external documentation, where programmers keep their notes and explanations in a separate document. For software developers, external documentation is useful as it consists of information that describes the problems with the program in order to solve them, or it can also focus on documenting general description of the software code without being concerned with its detail written. The main aim from external documentation is to provide easy views for software code.

The Internal documentation which is explained by comments, these block of comment for the Java and C/C++ programming language, can be categorized in the following seven different types (Steidl, 2013):

1- Copyright comments: this type of comments is usually found at the beginning of each file, it includes information about the license or the copyright of the source code file.
2- Header comments: In Java, headers they found after the imports but before the class declaration, it gives an overview about the functionality of the class and provides information about, e. g., the class author, the peer review status, or the revision number.
3- Member comments: they provide information for projects and for API the developer. It describes the functionality of a method, being located either before or in the same line as the member definition.
4- Inline comments: describe implementation decisions used within a method body.
5- Section comments address several methods/fields together belonging to the same functional aspect.
6- Code comments: this kind of comments is temporarily commented for potential later reuse or debugging purposes.
7- Task comments: are developer notes containing a remaining to do, a remark about an implementation hack, or a note about a bug that needs to be fixed.

1.2 Aims and the importance of this study

Software comprehension is an important field in the software engineering; it is the core for many other activities such as reuse, maintenance, development, and software changes. This requires software engineers to spend a lot of time and effort to analyze and understand the software. So summarizing software artifacts is the best solution that helps the developer, maintainer, or any other one who aims to understand the software.

Many of the previous researchers focus on summarizing source code artifacts. So the commit produced from summarizing the source code just provides summary information about part of the software, and doesn't cover the overall software. They either provide a summary that describes the context of the artifact or they describe the semantic behind the class or the method, by analyzing the stereotype.

From here the importance of this research comes, so it aims to give a number of external descriptive views that summarize all the granularity levels of the software (i.e.: method, class and the package) by providing a general description that describes a quantity information for each artifact in the software, and more detailed description that provide semantic information that the syntax of each artifact holds for the selected artifact, which are presented as a set of reports , also the method control flow graph that views the method with some metrics that aim to measure the method, and the class call graph which is also supported with the main class metrics that measures the class quality.

1.3 Thesis claims

This thesis aims to introduce the proposed approach as a substitute for many other approaches, since it has been used to provide a good comprehension and understanding to the software engineers in order to help them in many areas. So it will be easy to develop, maintain, reuse, and analyze the software by reducing effort and time.

1.4 Contribution of this research

Although there are different ways introduced to understand the software, automatic program comprehension is the most efficient and wanted way. Internal and external documentation help during program understanding and it is also still an important research area.

This research proposes a new approach which aims to summarize the software system by analyzing the source code statically, in order to determine its elements to understand the relations between those elements, by generating a descriptive summary for the target software project.

Since source code contains a lot of text so we parse source code to xml tags using srcML (source code Markup Language) tool in order to analyze the source code, because it adds much of the syntactic information that is found in the Abstract Syntax Tree (Maletic, 2002). It also combines text with both structural and textual information of the source code and provide an easy way to extract information from the source code (Collard, 2002). All this makes the software comprehension directly supported, the main contributions from the proposed work are summarized in the following points:

- The target software artifacts are packages, classes and methods. And the generated summaries are hydride of texts, graphs, and numerical measures.
- The proposed methodology introduces a new approach that aims to generate the both class call graph and method control flow graph that represents a view for both class and method.
- There are also some other important contributions which aim at answering the following research questions:
- Does the generated descriptive summary summarizes, describes, and identifies the source code artifacts (package, class and method) automatically?
- Does the generated descriptive summary reflect the developers understanding of the software?

1.5 Thesis Structure

The remainder of this thesis is organized as follows: Chapter 2 discusses the summarization overview and reviews most of the work in the field of source code artifact summarization techniques and the source code comprehension techniques. This chapter consists of reviewing the methodology of each work, and the results that were achieved and any open source case study that was used. The source code artifacts descriptive summary is discussed in Chapter 3 detail how the new method works, with a number of examples for the generated descriptive summary. Chapter 4 provides the results of the new methodology, conclusion with a summary of the research conducted and recommendations for future work.

العنوان:	Automatic Generation of Descriptive Summary for Source Code Artifact
المؤلف الرئيسي:	البطوش، أماني عبدالسلام
مؤلفين آخرين:	حماد، مصطفى محمد(مشرف)
التاريخ الميلادي:	2016
موقع:	مؤتة
الصفحات:	1 - 86
رقم MD:	951089
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة مؤتة
الكلية:	عمادة الدراسات العليا
الدولة:	الاردن
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، برمجية جافا، برمجية سي
رابط:	https://search.mandumah.com/Record/951089

Chapter 2
Review of Literature

Introduction

In order to understand the source code many approaches were introduced by following many methods and techniques. In general programmer makes a mental map of the code by looking at and recognizing various knowledge structures by including specific domain knowledge as well as recognized structures in the source code. For example, when programmer wants to understand a while loop in some code he will look for the end of the loop, the condition to exit the loop and how the condition is changed, since that is how while loops are structured in general. Then the mental map of the code used to predict what will happen next in the code.

As a starting point, simply the summary can be defined as producing a text from one or more texts or list of sentences produced from one or more documents that presents the main points in a concise form, which contains a significant portion of the information in the original text(s), where it is not longer than half of the original text(s). When this is done by the means of a computer or automatically, it will be called Automatic Text Summarization (Lloret, 2008).

2.1 Source Code Artifact

Correia in his work (de Figueiredo Correia, 2015), defines the software artifacts as both the products of software development and the things that developers work with. They may be themselves part of the final set of deliverables to be built; they may describe or support the process of developing software, and how it unfolds; and they are capable of describing the function and design of software, and therefore be used in the creation of other software artifacts.

Also (Juergens, 2011), defines software artifact as a file that is created and maintained during the life cycle of a software system. It is part of the system or captures knowledge about it. Examples include requirements specifications, models and source code. From the point of view of analysis, an artifact is regarded as a collection of atomic units. For natural language texts, these units can be words or sentences, for source code tokens or statements. For data-flow models such as Matlab/Simulink, atomic units are basic model blocks such as addition or multiplication blocks.

While (Fisher, 2009), defines the software artifact as something produced during the software development process. The ultimate goal of the process is to produce an operational program that satisfies user's needs. From an end user's perspective, this working program is the artifact of primary interest. Customers also need documentation artifacts that tell them how to use the software. This documentation can include users' manuals,

tutorials, and online program help. The major software engineering artifact is source code.

One of the active research topics in software maintenance is summarizing software artifacts. It can said that the artifacts of a software system includes sotware code and executable files, they also include a hirarical diagrams of the software such as UML class diagram. Table 1, summarizes the kinds of languages and notations that can be used for different software artifacts.

Table 1:
Languages or notations used for software artifacts

Software Artifact	Language or Notation
Requirements	English and pictures, in electronic or paper form.
Specification	A formal specification language, such as SpecL where SpecL manages the logic and date complexity, reporting ambiguities to the user, or by applying a modeling notation, such as UML.
Design	A structured software documentation format, such as Javadoc, or a modeling notation, such as UML
Implementation	A programming language, such as Java or C++, and the graphical program diagramming notation also can apply.

According to Table 1, each phase of software engineering is defined as an artifact, and for each phase it is possible to represent it as graphical notation, structured documents, pictures, or programming language such as C++ or java.

2.2 Software Comprehension

There are different methods to deal with the source code artifacts in order to understand it.

- Some of them deal with the source code as a text since it contain a natural language to introduce a document from it, other methods, to introduce a document deal with the source code as fragments were they invistigate the artifacts from them.
- Some methods aim to find some features.
- On the other hand some comprehension techniques aim to provide quantities that aim to measure the software quality.

Program comprehension is popular area, the idea in this area summarize in breaking a large program into more manageable slices or smaller parts . So, instead of trying to comprehend the program as a whole, the programmer can try to comprehend these slices. were a slice is a set of

statements related by data and control flow. This way can be performed programmatically and can be useful for debugging of computer programs and during program comprehension (O'brien, 2003). Table 2, provides a prife describtion to the methods proposed to comprehence the sotware:

Table 2
Software comprehension

Method	Description
Visualization techniques	Visualization techniques aims to visualize the application using graphs, uml diagram or views (Pierre Caserta, 2011).
Metrics trace techniques	Metrics are useful for analysis purposes, it aims to measure the software project in order to determine the complexity, software size and the qulity of the source code (Sneed, 2006).
Quering techniques	Quering techniques provide a mechanisim to extract the progrm artifacts and the relationship between them. This will help in visualization or take query results as an input for further queries and analyses (Gˆırba, 2008).
Text retrieval techniques	It is designed to work with the documents that are written in natural language, and since source code contain natural language, it can be easily applied (McBurney, 2014).
Heuristic based techniques	This kind of techniques employed for learning or solving problem solutions which are good enough for a given set of data or conditions. It generates a light abstractive summary to the extracted information from the source code (Nazara, 2015).
Dynamic analysis techniques	It means the analysis of data gathered from a running program, it exposes the system's actual behavior so provide an accurate picture of a software system. This technique comprises the analysis of a system's execution through interpretation (for example using the Virtual Machine in Java) (Hamou-Lhadj, 2009).
Static analysis techniques	Is the analysis of computer software without performing the actual execution of the programs built from that software, it is usually applied to the analysis performed with human analysis and by using automated software tool (Gomes, 2009).
Fact collection	According to this technique developers working in the source code in order to search, learn, review, implement and propose facts about the source code in order to serve numerous roles, such as predicting the amount of

	additional investigation necessary to make a change, constraining changes, and suggesting changes (LaToza, 2007).
Feature locating techniques	In order to fix a problem or add an enhancement to the program, there is a need to locate the code that implements a specific feature of a program (Wilde, 2003).
Data mining analysis techniques	Applying mining techniques to the source code in order to get useful knowledge about the design of large legacy systems. Association rules , or clustring techniques can be easily applied (Kanellopoulos, 2004), (Tjortjis, 2003).
Source to source summary techniques	In this technique it is look at source code artifact as code fragment, so it aims to investigate these artifacts in order to produce a text summary to these artifacts (Nazara, 2015).

As shown in (Table2), understanding source code occupied a wide area of research in software engineering. And many techniques proposed to reach this aim, some of these techniques produce a text summary and some of them produce a graph or a view, also some of the proposed techniques are depending on each other for example the query techniques can help in the visualization.

Measured comprehension, is popular area of software comprehension research. In this area of software comprehension, research uses graph theoretic software models and some software metrics to measure the comprehend ability of programs. One of the best known graph theoretic metrics which is used is McCabe's Cyclomatic complexity. This metric can be used to devise a methodology for structured testing, when a programs have a higher cyclomatic complexity number this means it should be more difficult to understand, because they have more control flow branches.

2.3 Source Code Summarization

Several researchers have attempted to reduce the difficulty of software maintenance, and there are a lot of tools that were built in order to help in program understanding. Some studies focus on analyzing source code artifacts statically to introduce a reports as English sentences, or to generate a view, the generated report, or view summarize one artifact of the source code .

Some of the proposed generating summaries aims to summarize the methods, where (Sridhara G. e., 2010), presents a novel technique to generate a descriptive comments automatically in order to summarize Java methods intent. This technique focuses on producing comments that should include the important statements in the code. Given the signature and body

of a method, the automatic generator for the comments identifies the content for the summary and generates natural language text that summarizes the method's overall actions. To identify linguistic elements of the method, this approach applies the Software Word Usage Model. Accuracy, content adequacy and conciseness where evaluated. It gives a quick understanding of what a method does.

Also (McBurney, 2014) proposes a tool that generates documents which summarize the context surrounding a method (the environment in which the method is invoked), rather than details from the internals of the method. This work considered as a complement work to (Sridhara G. e., 2010), in order to improve it, so the focus here on describing the behavior of a Java method. The output is a set of English sentences describing why the method exists in the program, how to use the method and what the methods do internally. In this research also the tool performs a case study with 12 Java programmers as participants where they show that this work have more contextual information than the previous work. The author in (Abid, 2015) proposes an automatic approach that generates natural language documentation summaries for C++ methods depending on methods stereotypes and by applying the static analysis and fact extraction on the method, which at end added as a comment for each method.

The work (Alimucaj, 2009), aims to introduce a view that represent the method as a control flow graph; this approach depends on the AST.

On the other hand (Moreno, 2013), focus on summarizing the class in order to automatically generate human readable summaries for Java classes; these summaries allow developers to understand the main goal and structure of the class. By conjunction the determined class and method stereotypes with a set of heuristics. To identify which methods are to be included in the summary, two filters are applied. Stereotype based filter, which removes the methods whose stereotypes are not relevant to the class stereotype according to its definition, and access level filter, which based on the access level permitted by the modifiers of the methods. After that the selected information included in the summaries, were the generated summary use the existing lexicalization tools. This tool use ArgoUML and aTunes 1.6.0 java open source system in evaluation by selecting 20 classes per each system. This work evaluates the following properties to the generated summaries, expressiveness, conciseness and content adequacy. This shows that this is a starting point for the generation of task specific summary. According to this work (Moreno, 2013) implements a tool called JSummarizer that highlights the main functionality of a class depending on class stereotype. This tool ignores the existing comments, and uses a set of predefined heuristics to determine the summary information, and it applies natural language processing and generation techniques to form the summary in order to understand large and complex classes.

(Moreno L. , 2014) proposes an approach to generate a natural language description to the class of the source code and sets of code changes, since they are a complex artifact and it will provide a broad and quick understanding to the software. This approach uses the class stereotype to describe the structure of the class, and the relevant methods to describe the behavior of the class. While (Sridhara G. L.-S., 2011) describes a novel technique to generate comments for Java method parameters automatically, in order to provide an overview of the role of the parameter in facilitating the wanted functionality of the method. This technique integrates the parameter comments with the summary. The evaluation task of this work depends on nine human evaluators with programming experience ranging from 4 to20 years, and have software industry experience ranging from 1 to 7 years to evaluate Accuracy, Utility-Standalone, Utility-Integrated and Necessity. Another approach were introduced by (Hammad, 2016), in order to generate a textual description to the main services provided by java packages, by extracting the syntactic information from java source code.

The previous static analysis techniques give a descriptive report, while (Ellina, 2007), and (Myers, 2011), provide class call graph that views the method invocation within each class.

Many other studies depend on visualizing source code, in order to understand the software. (Lanza, 2011), uses CodeCity tool to visualize software elements as a city in 3D view, each package is presented as a city within this city the building represent the class, the height of the building represent number of methods within class, and the width of the building represent number of attributes. While class metrics are shown as a solar system metaphor (Graham, 2004), whih represents LOCM, class coupling, inheritance level metrics. Also in (Lanza, M, 2001), software evolution is visualized using CodeCrawler, each software system is represented as evolution matrix. Each class in the matrix is represented as rectangle the width represents number of method, and the height represents the number of instance variables.

On the other hand (Haiduc S. J., 2013) proposes a novel technique that automatically generates an extractive summaries for source code entities, depending on lexical information, this approach is based on using lexical and structural information from the method in the source code by dealing each method as a separate document, this is done using Latent Semantic Indexing (LSI) as the text retrieval (TR) technique, the result this work show that applying text retrieval (TR) technique gives better results than applying natural language summarization.

2.4 Software Metrics

Metric is a measurement that is computed directly from the program source code, to improve the quality and validity of software systems. Every system has its own complexity which should be measured to improve the quality of the system; static metrics are derived from the measurement on static analysis of the software code (Sonal Chawla, 2013).

In 1994 Shyam R. Chidamber and Chris F. Kemerer in their work (Chidamber, 1994), aims to measure the class inheritance hierarchy, so they developed and implemented a new set of software metrics for OO design. The developed metrics reflect viewpoints of experienced OO software developers and also based in measurement theory. Those metrics are summarized below:

- Weighted Methods Per Class (WMC): this metric aims to count the number of methods per class, it is calculated by the following equation:

$WMC = \sum_{x=1}^{n} cx$ ..... **(1)**

Where cx is the complexity for each method (e.g., Cyclomatic complexity, volume, etc.)

- Depth of Inheritance Tree (DIT): this metric means the maximum length from a node to the base class or the root.
- Number of children (NOC): this metric aims to find the number of subclasses that are immediately subordinate to a class.
- Coupling between objects (CBO): it is the number of collaborations between two classes.
- Response For a Class (RFC) : to calculate RFC the following equation is used:

$RFC = \sum_{x=1}^{n} Mcx$ ..... **(2)**

Where Mcx is the number of methods called in response to a message that invokes methodMx. So $RFC = |RES|$ where RES is the response set for the class, given by: $RES = \{ME\} \cup all\ x\ \{Rsx\}$ ..... **(3)**

Where $\{Rsx\}$ is the set of methods called by method x , and $\{ME\}$ is the set of all methods in the class.

- Lack of Cohesion in Methods (LCOM): it is an important concept in OO programming is Cohesion. It aims to give an indication whether a class represents multiple abstractions or a single abstraction. The idea is that class should be refactored into more than one class if it represents more than one abstraction, each of which represents a single abstraction. The following equation:

Let $X = \{(Ai, Aj) | Ai \cap Aj = \emptyset\}$ and $B = \{(Ai, Aj) | Ai \cap Aj - \emptyset\}$ ..... **(4)**

If all n sets $\{A1\}, \ldots \{An\}$ are$\emptyset$, then let $X = \emptyset$.

So $LOCM = |X| - |Y|, if\ |X| > |Q|$ ..... **(5)**

In another words we can say that LCOM is coming from subtracting number of non-empty intersections from the number of null intersections.

On the other hand many other metrics aim to measure the quality of the method, for example Maurice Howard Halstead (Halstead, 1977), introduce the Halstead complexity measures metric, Halstead's goal was to identify measurable relations between software and properties of them. Several measures can be calculated according to Halstead complexity, by calculating the following numbers:

- $num1$: is the total number of distinct operators.
- $num2$: is the total number of distinct operands.
- $NUM1$: is the total number of operators.
- $NUM2$: is the total number of operands.

The measures that can be calculated according to the previous numbers are:

Program vocabulary: $num = num1 + num2$ ..... **(6)**

Program length: $NUM = NUM1 + NUM2$ ..... **(7)**

Volume: $VOL = NUM\ log_2\ num$ ..... **(8)**

Difficulty: $DIFF = \left(\frac{num1}{2}\right) * \left(\frac{NUM2}{num2}\right)$ ..... **(9)**

Program effort: $EFO = VOL/DIFF$ ..... **(10)**

In order to understand and analyze the program component and the relationship between them, and since the graphs are one of the preferred views for the analysts both call graph and flow graph are important representations for the software, so in order to represent calling relationships between functions within the class call graph is used. Call graphs are used for program analysis and human understanding as a basic program analysis result. In call graph each node represents a function and each edge between (f, g) indicates that function f calls function g. and so on; a cycle in the graph indicates recursive function calls.

Another important representation that is used to represent the function is the flow graphs, they are useful, and an important tool for testing programs or program components during software development, control flow graph (CFG) is a directed graph in which the nodes represent basic blocks and the edges represent paths between the control flow nodes.

(Allen, 1970), used control flow graph as a graphical view in order to represent the method. According to control graph McCabe's Cyclomatic Complexity (CC) can be calculated. McCabe's Cyclomatic Complexity (CC) is intended to measure the complexity of software by analyzing the software program's flow graph, it was introduced in 1976 by Thomas J. McCabe as shown in (McCabe, 1967), CC was organized to measure the size of the test case space, and it can be calculated as equation 11 below show:

$$CC = ED - ND + 2 \quad \text{..... (11)}$$

Where:

ED: is the total number of edges, and ND: is the total number of nodes.

2.5 Representing Source Code as XML

Source code is usually kept as a plain text, because it is easy to manipulate the plain text using text editor and other software tools. In order to represent the hierarchal structure for source code, the compiler builds a tree called the Abstract Syntax Tree (AST), which make it easy to reads source code, and analyze it (Fujita, 2007). The format of AST and its contents are great for the compiler but they are greatly lacking with respect to the need of software engineering. So the well structure of the source code enables reading, writing code but not explicit describing structure. The field of document engineering is the common solution for this problem; this field inserts special tags or characters into the document were it adds structural information. So the dealing is with text, which makes it easy to be parsed, searched and transformed with the aid of these tags. The standard that is used to solve software engineering problems and also forming documents and information is the Extensible Markup Language (XML) (Collard, 2002).

Analyzing and manipulating XML by software tools is easy; this comes from the universal format representation. The World Wide Web Consortium (W3C) designs the standard Generalized Markup Language (SGML) which has an important subset called XML. XML document consists of text marked up with tags enclosed in angle braces. In XML document the inherent hierarchical structure make it convenient for representing source code constructs. So representing source code as XML document have the following benefits (Aguiar, 2004):

1. Explicit code structure: by nature, XML documents are structured, and can be used to represent the code as a tree , so code generation and transformation using predefined templates will quickly done.
2. Powerful querying capabilities: modern IDEs usually include specific tools for source code that allow searching for any of program artifacts, such as classes, methods, fields, etc. In addition to textual searches using regular expressions. The features of source

code are useful but are only a small subset of what is able to query with XML standards and tools, such as XPath and XQuery.

3. Extensible representation: extensions are often supported as Comments because the plain-text source code is not easy to extend with new code artifacts because they would break down the code structure and require parser modifications. In a XML document. Distinct tools can insert and define their own elements in the code structure and then process only the elements that are relevant for them.
4. Flexible formatting: In XML representation of code, the structure can be extracted from the coding style. Re-formatting of source code is easy because of the XML standards and tools, such as XSLT, because it allow using different styles in order to enrich its readability through the suitable usage of layouts, fonts, colors and links.
5. Cross-referencing: referencing code fragments directly to code artifacts is possible, thus enabling the relocation of code fragments without disrupting references. The file position is usually the reference of the source code fragments in plain text, for example line and column numbers. On an XML tree like structure of a program.
6. Wide support: XML tools are available in all major systems so it is completely satisfy the requirement of program representation that it must be widely supported in a wide variety of platforms.

Many xml representations were introduced to represent java source code as a document representation. For example JavaML that introduced by Greg Badros, in 2000 (Badros, 2000), to replace the classical source representation of Java programs based on XML, JavaML adds the semantic and structural information to the source code text files, JavaML reflects directly the structure of the software artifact in the overlapping of elements in the XML-based syntax.

JavaML 2.0 enriches the original JavaML (Aguiar, 2004) which adds more information at several levels ranging from the lexical level to the semantic level to have a full lexical information about tokens, comments and formatting, small enhancements for structural information, and much richer semantic information to symbol definitions, references and type information. The work (Mamas, 2000) Integrated Software Maintenance Environment (ISME) was introduced in order to represent Source code as XML DOM trees that offer a higher level of portability and openness than custom Abstract Syntax Trees.

Another document-oriented XML representation of source code is the srcML (Collard M. D., 2011) as a single XML document it covers the source code text and the Abstract Syntax Tree information as a tag. This

tool supports the following programming languages, Java, C, and C++. It aims to provide full access to the source code at the lexical, structural, documentary, and syntactic levels.

This is supported by the srcML toolkit. SrcML have the ability to translate over than 7500 Line of Code per second, it also allows users to perform fact extraction in multiple ways, for example XPath query can be used to address the wanted facts in the document, and also XPath support the calculation of numeric results that aims to find the number of occurrences of elements. SrcML tool can perform two translation the first is translating source code into the srcML format, and the second is translating the srcML into source code. There are also many information can be derived from the srcML such as call graph and dependency graph (Collard M. L., 2005).According to that The proposed methodology uses the srcML tool as a parser in order to transform the source code to XML file.

The powerful expression within XPath query, make it easy to retrieve relevant information, and parse an xml document. To perform accessing and manipulating to the xml file there is a need to use the Document Object Model (DOM) interface. The DOM need because it enables reporting the information that is found in xml tree nodes.

العنوان:	Automatic Generation of Descriptive Summary for Source Code Artifact
المؤلف الرئيسي:	البطوش، أماني عبدالسلام
مؤلفين آخرين:	حماد، مصطفى محمد(مشرف)
التاريخ الميلادي:	2016
موقع:	مؤتة
الصفحات:	1 - 86
رقم MD:	951089
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة مؤتة
الكلية:	عمادة الدراسات العليا
الدولة:	الاردن
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، برمجية جافا، برمجية سي
رابط:	https://search.mandumah.com/Record/951089

Chapter 3
The Proposed Methodology

Introduction

The proposed methodology aims to summarize the source code for the software by generating a descriptive summary that contains a set of views in order to describe the source code artifacts. The generated views involve analyzing the software source code in order to determine its elements and the relations between them, and also to calculate many of the important software metrics automatically to enrich the view information.

In this thesis, static analysis technique is applied on the source code in order to reach our goal; firstly Java source code of the software is the input. Then, this source code is parsed into xml file is generated, this generated file contains all the elements of the source code as xml tags, which are organized in a hierarchical way, and according to srcML format a number of features are extracted depending on the xml file tags, that are statically analyzed which are then used then used in the generated descriptive summary views.

The generated descriptive summary of the proposed approach considered to be used and serve in the fields of maintenance, software understanding, reuse, evolution, changes and reverse engineering, since a huge amount of the source code needs to be analyzed and understood.

3.1 Proposed Methodology Overview

The proposed methodology aims to extract a number of views in order to summarize the source code artifacts, this methodology focus on analyzing the source code statically.

- Firstly the source code is transformed using the srcML tool into xml format as XML file that represent the source code.
- The parsed XML file is parsed and analyzed where many of XPath queries are applied to it, these queries depends on the DOM that describes the xml tree nodes and the relationships between them.
- At the end, the output are number of views that summarized the software source code artifacts, by focusing on describing the features of the following artifacts: packages, classes, and methods:
  1- Package report.
  2- Class report that contains a set of services shaping the main role of the class within the package
  3- Class call graph supported with main class metrics that measure the quality of the class and quantity information such as total number of methods, and total number of attributes.
  4- Method control flow graph that is supported with the Cyclomatic complexity and Hallstead complexity measures.

- The proposed methodology ignores the interface classes from summary since it groups the empty bodies' method to each other. It also ignores the summarization of constructors since they are used to initialize data fields.

This methodology consists of a number of processes to reach the goal of generating descriptive summaries for the source code artifacts. Figure 1 shows the main processes in the proposed approach.

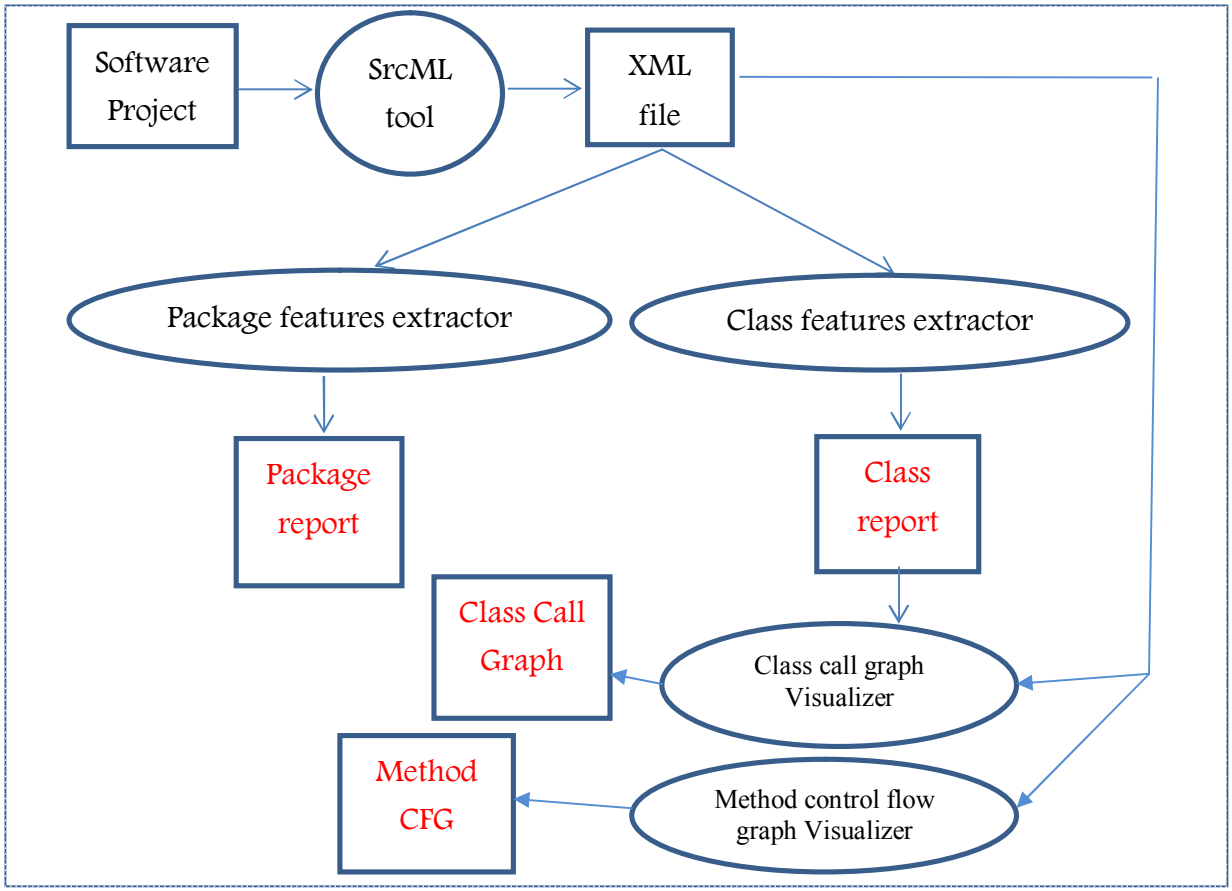

Figure 1
Proposed Methodology Overview

As shown in Figure 1, the proposed methodology consists of the following steps:

1- Project source code is parsed into XML file using srcML tool, which is generated as xml file.
2- Static analysis technique is applied in order to extract a number of features.
3- Two main extractors are applied to the xml file, where they provide a list of features.
4- The mined features provide us with two main reports, package report, and class report.
5- The class report is used with the XML file to pass a visualizer in order to provide us with class call graph, and the method control

flow graph. The following sub-sections describe in details each step in the proposed methodology.

3.2 SrcML tool

SrcML tool in this methodology is used to transform project source code into xml file that have the srcML tool format. Figure 2, provides an example of a Java class that is transformed into XML file using the srcML tool as shown in Figure 3. The following command generates an XML file named FigInspectorPanel from the class FigInspectorPanel.java:
src2srcml.exe --language=Java FigInspectorPanel.java -o FigInspectorPanel.xml.

```
1   package org.argouml.dev.figinspector;
2
3   import java.awt.BorderLayout;
4   import java.util.Vector;
5
6   import javax.swing.JPanel;
7   import javax.swing.JScrollPane;
8   import javax.swing.tree.DefaultMutableTreeNode;
9
10  import org.argouml.dev.MessageNodeBuilder;
11  import org.argouml.sequence2.diagram.FigClassifierRole;
12  import org.tigris.gef.base.Globals;
13  import org.tigris.gef.base.Layer;
14  import org.tigris.gef.event.GraphSelectionEvent;
15  import org.tigris.gef.event.GraphSelectionListener;
16  import org.tigris.gef.presentation.Fig;
17  import org.tigris.gef.presentation.FigEdge;
18  import org.tigris.gef.presentation.FigGroup;
19  import org.tigris.gef.presentation.FigText;
20
21
22  public final class FigInspectorPanel
23      extends JPanel implements GraphSelectionListener {
24
25      private static final long serialVersionUID = -3483456053389473380L;
26      private static final FigInspectorPanel INSTANCE =
27          new FigInspectorPanel();
28      public static FigInspectorPanel getInstance() {
29          return INSTANCE;
30      }
31      private FigInspectorPanel() {
32          Globals.curEditor().getSelectionManager()
33              .addGraphSelectionListener(this);
34          setLayout(new BorderLayout());
35      }
36      public void selectionChanged(GraphSelectionEvent selectionEvent) {
37          removeAll();
38          DefaultMutableTreeNode rootNode = new DefaultMutableTreeNode();
39          if (selectionEvent.getSelections().size() == 0) {
40              Layer lay = Globals.curEditor().getLayerManager().getActiveLayer();
41              for (Object o :  lay.getContents()) {
42                  addFig ((Fig) o, rootNode, false);
43              }
44          } else if (selectionEvent.getSelections().size() == 1) {
45              addFig ((Fig) selectionEvent.getSelections().get(0),
46                      rootNode,
47                      true);
48          }
49          FigTree tree = new FigTree(rootNode);
50          tree.setRootVisible(false);
51          tree.expandAll();
52
53          JScrollPane scroller = new JScrollPane(tree);
54          add(scroller);
55      }
56      private void addFig(
57              final Fig f,
58              final DefaultMutableTreeNode rootNode,
59              final boolean includeEncloser) {
60          // Build the selected Fig first and then iterate up through
```

```
        // its enclosers building those also.
        for (Fig fig = f;
                fig != null;
                fig = includeEncloser ? fig.getEnclosingFig() : null) {
            DefaultMutableTreeNode figNode =
                new DefaultMutableTreeNode(getDescr(fig));
            rootNode.add(figNode);
            buildTree(fig, figNode);
            if (fig instanceof FigClassifierRole) {
                MessageNodeBuilder.addNodeTree(rootNode,
                        (FigClassifierRole) fig);
            }
        }
    }
    private void buildTree(Fig f, DefaultMutableTreeNode tn) {
        if (f instanceof FigGroup) {
            FigGroup fg = (FigGroup) f;
            for (int i = 0; i < fg.getFigCount(); ++i) {
                addNode(tn, fg.getFigAt(i));
            }
        } else if (f instanceof FigEdge) {
            FigEdge fe = (FigEdge) f;
            Fig lineFig = fe.getFig();
            addNode(tn, lineFig);
            addNode(tn, fe.getSourceFigNode());
            addNode(tn, fe.getSourcePortFig());
            addNode(tn, fe.getDestFigNode());
            addNode(tn, fe.getDestPortFig());
            for (Fig pathFig : (Vector<Fig>) fe.getPathItemFigs()) {
                addNode(tn, pathFig);
            }
        }
    }

    private void addNode(DefaultMutableTreeNode tn, Fig fig) {
        DefaultMutableTreeNode childNode =
            new DefaultMutableTreeNode(getDescr(fig));
        buildTree(fig, childNode);
        tn.add(childNode);
    }

    private String getDescr(Fig f) {
        if (f == null) {
            return null;
        }
        String className = f.getClass().getName();
        StringBuffer descr = new StringBuffer(
                className.substring(className.lastIndexOf(".") + 1));
        descr.append(
                " - bounds=[" + f.getX() + "," + f.getY() + "," + f.getWidth()
                        + "," + f.getHeight() + "]");
        if (!f.isVisible()) {
            descr.append(" - INVISIBLE");
        }
        if (f.isFilled()) {
            descr.append(" - FILLED");
        }
        descr.append(
                " - fill=[" + f.getFillColor().getRed() + ","
                        + f.getFillColor().getGreen() + ","
                        + f.getFillColor().getBlue() + "]");
        if (f.getOwner() != null) {
            descr.append(" - owner=").append(f.getOwner());
        }
        if (f instanceof FigText) {
            descr.append(" \"").append(((FigText) f).getText()).append("\"");
        }

        descr.append(" - lay=").append(toString(f.getLayer()));
        descr.append(" - grp=").append(toString(f.getGroup()));
        return descr.toString();
    }

    private static String toString(Object o) {
        if (o == null) {
            return "null";
        }
        try {
            return o.toString();
        } catch (Throwable e) {
            return "???";
        }
    }
}
```

Figure 2
FigInspectorPanel Class for ArgoUML Open Source

Figure 2, provides an example for Java class, this class is called FigInspectorPanel.java class that is defined under a package called dev. figinspector, this package is a part of the ArgoUML open source project (CollabNet, 2001). FigInspectorPanel class, as shown in Figure 2, FigInspectorPanel class consists of two attributes called: serialVersionUID, and FigInspectorPanel, one constructor with the name FigInspectorPanel,

and seven methods. In this class the method addFig uses for statement, it invokes buildTree method in line 55. Using srcML tool we transform the class FigInspectorPanel. Java into XML files which have the srcML format with xml tags that are represented hierarchically.

As shown in Figure 3, part of the parsed xml file for FigInspectorPanel.java class. This part of the XML file holds all the syntactic information that is represented in the class, so we easily analyze each artifact of the source code.

The hierarchy of the XML file is clearly seen in this Figure 4, we note from the xml file that:

1. <class> tag that is shown as a parent node, where the type of the class shown as < specifier > tag, and the name of class shown as <name> tag, both < specifier >, and <name> are child node from<class> tag .
2. The constructors are represented as <constructor> tag, it is mentioned one time which provides us with the fact that one constructor is defined within the class.
3. The method is shown as <function> tag this tag is mentioned seven times in the XML file; from here we find that there are seven methods within FigInspectorPanel class.
4. The comment statement is represented with the tag that is started with <comment type=", and ended with the tag </comment>, the comment used in this class is line comment as the xml file shows.
5. Declaration statement is represented as <decl_stmt> tag, which appears in lines 13, and 14 to indicate that an attribute is declared. <decl_stmt> tag is a child node within the class <block> tag.
6. <specifier> tag that is an important part of class, declaration statements, and method, it provides us with the type of each one of them.
7. Each <block> tag within the xml file hold the information about each method declaration and the <call> tag which provides us with the method invocation.
8. For statement that is used in Figure 3 in the FigInspectorPanel class is represented in the xml file by line 35 in Figure 3.

According to this we can say that everything can be applied to a single Java class using the srcML xml format can be easily applied to the whole Java source code project. Since the Java project consists of a number of structured packages that are combining an organized set of classes. For the previous example that is represented in Figure 3, the package dev.figinspector consists of two classes, FigTree.java class, and FigInspectorPanel.java.

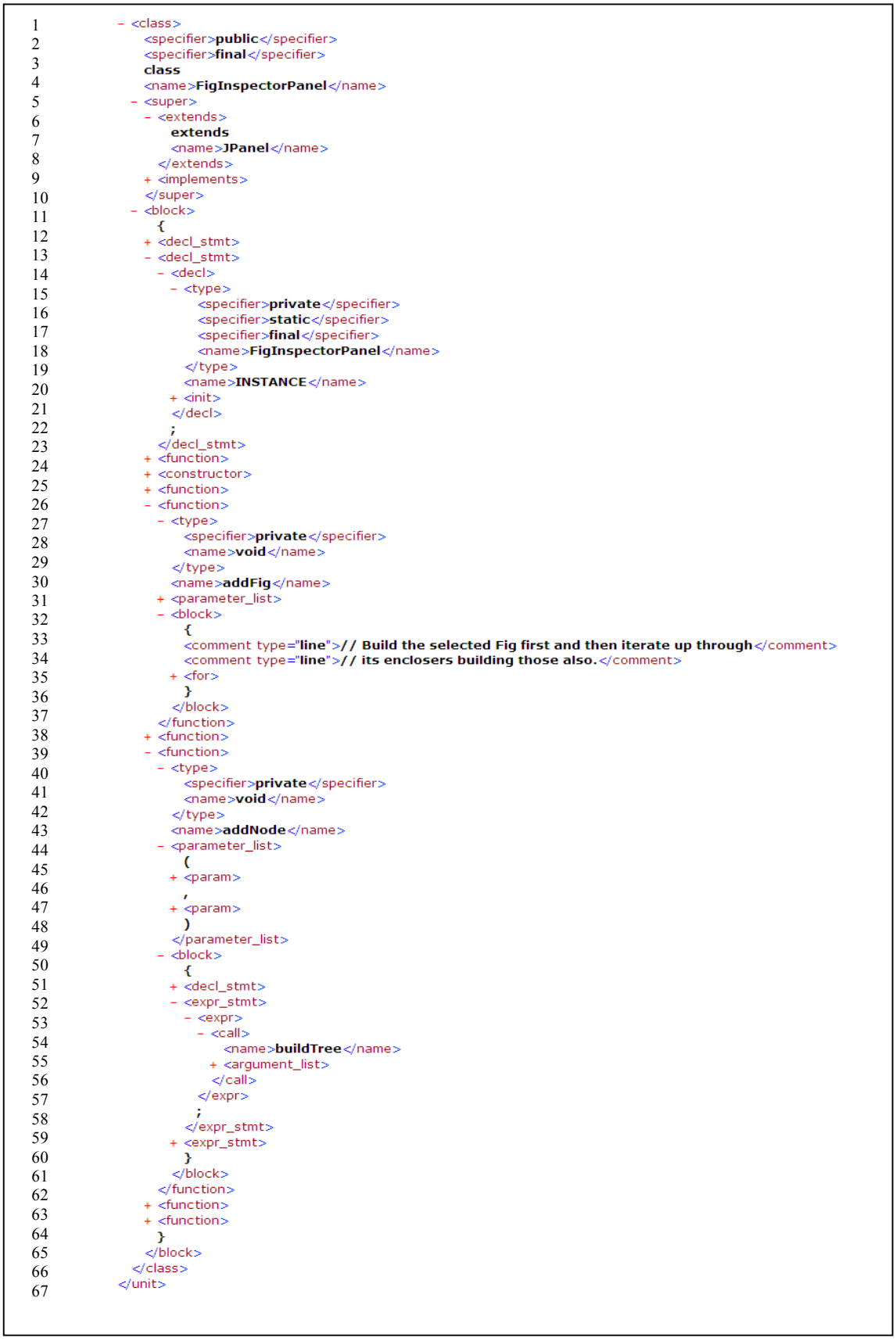

Figure 3
Part of the Xml File Parsed from FigInspectorPanel Class in Figure 2

3.3 Package feature extractor

Package feature extractor aims to extract the following features from each package within the software: package name, total number of classes, total number of constructors, total number of methods, and the total number of attributes within each package. This is implemented by using a set of XPath queries that are predefined in order to extract package features. For example, // package XPath query that is used to get all packages elements from srcML no matter where they are. The Package feature extractor is described by a pseudo code in subsection 3.3.1, and an example that represents the package report is presented in 3.3.2 subsection, those two sub-sections are discussed below:

3.3.1 Pseudo Code for Package feature extractor

In this subsection we show the pseudo code for the package feature extractor algorithm that is applied to extract the Package report, in this algorithm the input is the XML file that is parsed by srcML tool and the output is the Package report. Algorithm 1, illustrates the pseudo code for Package feature extractor.

Algorithm 1: Package Feature Extractor Algorithm

```
1   For each < unit language > tag, extract: <package>, and <class>.
2       Within each <package> tag, find : <name>.
3
4       Within each < class> tag, find : <name>, and <block>.
5       Within each <block> tag, find: <constructor>, <decl_stmt>, and
6
7   <function>.
8     Return:
9
10              Package name.
11              Total number of classes.
12              Total number of constructors.
13              Total number of methods.
14
15              Total number of attributes.
16
```

Algorithm 1, that represents the pseudo code for Package feature extractor aims to provide the package report, this report returns with total number of classes, total number of constructors, total number of methods, and total number of attributes within each package. In Line 1, the algorithm extracts two main elements from the XML file, package and class. Those two elements appears as tags, and since the class of the target source code contains the information about the constructors, methods, and attributes, class tag has an important tag called block, in Line 4 those elements are extracted. The general format for the package report is shown in Figure 4.

Package Report:
Package name of package:
Total number of classes: Total number of classes.
Total number of methods: Total number of methods.
Total number of constructors: Total number of constructors.
Total number of attributes: Total number of attributes.

Figure 4
The Generated Package Report Format

As shown by Figure 4, the key words of the generated package report are shown as bold words, while the returned values from Algorithm 1, and are shown as underlined words.

3.3.2 Example for Package Report

Figure 5, shows Screenshot example of the package report, which has been generated for the package dev.figinspector, from ArgoUML open source as plug-in library in NetBeans framework.

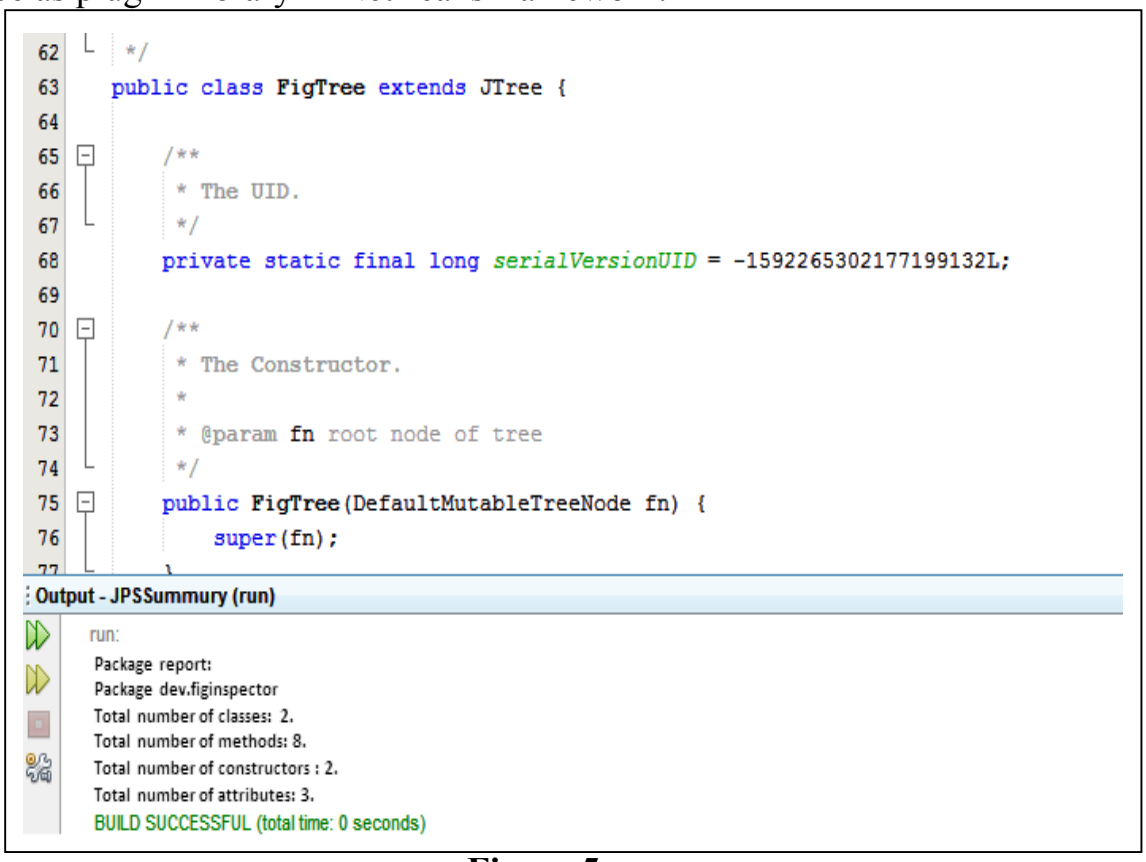

Figure 5
Screenshot Example for dev.figinspector Generated Package Report from ArgoUML Open Source as plug-in Library in NetBeans Framework

The output result in Figure 5, gives the package report, which represent the first textual view (package report). This view consists of

package name followed by the total number for each of classes, methods, constructors, and attributes within the package. So the Package dev.figinspector consists of two classes FigTree class, and FigInspectorPanel class, eight methods, one constructor and three attributes.

3.4 Class Feature Extractor

Class feature extractor is the second feature extractor from the proposed work; it is designed in order to provide us with the class report. Class feature extractor, aims to extract general information about each class. Such as class name, class type, and the package that class defined within. Also, it aims to extract the services that class provides by getting the methods that are defined within each class using XPath query, for example the following XPath query :[class/function], selects all method elements that are children in class.

3.4.1 Pseudo codes for Class feature extractor

This subsection describes two algorithms that are used to extract the features of class, the first algorithm is Algorithm 2, that aims to extract the general information of class, and the second algorithm is Algorithm 3, which aims to extract the services that each class provides. For both algorithms the input is the XML file that is generated by srcML tool.

Algorithm 2: Class Information Algorithm

```
1  For each <package>tag, extract: <name> tag, and <class> tag.
2    From each <class> tag, find: <name>, <specifier>, and <extends>.
3  Return :
4   Class name.
5   Package of the class.
6   Class type.
7   Super class name.
8
```

From Line 1, in Algorithm 2 the name of the package and the classes within this package are extracted. This algorithm aims to extract the features show in Lines (4-8), depending on the following elements of the target software class: name, specifier, and extends. <name> tag, gives the name of class, < specifier> tag, provides the type of class, and <extend> tag appears if the class has a super class.

Algorithm 3: Class Services Algorithm

```
1   For each <class> tag, extract <block> tag:
2    For each <block> tag, extract:
3
4      < function> tag:
5        From < function> tag, find:
6
7        <specifier>
8        <name>
9
10       <decl>
11
12     <decl_stmt> tag:
13       From <decl_stmt> tag, find:
14
15        <type>
16        <name>
17
18     <expr_stmt> tag:
19       From <expr_stmt> tag, find:
20
21        <call>
22        <name>
23
24     Return
25
26     Method name.
27     Method type.
28
29     Class attribute.
30     Attribute data type.
31
32     Local data.
33     Local data types.
```

Each class is built from a number of methods, where every method in the class provides a service to the class, those services are listed according to their occurrence on the class, and by collecting these services we can say that the class provides them to the software. The main tags that are extracted from <block> tag are:

1- < function> tag, in Line 4. Where the children specifier, name, and decl are extracted.
2- <decl_stmt> tag, in Line 12. Where the children type, and name are extracted.
3- <expr_stmt> tag, in line 19. Where the children call, and name are extracted.

At the end this algorithm returns the features that are shown in Lines (25–33). Both Algorithms 2 and 3 are used to perform the class report. The generated report format is shown in Figure 6, that shows the class report format that is used to view the class report, the bold words in this figure are the key words in the report, and the underlined words in

the report are the returned values from both Algorithm 2, and Algorithm 3.

Class class name **report:**
Class class name **is declared in package:** package name **as:** class type**. Has a super class**: class name.

If the method uses the attributes of class:
Give the following report:
The service is: Method name. **The service returns** method data type**. The service uses the attributes:** Attribute name **with** attribute data type.
This service uses the local method: Method name.
This service use Method: Invoked method name.

If the method uses the local parameters:
Give the following report:
The service is: Method name. **The service returns** method data type**. This service uses local data:** Local data **with** local data type.
This service uses the local method: Method name.
This service use Method: Invoked method.

Figure 6
The Generated Class Report Format

3.4.2 Example for Class report

Firstly, the generated report gives general information about the class, this information provides the name of class, the package that holds this class, and if the class has super class, the name of super class is shown. Then, the services that each class provides are mentioned.

For FigInspectorPanel class that is shown in Figure 3, the generated class report from this class is shown in Figure 7 as plug-in library in NetBeans framework, where it describes the services that are provided in FigInspectorPanel class. And since FigInspectorPanel class consists of seven methods, it provides seven services, followed by FigTree class report, where FigTree is the second class within the package dev.figinspector, and consists of one method, so FigTree class report provides one service.

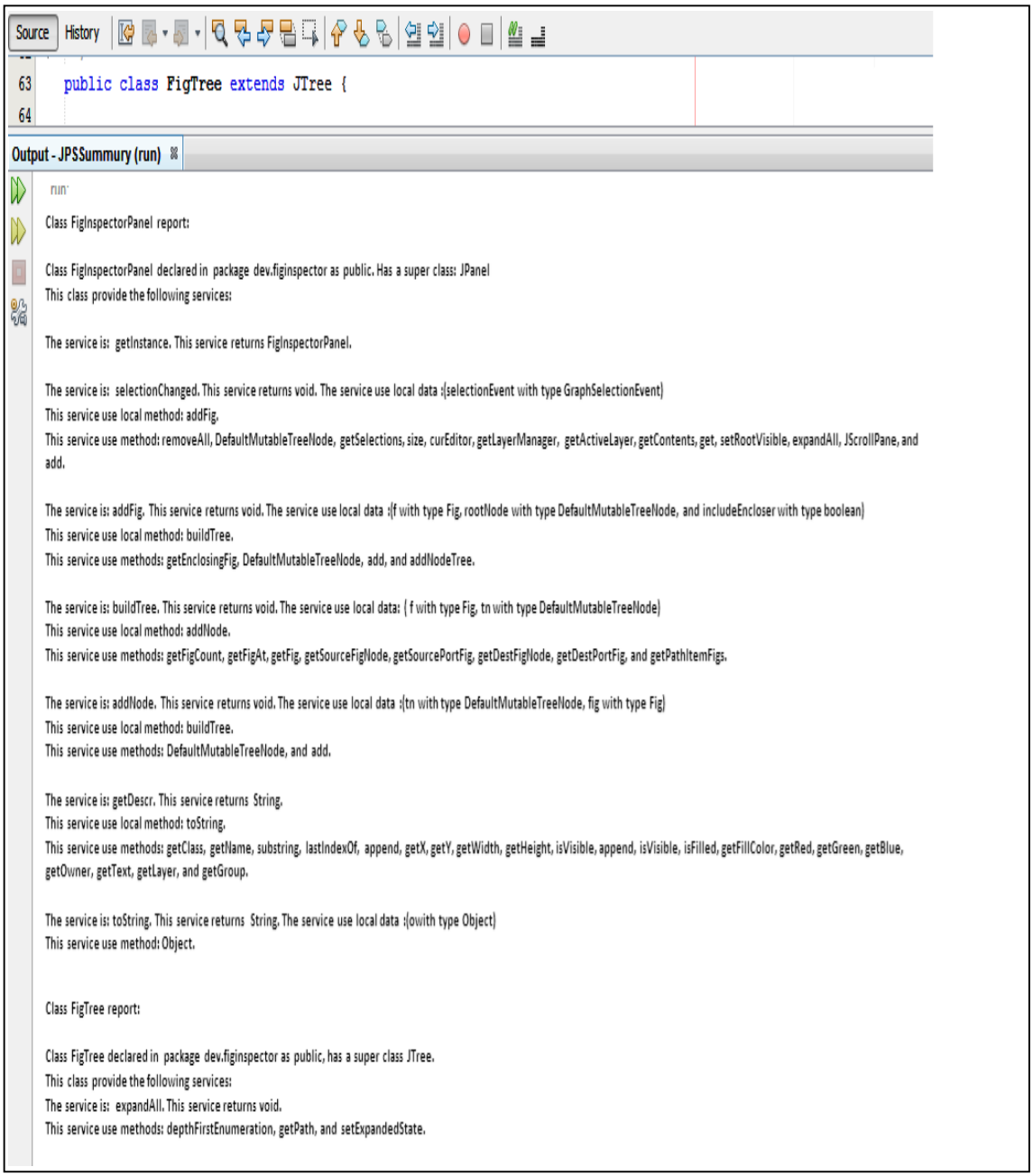

Figure 7

Screenshot Example from The Generated Class Report for Class FigInspectorPanel, and Class FigTree in package dev.figinspector as plug-in Library in NetBeans Framework

3.5 Class Call Graph Visualizer

In order to measure the quality of the class, a set of metrics are applied. The proposed methodology applies four metrics to measure the quality of the class: LOCM, RFC, NOC, and WMC metric. Also the proposed methodology calculates the class size. These metrics and calculations are discussed by the algorithms 4, 5, 6, 7 and 8. The generated class call graph depends on both class report and the xml file.

Each one of the metrics that are used in the class call graph is generated by an algorithm. The extracted class call graph represents all the method invocations, if they are directly invoked, or indirectly. Class call graph Visualizer is discussed by the following subsections.

3.5.1 Class Call Graph Visualizer pseudo codes

In order to visualize the class call graph, firstly we start to discuss class size by Algorithm 4, and then class metrics are discussed by Algorithms (5-8).

Algorithm 4: Class Size Algorithm

```
1  For each <block>  tag:
2     Count < function> tag.
3     Count <decl_stmt> tag.
4     Return :
5     Total number of methods.
      Total number of attributes.
```

As Algorithm 4 shows, both total number of methods and total number of attributes are the main features used to shape the class size, where this algorithm depends on <block> tag, where the two calculations occurs, counting <function> tag, and counting < decl_stmt> tag. After that Algorithm 5 is transformed in order to measure (LOCM).

Algorithm 5: Lack Of Cohesion Metric (LOCM) Algorithm

```
1  For each < function> tag, Do:
2        In every <name> within <expr> check :
3           If (<name> in <decl_stmt> is accessed by the same <expr> in
4   <expr_stmt>)
5           Define set i, where: Set i= {accessed attribute names}.
6
7           Else define set j, where: Set j= φ.
8        For Function(1) to Function(n):
9              ((Function1.set1) ∩ (Function 2.set2)….(Function (n-
10  1).set(n-1)) ∩ Function (n).set(n))
11    Let X = the number of null intersections.
12    Let Y = the number of non-empty intersections.
13    Calculate LOCM =|X| − |Y|.
14 Return (LOCM).
15
```

Algorithm 5 illustrates the pseudo code for LOCM algorithm. Line 3 checks if the name of attribute within the declaration statement is accessed by the expression of functions within class in order to organize them into sets, where two sets for each method of each class are defined. The following XPath query //src:decl_stmt, find all declaration statements.

The first set holds the names of the attributes accessed by the method, and the second set defined to hold φ. Which then the intersection between those two sets is found, at the end subtracting the non-empty intersections from the null intersections gives the LOCM for each class. Algorithm 6 illustrates the pseudo code for the Response for Class Metric algorithm.

Algorithm 6: Response for Class Metric Algorithm

```
1   Input: Algorithm 2.
2      Define Set 1, Set 2.
3        In every <function> tag, do:
4          If (<name> of <function> declared in <class>)
5            Add name to Set 1.
6          Else
7           Add method name to Set2.
8       RFC = Set 1 U Set 2.
9      Return (RFC).
10
```

The pseudo code of the response for class that is represented by Algorithm 6 depends on Algorithm 2, class information algorithm. Set 1 contains the names of local method invocation as described in line 4, and Set 2 contains the names of non-local method invocation. The union between those two sets provides us with RFC metric as Line 9 shows. In order to measure the software quality Weighted Methods per Class is another metric used, this metric is shown in Algorithm 7.

Algorithm 7 : Weighted Methods per Class Algorithm

```
1  Input1: Algorithm 10.
2  Input2: Xml file.
3  In each <class> tag, find every <function> tag:
4     Calculate program vocabulary for each <function> tag.
5     WMC = ∑ program vocabulary.
6  Return (WMC).
```

Algorithm 7, aims to calculate Weighted Methods per Class depends on the program vocabulary for each method, so we call the Hallstead complexity that is calculated by Algorithm 10. To return the WMC, the algorithm sums the program vocabulary for each method in the class. Algorithm 8 is set to calculate the number of children metric for each class.

Algorithm 8 : Number Of Children algorithm (NOC)

```
1  Find all super classes
2     If ( the <name> of the super class is found in another <class>
3          or <name> have <extend>)
4     Count <class>.
5     Return count;
```

NOC metric that is shown by Algorithm 8 depends on the super class names that are mentioned as a class name in another <class> tag. The following XPath query example aims to Find all super classes: [//src:class[src:specifier[.='super ']]], and in order to returns the NOC, we count the classes that satisfy the condition in Line 3. Algorithm 9, illustrates the pseudo code for the class call graph algorithm.

Algorithm 9: Class Call Graph Algorithm

```
1  Input 1: Returned values from Algorithm 4.
2  Input 2: Returned values from Algorithm 5.
3  Input 3: Returned values from Algorithm 6.
4  Input 4: Returned values from Algorithm 7.
5  Input 5: Returned values from Algorithm 8.
6
7        Draw rectangle with local method name for each method.
8        Draw line between the local methods that call each other.
9        Draw dotted rectangle with method name for each invoked
10       method.
11       Draw dotted line between the local methods of class and its
12       invoked method.
13       Count the number of local methods that calls each other.
14       Put this number on the line matches between those methods.
```

As shown in Algorithm 9, this algorithm is used to draw the class call graph depending on calling number algorithms (4-8), in Lines (1-5), two types of rectangles are drawn, rectangle for local methods names that performs an invocation, and doted rectangle for local methods names that are invoked. Also two types of lines are drawn, line that matches between local methods that calls each other, and the dotted line that matches between the local methods of class and its invoked method.

Figure 8, shows the Class Call Graph format. From Figure 8, the named rectangles (M1-M5), are local methods names that performs an invocation, and the named of doted rectangles (M6, M7) are the invoked methods that are not local methods. The lines that are drawn within the class shape matches between local methods that calls each other, and the dotted line that appears outside the class shape matches between the local methods of class and its invoked method. The name of class, class size, and

the main class metrics (LOCM, RFC, WMC, and NOC), are represented as a dotted rectangle that appears on the top left corner from the class.

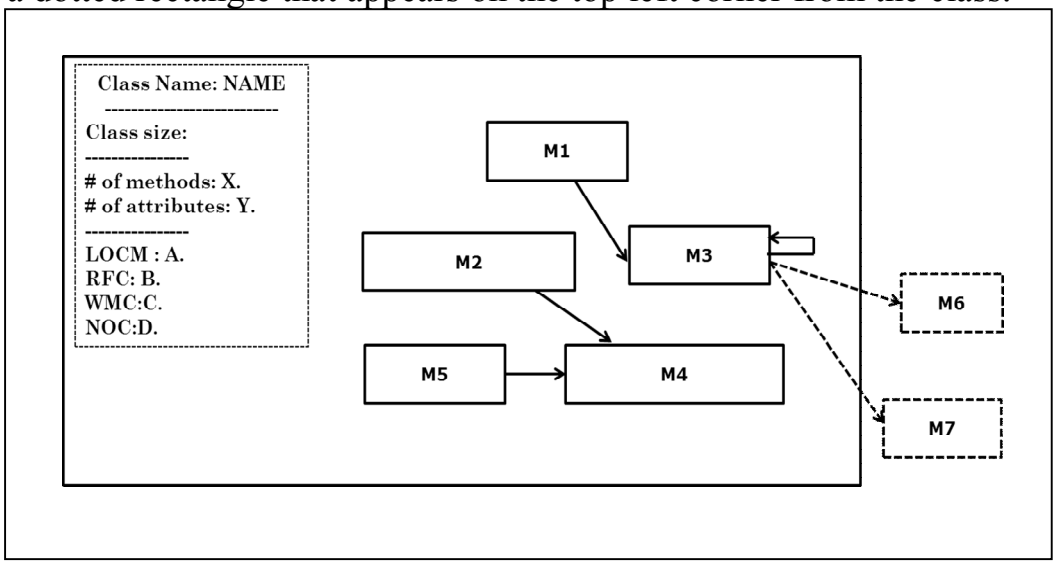

Figure 8
Class Call Graph Format

3.5.2 Class Call Graph (CCG) Example

FigTree class that is reported in Figure 5 is represented as FigTree class call graph in Figure 9. The method expandAll invokes depthFirstEnumeration, getPath, and setExpandedState methods that appears in dotted rectangles, and matches by a dotted lines, this indicates that they are not local methods.

Class metrics and class size are shown on the left top of the class call graph figure, for FigTree class, this class contains one method, and one attribute. Class metrics are clearly shown, where LOCM= -1, RFC=4, WMC=25, and NOC=0.

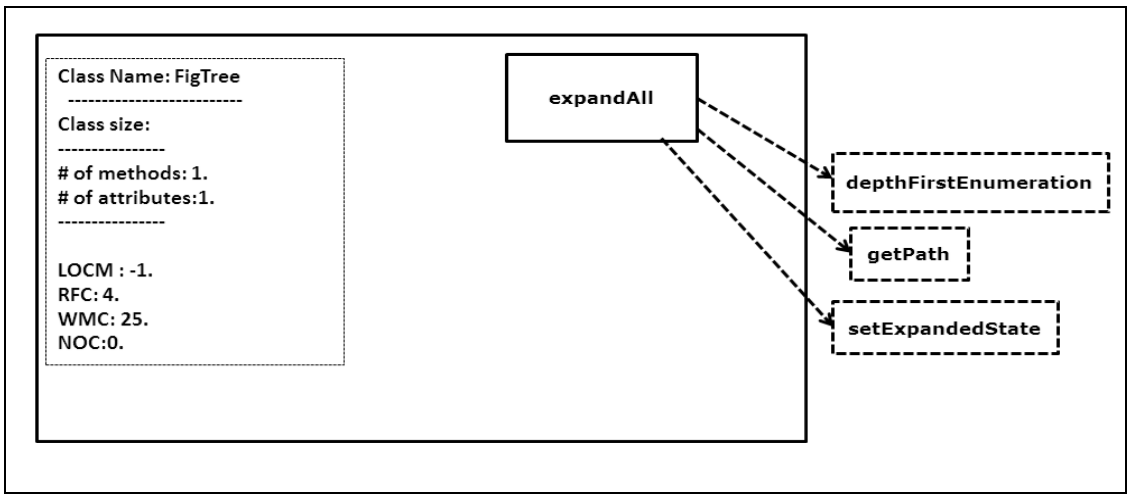

Figure 9
The Generated Class Call Graph for FigTree Class from Package dev.figinspector

3.6 Method Control Flow Graph Visualizer

Method control flow graph visualizer provides us with method control flow graph. This visualizer is discussed in the following subsections:

3.6.1 Pseudo codes for method control flow graph visualizer

Halstead complexity and Cyclomatic complexity are the main metrics that are used to measure the quality of the method. Algorithm 10, illustrates the pseudo code for the Halstead complexity algorithm. And in order draw control flow graph automatically, we propose a new approach that depends on introducing a group of rules shown in Table 3, and Appling the pseudo code of Algorithm 11.

Algorithm 10: Halstead Complexity Algorithm

1 Define the following numbers:
2
3 $num1$: is the total number of distinct operators.
4
5 $num2$: is the total number of distinct operands.
6
7 $NUM1$: is the total number of operators.
8
9 $NUM2$: is the total number of operands.
10 nput: xml file extracted by srcML generator.
11 n every < function> tag Do:
12 Ignore <comment> tag information.
13 Split word followed by a character, or character followed by word.
14 Split number followed by character, or character followed by number.
15 Find fixed symbols and reserved word within <function> tag.
16 Define fixed symbols and reserved word within <function> as NUM1.
17 Define everything else as NUM2.
18 Calculate the defined numbers.
19 Perform the following calculations:
20
21 Program vocabulary: $num = num1 + num2$
22
23 Program length: $NUM = NUM1 + NUM2$
24
25 Volume: $VOL = NUM\ log_2\ num$
26
27
28 Difficulty: $DIFF = \left(\frac{num1}{2}\right) * \left(\frac{NUM2}{num2}\right)$
29 Program effort: $EFO = VOL/DIFF$
30 **Return :**
31 Program vocabulary.
32 Program length.
33 Volume.
34 Difficulty.
35 Program effort.

As shown in Algorithm 10, this algorithm is used to calculate the measures of Halstead complexity, depending on the xml file, immediately the function tag. The features that this algorithm provides are shown in lines (31- 35), those extracted features are depending on number of steps performed in Lines (12-19). In Algorithm 11, we illustrate a pseudo code to draw the method control flow graph.

Algorithm 11: Method Control Flow Graph Algorithm

Input 1: returned values from Algorithm 10.
Input 2: the generated xml file.

- Insert the start node for each < function > tag followed by an edge.
- Insert branches for each of the following tags < condition >, <expr_stmt>, < do >, < try >and the < break> tag, the break statement contain no branch.
- Insert the end node for each < /function > tag.
- Apply ruled defined in Table 3 below.
- Define ED as total number of edges.
- Define ND as total number of nodes.
- Perform the following equation:
- $CC = ED - ND + 2$

Return :

Method control flow graph.

Method name.

Cyclomatic complexity CC.

Program length.

Program vocabulary.
Volume.

Difficulty.

Program effort.

Algorithm 11, performs the operations that draw the method control flow graph depending on a list of rules defined in Table 3, and also it depends on both Cyclomatic complexity measures that are discussed in Lines (12-17), and Halstead Complexity measures that is taken as input to represent the method control flow graph, where the main elements of this measure are both nodes and edges.

Method is the main element in MCFG, so, start node and end node are inserted into the graph according to the <function> tag, this is shown in Lines (3, and 7) insertion , the branches insertion according to the XML file are shown in Lines (4, and 5). At the end the main elements that shape method control flow graph are shown in Lines (21-32). Figure 10, shows MCFG format.

Table 3
Rules To draw Branch Statements in the Generated MCFG

Statement Type	Rule
If statement	The node of if statement contains two branches, the true condition node and the false condition.
Try statement	The node of the try statement contains as many branches as catch plus try statement and if finally statement found as many try and catch statement as edges between them point to the finally node.
For statement	Each for statement node has two branches, the condition node and the increment node. Draw a forward edge between increment node and condition node. Also, there is an edge between condition node and the last node that have an edge with end node.
While statement	Each while statement node has two branches.
Switch statement	Switch statement node contains a number of branches as many cases and/ or the default. If the case ended with the break, draw an edge between case node and end node; else draw an edge between the node and the next expression statement.
Do- while statement	Draw a forward edge from the while expression statement node to do node.

As shown from Table3, a set of rules are introduced to discuss how the branches and nodes of the MCFG are inserted. Each statement type has a set of rules that are presented to draw the branches that represents this statement. We can see from Table 3, that if statements have two branches, depending on the condition, so this condition may be either true, or false. In try statement the branches are drawn to represent try and catch. Also since switch statement has many cases and/or default, each of them is represented as branch.

In For statement the first branch is the next. And the second one is the inner part of the loop. While statements, and do-while statements are similar to the for loop, but in do-while do node make the different between them.

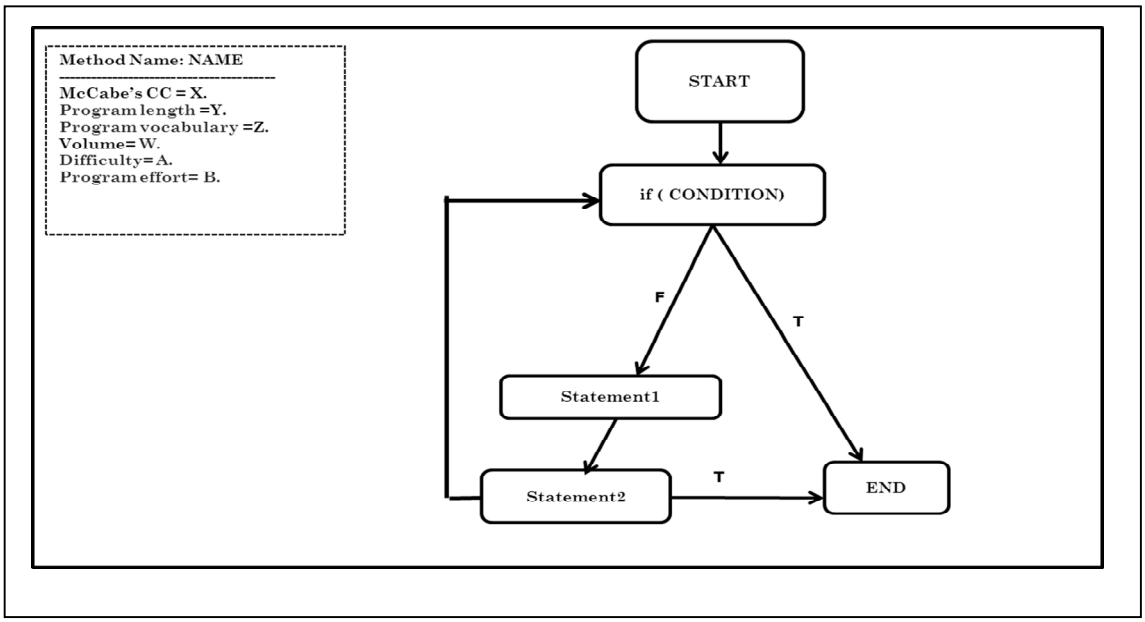

Figure 10
Method Control Flow Graph Format

As shown in Figure 10, the MCFG format represents the method information as number of branches, in this figure the condition has two branches, one of them aims represents the true condition, and the other branch aims to represent the false condition, the true, and false appear on the branch that matches the condition node with the child nodes from each branch. Every statement in the XML file is represented as a node.

Method metrics are represented within a rectangle that appears on the left top corner of the MCFG representation, where it holds both Cyclomatic complexity measures, and Hallstead complexity measures, after mentioning the method name.

3.6.2 Method Control Flow Graph (MCFG) Example

The graphical views that represent the method control flow graph from the proposed methodology contain the some metrics calculation within it. Figure 11, shows an example of the method addFig control flow graph, this method is represented in the class FigInspectorPanel, as Figure 3 shown. The control flow graphs in Figure 11, provides the information about both Cyclomatic complexity which is equals to 2, and Hallstead complexity metrics which are applied to measure the quality of the method addFig. Program length is 79, program vocabulary is 31, the volume of this method is 391.38, difficulty is 54, and the program effort is 7.25.

In Figure 11, the control flow graph that represents addFig method provides this service depending on for loop statement, which is started in the following declaration statement: fig = f, that is represented in the first node of the tree, followed by the condition node which has two child's, the true condition that perform two expression statements each one represented in a node, then if statement which also depends on a condition represented

in a node and its two child each of them also represented in a node, both if condition child nodes when they end they go to the end node in the control flow graph when true, or false condition occurs.

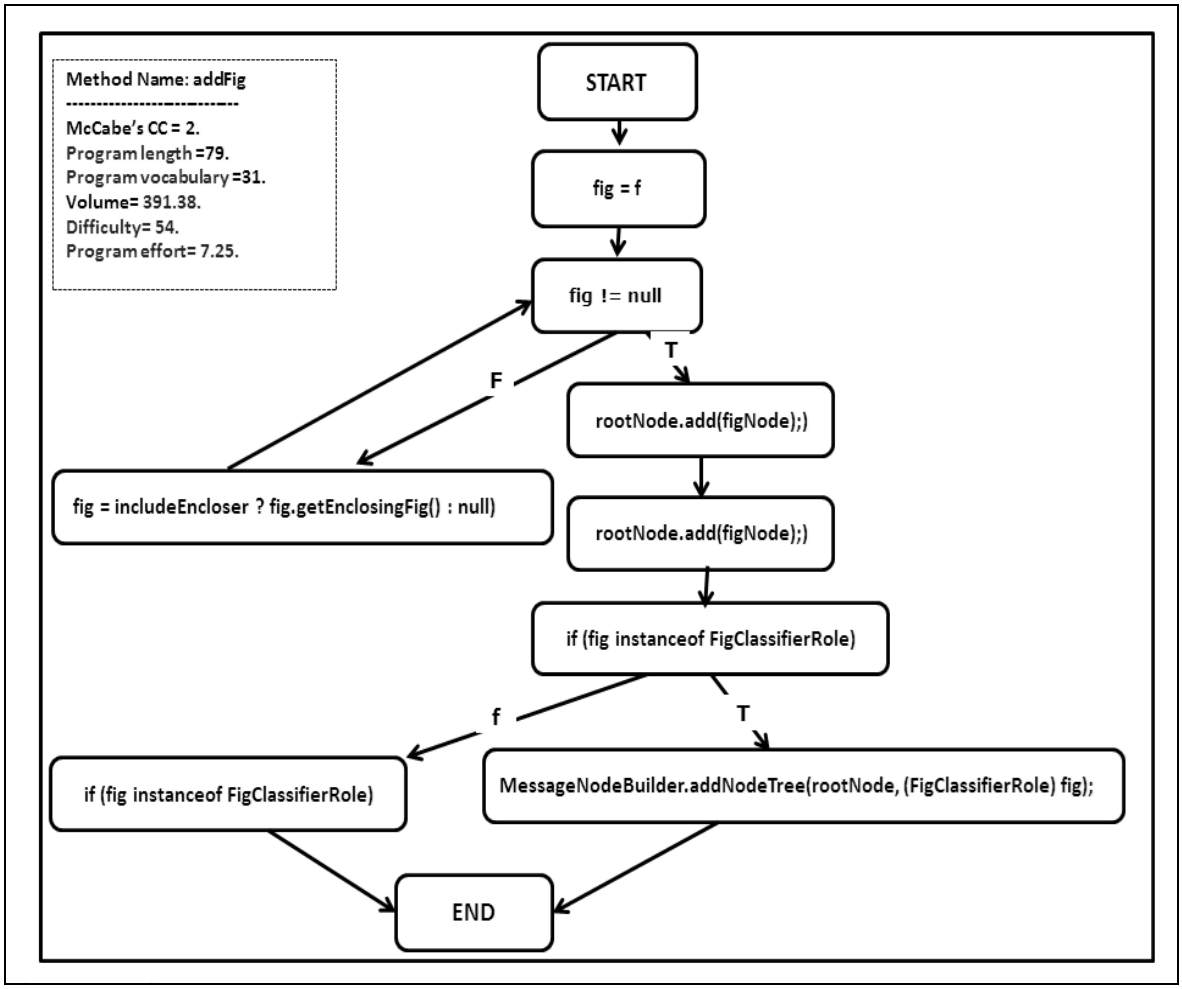

Figure 11
The Generated Method Control Flow Graph For addFig Method from Class FigInspectorPanel

العنوان:	Automatic Generation of Descriptive Summary for Source Code Artifact
المؤلف الرئيسي:	البطوش، أماني عبدالسلام
مؤلفين آخرين:	حماد، مصطفى محمد(مشرف)
التاريخ الميلادي:	2016
موقع:	مؤتة
الصفحات:	1 - 86
رقم MD:	951089
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة مؤتة
الكلية:	عمادة الدراسات العليا
الدولة:	الاردن
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، برمجية جافا، برمجية سي
رابط:	https://search.mandumah.com/Record/951089

Chapter 4
Experimental study and conclusion

In this chapter, we introduce two case studies that aim to discuss the proposed methodology. And in order to discuss the results of the generated descriptive summary for the proposed methodology, Table 4 lists the previous approaches that focuses on generating descriptive summaries for the source code artifact, the input artifact from the target source code, and the output that represents the type of descriptive summary from the input artifact from the previous approaches, and the proposed approach.

In this subsection there are two case studies are illustrated to discuss the proposed methodology, the first case study summarizes the Junit 3.8.1. (Slashdot, 2016) open source code, which has twelve organized packages. And the second case study are applied to summarize the open source code UMLGraph-5.7_2.3 (License, 2014), where it has just two packages. For each case study the generated descriptive summary contains the textual report that appears as one package report, ten class reports. And the graphical representation that appears as ten class call graphs, and ten method control flow graphs. The following sub-sections discuss in details the experimental study for each case study.

4.1 Experimental study

In order to discuss the proposed methodology, two java open source code project are summarized. The first case study is Junit 3.8.1, this open source project is a framework that is used to write repeatable tests. It is large size project. The second case study is UMLGraph-5.7_2.3, a java open source code project that is used to draw UML class, and sequence diagrams automatically. It is a small size project. The following subsections are introduced to discuss the case studies in details.

4.1.1 Case study 1

The first case study discussed according to the proposed methodology is the Junit 3.8.1(Slashdot, 2016), Java open source code, in order to generate a descriptive summary. This generated summary is hydride of texts, graphs, and numerical measures. The summarized artifacts from this case study are discussed in Figure 11, the described sub sections of case study 1 are part from the generated descriptive summary, where it is shown in Appendix 1.

Packages	Classes	Methods
awtui	*awtui/ ProgressBar*	*framework/ Assert /assertTrue*
extensions	*framework/ Assert*	*framework/ Assert / assertFalse*
framework	*framework/ AssertionFailedError*	*framework/ Assert / fail*
runner	*framework/ ComparisonFailure*	*framework/ Assert / assertEquals*
samples	*framework/ TestFailure*	*framework/ Assert / assertNotNull*
samples.money	*framework/ TestResult*	*framework/ Assert / assertNull*
swingui	*framework/ TestSuite*	*framework/ Assert / assertSame*
tests	*extinstion/ RepeatedTest*	*framework/ Assert / assertNotSame*
tests.extensions	*swingui/ AboutDialog*	*framework/ Assert / failNotEquals*
tests.framework	*runner/LoadingTestController*	*framework/ Assert / failNotSame*
tests.runner		
textui		

Figure 11
The Target Software Artifacts from Junit Open Source Code that are used to Generate the Descriptive Summary

4.1.1.1 Package report

In package report we aims to give a brief description about the software, this description includes information about each package, these information are focused on giving a quantity number for each package such as: total number of classes, total number of constructors, total number of methods, and the total number of attribute within each package.

Figure 12 shows the first generated report, the package report for the Junit open source code. It shows that, the generated package report starts

with the package name, followed by the total numbers for each classes, methods, constructors, and attributes. This report lists all the packages that form the software. This report starts to summarize the package awtui, where it has 4 classes, 22 methods, 4 constructors, and 27 attributes. And ended by summarizing the package textui, where it has 2 classes, 24 methods, 4 constructors, and 6 attributes.

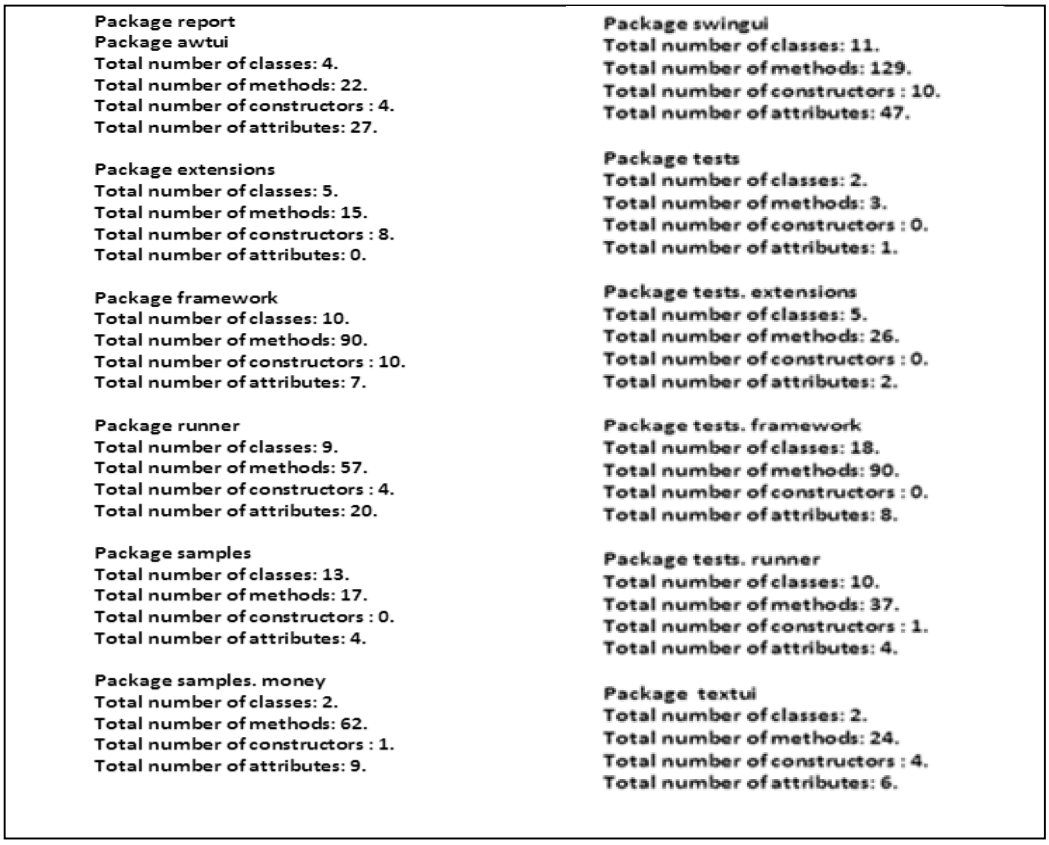

Figure 12
The Generated Package Report for Junit Open Source

4.1.1.2 Class report

Class report provides a textual descriptive summary that summarize the class with the services that class provides. This subsection shows the generated class report for Assert class of the package framework. Figure 11, is the second generated report from the proposed methodology, it summarize Assert class in framework package.

Class Assert report:
Class Assert declared in package framework as public.
This class provide the following services:

The service is: assertTrue. This service returns void. The service uses local data:(message with type String, and condition with type boolean).
This service use local method: fail.

The service is: assertTrue. This service returns void. The service uses local data:(condition with type boolean).
This service use local method: assertTrue.

The service is: assertFalse. This service returns void. The service uses local data:(message with type String, and condition with type boolean).
This service use local method: assertTrue.

The service is: assertFalse. This service returns void. The service uses local data:(message with type String, condition with type boolean).

The service is: assertFalse. This service returns void. The service uses local data:(with type boolean).
This service use local method: assertFalse.

The service is: fail. This service returns void. The service uses local data:(message with type String).
This service use local method: AssertionFailedError.

The service is: fail. This service returns void.
This service use local method: fail.

The service is: assertEquals. This service returns void. The service uses local data:(message with type String, expected with type Object, and actual with type String).
This service use local method: failNotEquals.

The service is: assertEquals. This service returns void. The service uses local data:(expected with type Object, and actual with type Object).
This service use local method: assertEquals.

The service is: assertEquals. This service returns void. The service uses local data:(message with type String, expected with type String, and actual with type String).
This service use the method: ComparisonFailure.

The service is: assertEquals. This service returns void. The service uses local data:(expected with type String, and actual with type String).
This service use local method: assertEquals.

The service is: assertEquals. This service returns void. The service uses local data:(message with type String, expected with type double, actual with type double, and delta with type double).
This service use local method: failNotEquals.
This service use methods: isInfinite.

The service is: assertEquals. This service returns void. The service uses local data:(expected with type double, actual with type double, and delta with type double).
This service use method: failNotEquals.

The service is: assertEquals.This service returns void. The service uses local data:(message with type String, expected with type float, actual with type float, and delta with type float).
This service use local method: failNotEquals.
This service use methods: isInfinite.

The service is: assertEquals. This service returns void. The service uses local data:(expected with type float, actual with type float, and delta with type float).
This service use local method: assertEquals.

The service is: assertEquals. This service returns void. The service uses local data:(expected with type long, and actual with type long).
This service use local method: assertEquals.

The service is: assertEquals. This service returns void. The service uses local data:(message with type String, expected with type boolean, and actual with type boolean).
This service use local method: assertEquals.

The service is: assertEquals. This service returns void. The service uses local data:(expected with typ boolean, and actual with type boolean).
This service use local method: assertEquals.

The service is: assertEquals. This service returns void. The service uses local data:(message with type String, expected with type byte, and actual with type byte).
This service use local method: assertEquals.

The service is: assertEquals. This service returns void. The service uses local data:(expected with type byte, and actual with type byte).
This service use local method: assertEquals.

The service is: assertEquals. This service returns void. The service uses local data:(message with type String, expected with type char, and actual with type char).
This service use local method: assertEquals.

The service is: assertEquals. This service returns void. The service uses local data:(expected with type char, and actual with type char).
This service use local method: assertEquals.

The service is: assertEquals. This service returns void. The service uses local data:(message with type String, expected with type char, and actual with type short).
This service use local method: assertEquals.

The service is: assertEquals. This service returns void. The service uses local data:(expected with type char, and actual with type short).
This service use local method: assertEquals.

This service is: assertEquals. This service returns void. The service uses local data:(message with type String, expected with type int, and actual with type int).
The service use local method: assertEquals.

The service is: assertEquals. This service returns void. The service uses local data:(expected with type int, and actual with type int).
This service use local method: assertEquals.

The service is: assertNotNull. This service returns void. The service uses local data:(object with type Object).
This service use local method: assertNotNull.

The service is: assertNotNull. This service returns void. The service uses local data:(message with type String, object with type Object).
This service use local method: assertTrue.

The service is: assertNull. This service returns void. The service uses local data:(object with type Object).
This service use local method: assertNull.

The service is: assertNull. This service returns void. The service uses local data:(message with type String, and object with type Object).
This service use local method: assertTrue.

The service is: assertSame. This service returns void. The service uses local data:(message with type String, expected with type Object, and actual with type Object).
This service use local method: failNotSame.

The service is: assertSame. This service returns void. The service uses local data:(expected with type Object, and actual with type Object).
This service use local method: assertSame.

The service is: assertNotSame. This service returns void. The service uses local data:(message with type String, expected with type Object, and actual with type Object).
This service use local method: failSame.

The service is: assertNotSame. This service returns void. The service uses local data:(expected with type Object, and actual with type Object).
This service use local method: assertNotSame.

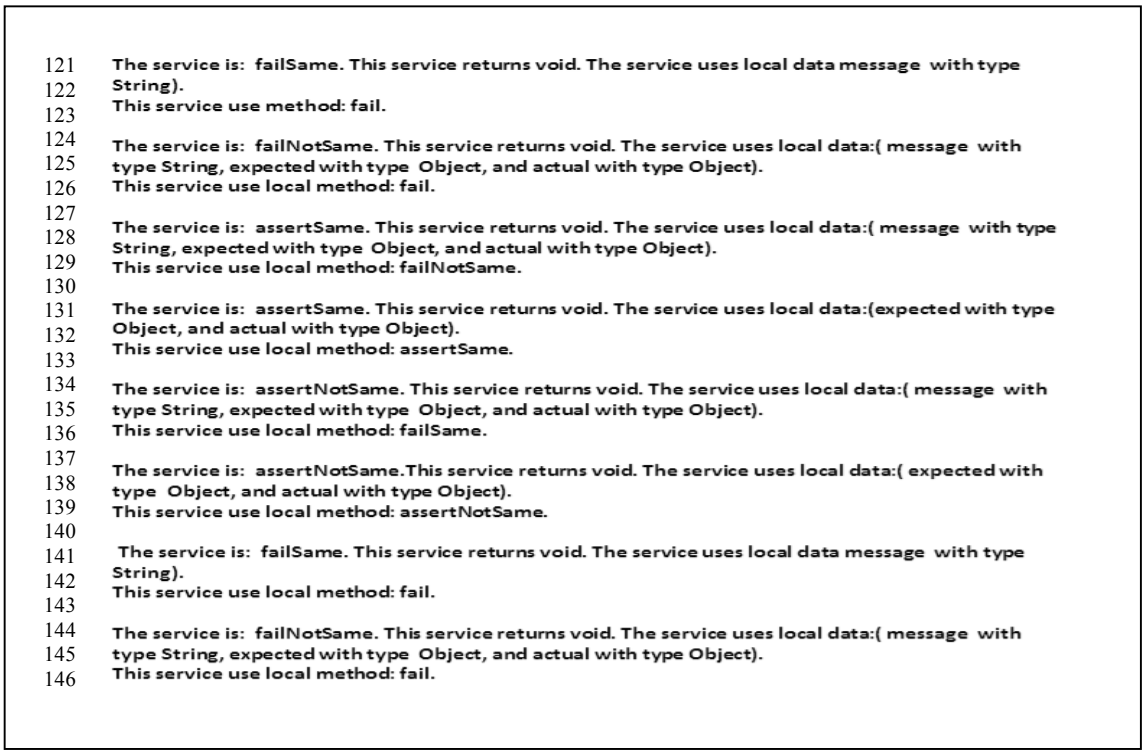

Figure 12
The Generated Class Report From Assert Class

As shown from Figure 12, Assert class report starts by describing general information about the class Assert, which shows that class Assert declared as a public class in the package framework, and it has no super class. There are 42 method within assert class, so each method provides a service. The first service that Assert class provides is assertTrue that is shown by line 4, this service is also provided another time in line 7, but this service uses different local data.

The service assertEquals provides to Assert class more than one time, in each time it uses different number and different type of local data. The last service that Assert class provides is failNotSame that is shown in line 144, this service returns void, and uses three local data: message, expected, and actual. It also depends on fail method, since it uses it as a local method.

4.1.1.2 Class Call Graph (CCG)

Figure 13, shows Assert class call graph, that is the third part from the generated summary. In the generated class call graph, we note that the method assertequals invokes failnotequals method three times. Failname method invoked by assertNotsame method within the same class Assert. AssertionFailedError method doesn't defined in Assert class, but it is invoked by fail method. So it is shown as dotted rectangle. The total numbers of class assert methods equals: 34. Are represented on left top of the class call graph.

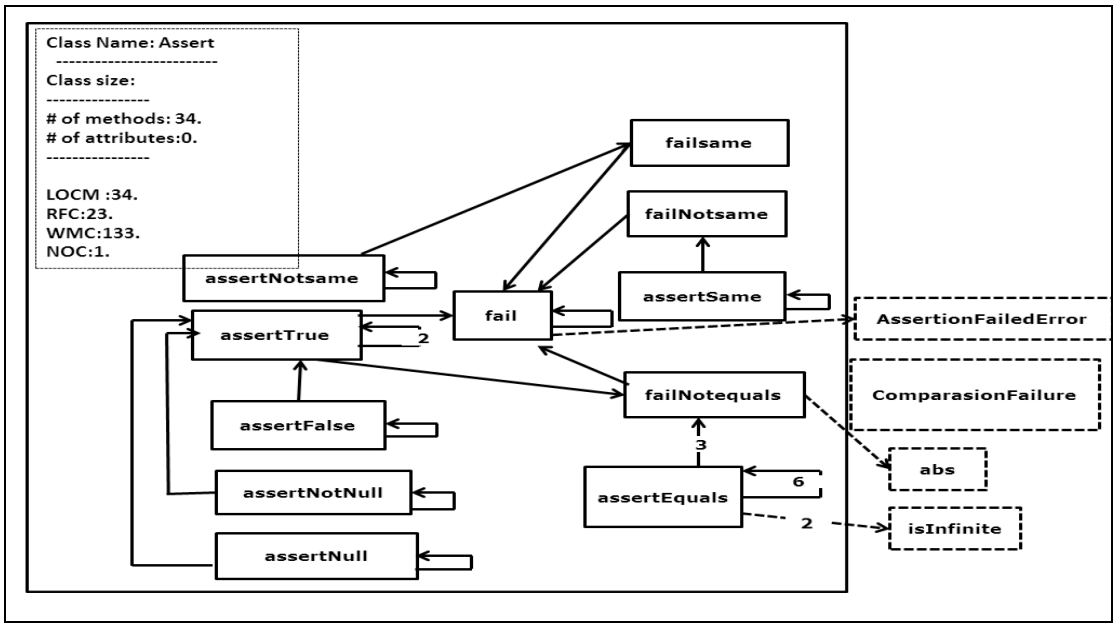

Figure 13
The Generated Class Call Graph For Assert Class

4.1.1.3 Method Control Flow Graph (MCFG)

In this subsection, an example for the fourth part of the generated summary, which summarize assertTrue method control flow graph. It is represented in Figure 14, where assertTrue is service provided by Assert class, in this figure, if statements have two child nodes, the true condition and the false condition, for the assertTrue method, if the condition is true it calls the method fail, else it returns. The method metrics for this method are shown in the left top of the figure.

The calculated method metrics for assertTrue method shown as follows: the Cyclomatic complexity is shown by McCabe's CC = 2. The Hallstead complexity is shown by the following results: Program length is 27, Program vocabulary is 6, Volume is 28.53, Difficulty is 1.6, and the Program effort is 17.83.

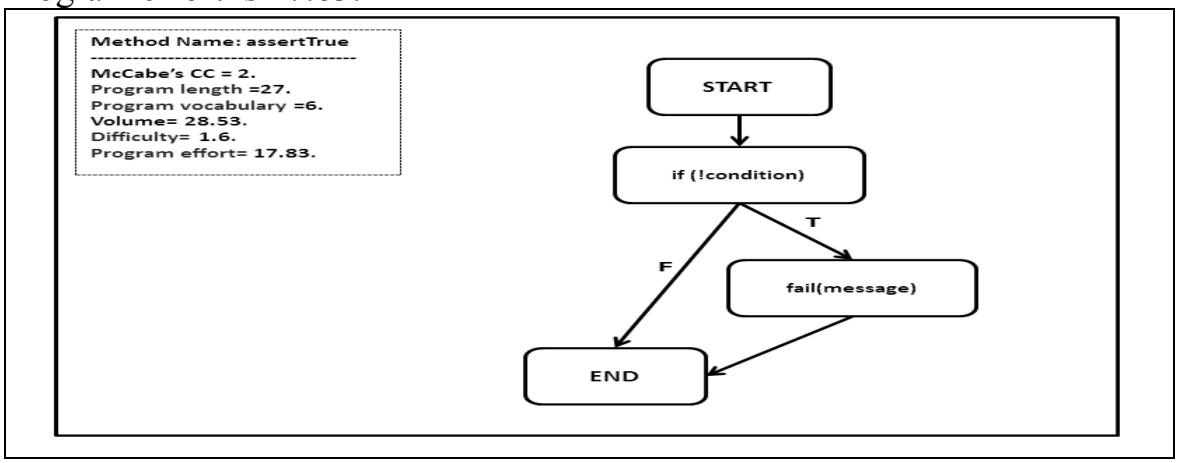

Figure 14
The Generated Method Control Flow Graph For assertTrue Method

4.1.2 Case Study 2

The second case study discussed according to the proposed methodology is the UMLGraph-5.7_2.3 (License, 2014), the Java open source code, in order to generate a descriptive summary. This generated summary is hydride of texts, graphs, and numerical measures. This case study is discussed in Figure 11. The generated descriptive summary of case study 2, is described as a subsection, and shown in Appendix2.

Packages	Classes	Methods
doclet *test*	*doclet/ ClassInfo* *doclet/ ClassGraphHack* *doclet/ ContextMatcher* *doclet/ DevNullWriter* *doclet/ ContexView* *doclet/ Shape* *doclet/ RelationPattern* *test/ RunDoclet* *test/ TestUtils* *test/ RunOne*	*doclet/ ClassInfo / addRelation* *doclet/ ClassInfo / main* *framework/ Assert / matchesOnes* *framework/ Assert / addRelation* *framework/ Assert / addToGraph* *framework/ Assert / runDoclet* *framework/ Assert / cleanFolder* *framework/ Assert / cellBorder* *framework/ Assert / graphvizAttribute*

Figure 15
The Target Software Artifacts from UMLGraph Open Source Code that are used to Generate the Descriptive Summary

4.1.2.1 Package Report

The generated package report for UMLGraph-5.7_2.3 open source code is shown in Figure 16, which is described as a textual report that provides the quantities calculation for each package within the target software.

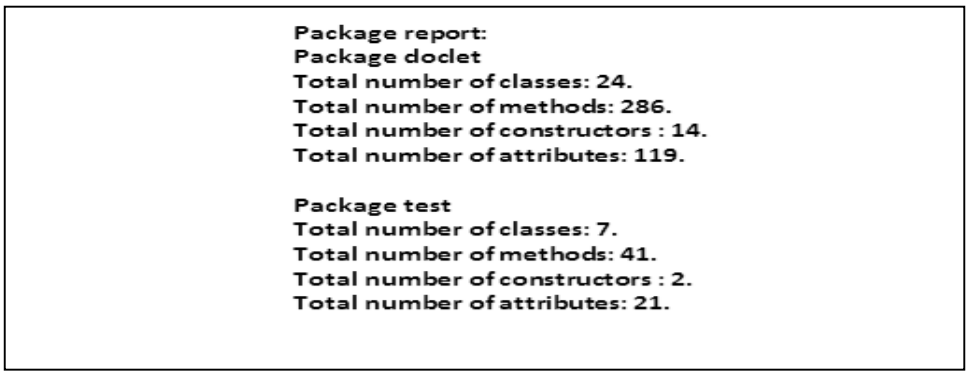

Figure 16
The generated Package Report For UMLgraph Open Source Code packages

4.1.2.2 Class Report

Class report provides a text summary description about the class with the services that class provides. A list of figures below shows the generated class report for some classes of UMLgraph Open Source Code. Figure 17, shows ClassInfo class report.

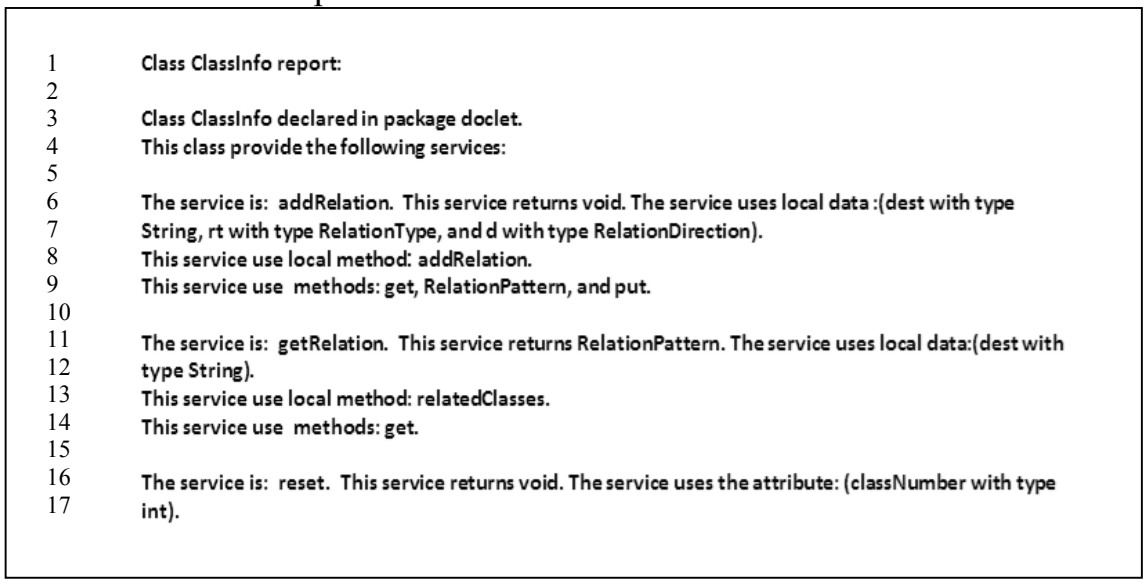

Figure 17
The Generated Class Report for ClassInfo Class

As shown in class report in Figure 44, ClassInfo class is declared in package doclet, this class provides three services, the first service is addRelation which returns void, this service depends on dest, rt, and d data type. addRelation service depends on the methods: get, relationPattern, and put.

4.1.2.3 Class Call Graph (CCG)

In this sub-section the classes that are reported in section 4.1.1.2 are viewed as class call graph for each one of them.

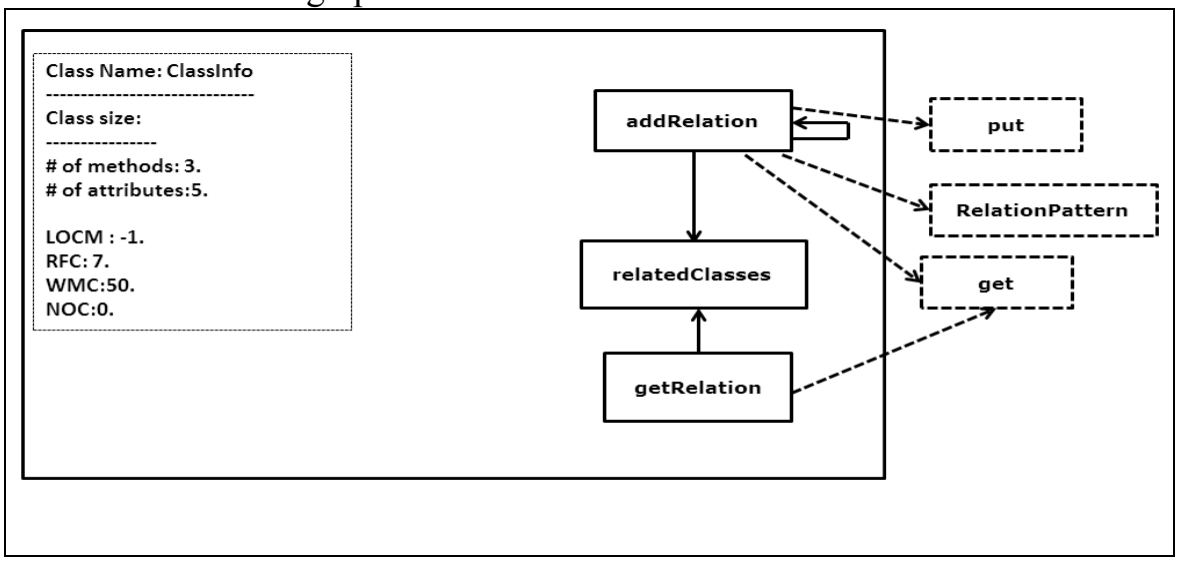

Figure 18
The Generated Class Call Graph for ClassInfo Class

ClassInfo class call graph that is viewed in Figure 18 that was reported in Figure 17. ClassInfo class consists of three methods, each method in this class participates in an invocation, either it invokes another method, or it is invoked by another method. For example relatedClasses method is invoked by both getRelation method, and addRelation method. Also the method addRelation perform a recursive call, and it is clearly shown in this figure.

The method RelationPattern, put, and get are represented as a dotted rectangles, and occur outside the class, which indicate that they are not local method. The total number of attributes in this class is 5, ClassInfo metrics shows that this class has no children, the LOCM is 3, RFC is 7, and the WMC is 50, NOC is 0.

4.1.2.4 Method Control Flow Graph (MCFG)

This subsection views addRelation method control flow graph in Figure 19, this method is reported in the ClassInfo class report within Figure 16.

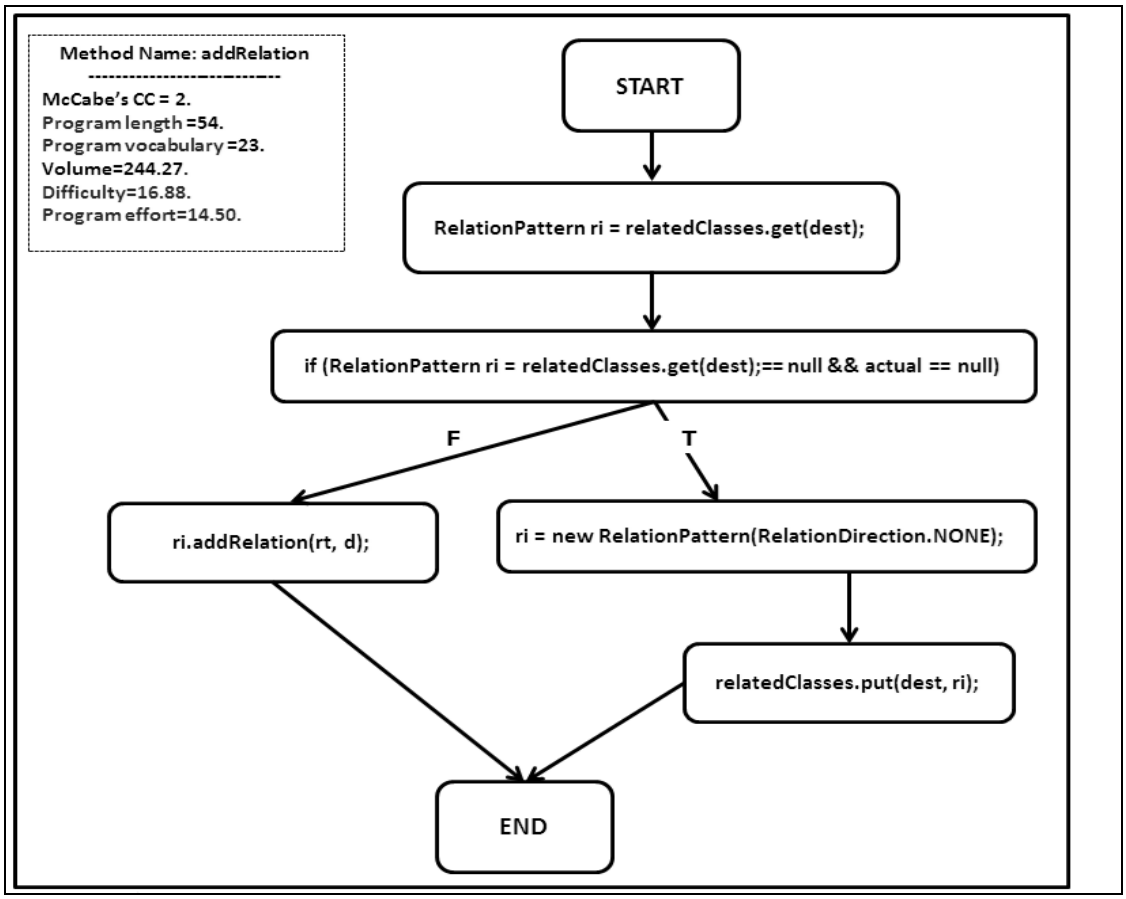

Figure 19
The Generated Method Control Flow Graph For addRelation Method

The generated method control flow graph for addRelation method, that is shown in Figure 19, starts with a declaration statement that is added after the start node, then if statement node, is added when the condition is true it performs the statement " ri= new RelationPattern(RelationDirection.NONE); then the statement relatedClasses.put(dest,ri); is shown as a branch. And if the condition is false, the branch ri.addRelation(rt, d); is performed.

4.2 Source Code Artifacts Summarization Approaches and Results Discussions

In this sub-section we list some of the previous approaches that introduce a descriptive summary for the source code artifact, in order to discuss the proposed approach results by comparing them with the previous approaches. These approaches are summarized by Table 4. The aim of introducing this sub-section to show that our generated descriptive summaries depend on the previous evaluated approaches to enhance the generated summaries.

From Table 4, and Table 5, the proposed approach input software that contains packages, classes, and methods. This software is analyzes

statically using srcML tool that provides more information about the parsed source code for AST as a data format, and as a structured document srcML it is directly supports representing multiple levels within the AST, In order to generate a descriptive summary which is hydride of texts, graphs, and numerical measures from the analyzed software.

The reason that method report is not introduced as a single report that class report adds all method descriptions together, and also the method control flow graph that introduced to each method within class cover the un-reported information about each method within the summarized class as a graphical representation.

Our primary goal of the proposed approach is to generate automatic descriptive summary for source code artifact, so this study aims to examine the following research questions:

RQ1- Does the generated descriptive summaries summarize the software and describes, and identify the source code artifacts (package, class and method) automatically?

We can clearly see that the proposed approach describes and identifies the following source code artifacts: the package report was generated and identifying the software packages automatically since it presents a view about the size of the software, and also it provides a general view about the software.

The class report, and class call graph display information about each class, they show the main content for each class, class size that is presented with a total number of methods, and total number of attributes, and the main class metrics that measures the quality of each class. At the end the method control flow graph that provides an important view for each method within the class supported by the main method metrics that measures the quality of each method.

RQ2- Does the generated descriptive summaries reflect the developers' understanding of the software?

To answer this question we return to the previous evaluation methods that were applied on the generated summaries, which shows a good result. And since the proposed approach is similar to them, this indicates that it will also provide a good results in reflecting the developers understanding of the software. Especially that it provides a descriptive summary that is represented in textual, and graphical views, and it also covers more than one granularity level within the software.

Table 4
Our Methodology vs others

	Work	Methodology
	Proposed Methodology	Static analysis depending on srcML tool, Metrics trace technique, Visualization techniques, Querying technique, and Feature locating techniques
1.	(Moreno, 2013)	Static analysis based on AST and method call
2.	(Sridhara G. e., 2010)	Static analysis depending on AST, and CFG, and the natural language analysis
3.	(McBurney, 2014)	Static analysis based on AST
4.	(Sridhara G. e., 2010)	Static analysis based on AST, and natural language analysis
5.	(Graham, 2004).	Metrics trace technique
6.	(Ellina, 2007).	analyzing the python source code statically
7.	(Alimucaj, 2009).	Static analysis for java method depending on the AST
8.	(Myers, 2011)	Static analysis for java method
9.	(Gerald Kaszuba, 2007).	Dynamic analysis for java class
10	(Hammad, 2016)	Static analysis for java package depending on srcML tool
11.	(Lanza, 2011)	Visualization technique to visualize software elements as a city in 3D view
12.	(Lanza, M, 2001)	Visualization technique

Table 5
Our Methodology inputs and outputs vs others

	Work	Input			Output						
					Report						
		Package	Class	Method	Package	Class	Method	CCG	MCFG	Software metrics	
	Proposed Methodology	√	√	√	√	√		√	√	√	
1.	(Moreno, 2013)		√			√					
2.	(Sridhara G. e., 2010)			√			√				
3.	(McBurney, 2014)			√			√				
4.	(Sridhara G. e., 2010)		√				√			√	
5.	(Graham, 2004).	√						√			
6.	(Ellina, 2007).			√					√		
7.	(Alimucaj, 2009).			√					√		
8.	(Myers, 2011)		√					√			
9.	(Gerald Kaszuba, 2007).		√					√			
10.	(Hammad, 2016)	√			√						
11.	(Lanza, 2011)	√	√	√						√	
12.	(Lanza, M, 2001)		√							√	

From Table 4, and 5, the previous summarization approaches focus in generating one type of summary, the generated summary either provide a report, this report is shown internally as a comment or externally which depends on either syntax or the semantics of the summarized source code

artifact. Works (2, 3, 4, and 5), aim to generate a natural language description summary that provides a descriptive summarizes for Java classes, or Java methods.

In work 2 the generated descriptive summary for Java class include a general description that provides a general idea, and a description about the behavior of each class, which are described as natural language sentences depending on the selected methods. The evaluation of this research shows that Expressiveness with the category: the summary is easy to read and understand gives 68%. And Content adequacy with the category: Not missing any information gives 45%.

On the other hand the generated class report for the proposed approach provide a general description for the class, it also adds to the general information the package name, and for each method summary within the class report it adds the local method invocation, external method invocation, local data with their type, and the attribute access, and their type for each method. This shapes also the behavior of the class depending on the provided services from methods of each class.

Works (3, 4, and 5), aims to summarize Java methods. The generated descriptive method summaries is enhanced over those researches, the first evaluation results from those researches was from Work 5, that gives the conciseness rate with catagory has no unnecessary information 23%, and content adequacy rate with the category not missing any information 36%. The proposed methodology covered the method summary within the generated class report, and also generates method call graph that is supported with more semantic information's that are hold within both Cyclomatic complexity, and Hallstead complexity measures.

As mentioned previously the propoesd approach provides more than one view to summarize source code artifacts. The second view presented from the proposed methodology provides the class call graph that is supported by class size information shaped in total number of methods, and total number of attributes followed by the following class metrics: LOCM, RFC, WMC, and NOC.

While the previous works (7,9, and 10), introduce the class call graph just to provide the method invocation within each class, or the method invocations with the number of these invocations. Also works (6,12, and 13), aim to show the class metrics such as LOCM, class coupling, inheritance level metrics, number of methods within class, number of attributes as a visualized shape, or systems without analyzing the source code. But the proposed approach depends on analyzing the source code to calculate the class metrics and then viewing them within the class call graph for each class within the software.

The third view proposed by this approach provides the method control flow graph to each method within each class of the software is supported

with both Cyclomatic complexity, and Hallstead complexity measures, that aims to measure method quality, while the previous approaches introduce the method control flow graph in work 9, provides the control flow graph for Java method, with the Cyclomatic complexity measure for each one.

By answering the research questions, and discussing the previous approaches that aims to provide a descriptive summary for source code artifacts, we find that the proposed methodology shows that generating a descriptive summary for the following target software artifacts: packages, classes, and methods in more than one view, make it more easy, and understandable for who concern with the software, to understand the target software in the way that he find it more suitable.

4.3 Conclusion and Future Work

A new approach for generating a descriptive summary for the source code artifacts, in more than one view, is proposed. The generated descriptive views consist of text and graph based information. Textual based reports provide syntactic information project's packages and classes. The class call-graph view is generated for each class with a number of class's metrics, such as LOC, RFC, NOC, and WMC. The method control-flow graph is produced for each method with Cyclomatic complexity measures, and the Hallstead complexity measures. The main metrics in both CCG and MCG hold the semantics information within both class and method.

Summarizing the software and presenting the extracted information for each package, class, and method in more than one view that covers these granularities of the target source code, makes the source code more understandable and maintainable. Two case studies are applied for the proposed approach. The generated reports and views showed that they can support software comprehension and can be seen as a reverse engineering analysis for the source code.

We are planning to extend our work by enhancing the package report to cover the relationships among the package's classes. Moreover, measuring the effectiveness of the proposed approach, and implementing a full feature GUI, for the proposed approach are other goals for the future work.

العنوان:	Automatic Generation of Descriptive Summary for Source Code Artifact
المؤلف الرئيسي:	البطوش، أماني عبدالسلام
مؤلفين آخرين:	حماد، مصطفى محمد(مشرف)
التاريخ الميلادي:	2016
موقع:	مؤتة
الصفحات:	1 - 86
رقم MD:	951089
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة مؤتة
الكلية:	عمادة الدراسات العليا
الدولة:	الاردن
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، برمجية جافا، برمجية سي
رابط:	https://search.mandumah.com/Record/951089

الملخص

انشاء ملخص وصفي للبرمجيات بشكل تلقائي

اماني عبد السلام البطوش

جامعة مؤتة، 2016

احتل تلخيص البرمجيات مساحة واسعة من الاهتمام في مجال هندسة البرمجيات ، حيث هدف تلخيص جزئيات البرنامج إلى التقليل من الوقت والجهد في مرحلة التطوير، إضافة إلى توفير طريقة سهلة لفهم البرنامج. و تقترح هذه الدراسة تقديم أكثر من طريقة بهدف تلخيص جزئيات البرنامج المستهدف و في أكثر من مستوى.

اعتمدت هذه الرسالة على تطبيق أسلوب التحليل الثابت للبرنامج بهدف توليد ملخص وصفي بسيط و سهل للمشاريع المكتوبة بلغتي البرمجة سي و الجافا . حيث أن الملخص المقترح هو عبارة عن مجموعة من التقارير التي تصف كل من (package, class) للمشروع، علاوة على ذلك تم إنشاء واجهة رسومية لclass ممثله ب class call graph والذي يتضمن أهم القياسات الخاصة به. اضافة لذلك تم تمثيل كل method ضمن المشروع من خلال واجهة رسومية تدعى method control flow graph والتي تحتوي ايضا على أهم القياسات التي تمثلها.

تم تنفيذ طريقة التحليل الثابت على البرمجية، من خلال توليد ملف XML للمشروع المستهدف والذي تم تمثيله على شكل بنيه هرميه، حيث تم تحليله من خلال استعلامات XPath. هذه الاستعلامات تم تحديدها بهدف استخلاص الخصائص المرجوة من الجزئية المراد تلخيصها في البرنامج.

تم تطبيق الدراسة التجريبية على مشروعين مختلفين من مشاريع برمجية جافا بهدف اختبارها، حيث أظهرت نتائج الطريقة المقترحة أنها يمكن أن تكون مفيدة في استخلاص المعلومات المعقدة من البرنامج بطريقة منظمة وتقديم هذه المعلومات للمطورين في عدة مستويات تجريدية و بأكثر من أسلوب سلس.

Abstract
Automatic Generation of Descriptive Summary for Source Code Artifact

Amani Al-Btoush
Mutah University, 2016

Source code summarization occupied a wide area of interest in the field software engineering. Summarizing source code artifacts reduce time and effort in the maintenance stage and provide easy way to comprehend the software. This study proposes an automatic approach to summarize the target source code in different views and levels.

The proposed approach applied static analysis techniques on the source code to generate simple and easy use descriptive summary for projects written in Java, and C programming languages. The proposed summary is a collection of a set of reports that describe the project's packages and classes. Moreover, for each class, a call graph for its methods is generated with the values of the main class's metrics. Furthermore, a control flow graph for each method is generated with the values of method's metrics.

الصفحةXIIITwo experimental studies are applied on two different java open source projects to test the proposed methodology. From these two case studies, the proposed approach showed that it can be useful and helpful in extracting complex information about the source code in a systematic way and present it in abstract levels with different friendly ways for the developers.

العنوان:	Automatic Generation of Descriptive Summary for Source Code Artifact
المؤلف الرئيسي:	البطوش، أماني عبدالسلام
مؤلفين آخرين:	حماد، مصطفى محمد(مشرف)
التاريخ الميلادي:	2016
موقع:	مؤتة
الصفحات:	1 - 86
رقم MD:	951089
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة مؤتة
الكلية:	عمادة الدراسات العليا
الدولة:	الاردن
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، برمجية جافا، برمجية سي
رابط:	https://search.mandumah.com/Record/951089

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF ALGORITHMS

العنوان:	Automatic Generation of Descriptive Summary for Source Code Artifact
المؤلف الرئيسي:	البطوش، أماني عبدالسلام
مؤلفين آخرين:	حماد، مصطفى محمد(مشرف)
التاريخ الميلادي:	2016
موقع:	مؤتة
الصفحات:	1 - 86
رقم MD:	951089
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة مؤتة
الكلية:	عمادة الدراسات العليا
الدولة:	الاردن
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، برمجية جافا، برمجية سي
رابط:	https://search.mandumah.com/Record/951089

Mutah University
College of Graduate Studies

Automatic Generation of Descriptive Summary for Source Code Artifact

By
Amani Abdel-Salam Al-Btoush

Supervisor:
Dr. Mustafa Hammad

A thesis Submitted to the College of Graduate Studies in partial fulfillment of the requirements for the Master's degree in Computer science to the Department of Information Technology, University of Mutah.

Mutah University, 2016

بسم الله الرحمن الرحيم

MUTAH UNIVERSITY
College of Graduate Studies

جامعة مؤتة
كلية الدراسات العليا

نموذج رقم (14)

قرار إجازة رسالة جامعية

تقرر إجازة الرسالة المقدمة من الطالبة اماني عبدالسلام البطوش الموسومة بـ:

Automatic Generation of Discriptive Summary For source Code Artifact

استكمالاً لمتطلبات الحصول على درجة الماجستير في **الحاسوب**.

القسم: تكنولوجيا المعلومات.

	التوقيع	التاريخ	
د. مصطفى محمد حماد		26/4/2016	مشرفاً ورئيسا
د. عوني منصور الحموري		26/4/2016	عضواً
د. محمد موسى النبهان		26/4/2016	عضواً
د. أحمد خضر أحمد		26/4/2016	عضواً

عميد الدراسات العليا

د. محمد المحاسنة

MUTAH-KARAK-JORDAN
Postal Code: 61710
TEL :03/2372380-99
Ext. 5328-5330
FAX:03/ 2375694
e-mail: dgs@mutah.edu.jo sedgs@mutah.edu.jo
http://www.mutah.edu.jo/gradest/derasat.htm

مؤته – الكرك – الاردن
الرمز البريدي :61710
تلفون :03/2372380-99
فرعي 5328-5330
فاكس 03/2 375694
البريد الالكتروني
الصفحة الالكترونية

Dedication

This work is dedicated to all my family members, especially my beloved husband Zeiad who gave me the strength and the courage to keep moving forward and achieve my goals, and my children Hala, Tareq, and leen. It is also dedicated to my parent for their great support.

Acknowledgments

I thank Allah for giving me the ability to accomplish what I have achieved in this thesis. I would like also to thank my supervisor, Dr Mustafa Hammad, for his patience, support, inspiration and persistence which has been the driving force that has enabled me to complete this project, and for the valuable time he has spent in advising me step by step.

Amani Abdel-Salam Al-Btoush

ABBREVIATIONS

CC	McCabe's Cyclomatic Complexity
CCG	Class Call Graph
DOM	Document Object Model
IEEE	Institute of Electrical and Electronics Engineers
ISME	Integrated Software Maintenance Environment
JavaML	Java Markup Language
LOCM	Lack Of Cohesion Metric
MCFG	Method Control Flow Graph
OO	Object Oriented
NOC	Number Of Children algorithm
RFC	Response For Class Metric
SGML	the standard Generalized Markup Language
srcML	source code Markup Language
UML	Unified Modeling Language
W3C	World Wide Web Consortium
WMC	Weighted Methods per Class
XML	The Extensible Markup Language

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF ALGORITHMS

الملخص

انشاء ملخص وصفي للبرمجيات بشكل تلقائي

اماني عبد السلام البطوش

جامعة مؤتة، 2016

احتل تلخيص البرمجيات مساحة واسعة من الاهتمام في مجال هندسة البرمجيات ، حيث هدف تلخيص جزئيات البرنامج إلى التقليل من الوقت والجهد في مرحلة التطوير، إضافة إلى توفير طريقة سهلة لفهم البرنامج. و تقترح هذه الدراسة تقديم أكثر من طريقة بهدف تلخيص جزئيات البرنامج المستهدف و في أكثر من مستوى.

اعتمدت هذه الرسالة على تطبيق أسلوب التحليل الثابت للبرنامج بهدف توليد ملخص وصفي بسيط و سهل للمشاريع المكتوبة بلغتي البرمجة سي و الجافا . حيث أن الملخص المقترح هو عبارة عن مجموعة من التقارير التي تصف كل من (package, class) للمشروع، علاوة على ذلك تم إنشاء واجهة رسومية لclass ممثله ب class call graph والذي يتضمن أهم القياسات الخاصة به. اضافة لذلك تم تمثيل كل method ضمن المشروع من خلال واجهة رسومية تدعى method control flow graph والتي تحتوي ايضا على أهم القياسات التي تمثلها.

تم تنفيذ طريقة التحليل الثابت على البرمجية، من خلال توليد ملف XML للمشروع المستهدف والذي تم تمثيله على شكل بنيه هرميه، حيث تم تحليله من خلال استعلامات XPath. هذه الاستعلامات تم تحديدها بهدف استخلاص الخصائص المرجوة من الجزئية المراد تلخيصها في البرنامج.

تم تطبيق الدراسة التجريبية على مشروعين مختلفين من مشاريع برمجية جافا بهدف اختبارها، حيث أظهرت نتائج الطريقة المقترحة أنها يمكن أن تكون مفيدة في استخلاص المعلومات المعقدة من البرنامج بطريقة منظمة وتقديم هذه المعلومات للمطورين في عدة مستويات تجريدية و بأكثر من أسلوب سلس.

Abstract
Automatic Generation of Descriptive Summary for Source Code Artifact

Amani Al-Btoush
Mutah University, 2016

Source code summarization occupied a wide area of interest in the field software engineering. Summarizing source code artifacts reduce time and effort in the maintenance stage and provide easy way to comprehend the software. This study proposes an automatic approach to summarize the target source code in different views and levels.

The proposed approach applied static analysis techniques on the source code to generate simple and easy use descriptive summary for projects written in Java, and C programming languages. The proposed summary is a collection of a set of reports that describe the project's packages and classes. Moreover, for each class, a call graph for its methods is generated with the values of the main class's metrics. Furthermore, a control flow graph for each method is generated with the values of method's metrics.

الصفحةXIIITwo experimental studies are applied on two different java open source projects to test the proposed methodology. From these two case studies, the proposed approach showed that it can be useful and helpful in extracting complex information about the source code in a systematic way and present it in abstract levels with different friendly ways for the developers.

Chapter 1
Introduction

Overview

Keeping up with an enormous amount of source code that you need to read and understand and the lack of summary commits that are made by programmers, are the main challenges faced by today's developers. So in order to help developers deal with this problem and in order to reduce the cost, one solution is to use a simple text description, or simple graphical representation view of the source code features that developers can easily understand. This can also help developers to understand and validate changes, trace changes to other software artifacts, and locate and re (assign) bug reports.

In fact, automatic summarization is one of the oldest research areas dating back to the late 1950s, which is noted in all programming languages starting from FORTRAN that have provided a facility to write comments. However, in recent years there has been an increasing attention to this field from academia, government and industry. The reason is the rapid growth of accessible information resources, mostly the World Wide Web, which has resulted in a well-known problem of information overload (Mani, 1999). The need for automated source code summary represents a main source for system documentation and it is the core for source code understanding with respect to maintenance, development and reducing reuse cost.

Software systems are developed in a number of different phases. The first stage is the analysis of requirements followed by the design of the system in order to meet the requirements. The next step involves writing code in a programming language to implement the design specifications. Finally, the system is tested before it is released for use by an end user. Once the product has been shipped, the system enters a phase known as maintenance. Software maintenance is one of the most time and effort consuming. In software engineering, it means the modification of a software product after delivery to correct faults, in order to improve the performance or other features (Eddy, 2013). Developers during maintenance need quick understand to the source code entities such as (packages, classes or methods), since they cannot read the entire code of large systems. So the identifying will occur efficiently and then they just focus on the ones related to the task at hand. And since the most common two activities to deal with software systems are searching and browsing, the source code with thousands or millions lines of code, source code documentation becomes important.

Also, modifications source code documentation takes place, which are often documented with long messages. Those messages are a key

component of software maintenance; they can help developers locate and triage defects, validate changes, and understand modifications (Haiduc, 2010, & Haiduc S. J., 2010). In maintenance stage software change may occur, so it affects another part of the source code. This requires spending more effort and time from developers to find the affected lines of the source code in order to understand the software.

Software changes are the basic and essential building blocks and characteristic of software evolution in software development since the software systems must respond to evolving platforms, requirements, and other environmental pressures, and after the first version has shipped the software continues to evolve, software evolution offers a different point of view on the traditional about software maintenance: it indicates the idea of essential change within an environment (Godfrey, 2008). Software evolution appeared as an unexpected and unplanned phenomenon that was observed in the original case study, in the evolution step, developers add new features, correct previous mistakes and misunderstandings, and react to the requirements, technologies, and knowledge volatility as it plays out through time. And each change introduces a new feature or some other new properties into software. During evolution, the programmers must comprehend the existing program to be able to add new functionalities or new properties to it (Rajlich, 2014).

In software development, similar problems are solved again and again, so the best career is not to repeat solving of what has been already solved. The best solution here is to reuse the same solution. Software reuse is the use of software knowledge or the existing software in order to build new software. It is also means the reuse of the code (Frakes, 2005). The importance of software reuse comes because the need to reduce effort in software maintenance and development. It also improves the quality of software and decreases time to market (Poulin, 1993). So a good software reuse process facilitates the increase of productivity, reliability, quality, and the decrease of costs and implementation Time. Software systems and components are specific reusable entities, mathematical function or an object class. According to (Selby, 2005) found that a set of programs consist of 32% reused code (not including libraries), so in order to reuse the existing software it is important to understand and document source code.

Software comprehension is the main activity that simplifies maintenance, reuse, code understanding and many other activities in software engineering. (Storey, 2005), so the summary can be one of the techniques that simplify software comprehension , which produce a text that contains a large amount of the information, contained in the original text, and do not exceed half of the original text. Program-comprehension can be categorized into three models: top-down models, bottom-up models, and integrated models. Comprehension according to the top-down model is

working on deriving and formulating hypotheses about program purpose while ignoring details, in order to evaluate them by the developers. Bottom-up comprehension describes how a program is understood when a programmer doesn't have a knowledge about a program's domain, here the programmer checks the statements of a program and groups them into semantic chunks. This then can be combined further until the developer has an understanding of the general purpose of a program. The third model is the integrated models combine top-down and bottom-up program comprehension.

The developer typically uses top-down comprehension ever possible. If a programmer has some knowledge about the domain, he/she will start with top-down comprehension. When he encounters code fragments he/she cannot explain using his domain knowledge, he/she will switch to the bottom-up comprehension (Feigenspan, 2011). A better code understanding by programmers and what is most efficient and effective can lead to many kinds of improvements such as better tools, better maintenance processes and guidelines, and documentation that support the cognitive process.

Static analysis is one of the most important areas that focus on understanding the source code; it has the ability to analyze large amounts of source code in considerably shorter amount of time than a human could. Static analysis aims to statically test the text of a program, without attempting to execute it; static analysis tools generate a first pass of the code base and highlight areas that require more attention from a senior developer.

Software metrics are one of the important aspects of software engineering. Which acts as an indicator for software attribute. It also plays an important role in the management of software projects. Software metric is defined in the IEEE 1061 standard as a function that has an input software data, and the output from these data is a single numerical value, that can be explained as the degree to which software possesses a given attribute that affects its quality. The goal is gaining objective, quantifiable measurements and reproducible, which may have valuable applications in budget planning, cost estimation, software debugging, quality assurance testing, and optimizing personnel task assignments, so analyzing software metric provide another way to understand the software from the produced numerical value.

1.1 Source Code Summarization:

Source code, is a description of a computer program which can be textual, readable, human readable, static, and fully executable that can be compiled automatically into an executable form (Binkley, 2007). Source code also can be defined as a mixed artifact that contains information that enables the communication between the developers and the compiler. So

the Automatic Source code summarization is the process of producing an illustrative subset of the data, with a computer program that contains an information of the entire source code. So in order to summarize the source code there is a need to understand the source code.

When any software product has been developed, not only the executable file or the source code is developed, but also a different kind of documents are developed as a part of software engineering process such as software requirement document, design document, test document, etc. Good documents are very useful and they serve many purposes. The documents that are produced in order to understand the source code may be included within the source code, so here the software or the source code have an Internal Documentation, or included outside the source code which is called external documentation, where programmers keep their notes and explanations in a separate document. For software developers, external documentation is useful as it consists of information that describes the problems with the program in order to solve them, or it can also focus on documenting general description of the software code without being concerned with its detail written. The main aim from external documentation is to provide easy views for software code.

The Internal documentation which is explained by comments, these block of comment for the Java and C/C++ programming language, can be categorized in the following seven different types (Steidl, 2013):

1- Copyright comments: this type of comments is usually found at the beginning of each file, it includes information about the license or the copyright of the source code file.
2- Header comments: In Java, headers they found after the imports but before the class declaration, it gives an overview about the functionality of the class and provides information about, e. g., the class author, the peer review status, or the revision number.
3- Member comments: they provide information for projects and for API the developer. It describes the functionality of a method, being located either before or in the same line as the member definition.
4- Inline comments: describe implementation decisions used within a method body.
5- Section comments address several methods/fields together belonging to the same functional aspect.
6- Code comments: this kind of comments is temporarily commented for potential later reuse or debugging purposes.
7- Task comments: are developer notes containing a remaining to do, a remark about an implementation hack, or a note about a bug that needs to be fixed.

1.2 Aims and the importance of this study

Software comprehension is an important field in the software engineering; it is the core for many other activities such as reuse, maintenance, development, and software changes. This requires software engineers to spend a lot of time and effort to analyze and understand the software. So summarizing software artifacts is the best solution that helps the developer, maintainer, or any other one who aims to understand the software.

Many of the previous researchers focus on summarizing source code artifacts. So the commit produced from summarizing the source code just provides summary information about part of the software, and doesn't cover the overall software. They either provide a summary that describes the context of the artifact or they describe the semantic behind the class or the method, by analyzing the stereotype.

From here the importance of this research comes, so it aims to give a number of external descriptive views that summarize all the granularity levels of the software (i.e.: method, class and the package) by providing a general description that describes a quantity information for each artifact in the software, and more detailed description that provide semantic information that the syntax of each artifact holds for the selected artifact, which are presented as a set of reports , also the method control flow graph that views the method with some metrics that aim to measure the method, and the class call graph which is also supported with the main class metrics that measures the class quality.

1.3 Thesis claims

This thesis aims to introduce the proposed approach as a substitute for many other approaches, since it has been used to provide a good comprehension and understanding to the software engineers in order to help them in many areas. So it will be easy to develop, maintain, reuse, and analyze the software by reducing effort and time.

1.4 Contribution of this research

Although there are different ways introduced to understand the software, automatic program comprehension is the most efficient and wanted way. Internal and external documentation help during program understanding and it is also still an important research area.

This research proposes a new approach which aims to summarize the software system by analyzing the source code statically, in order to determine its elements to understand the relations between those elements, by generating a descriptive summary for the target software project.

Since source code contains a lot of text so we parse source code to xml tags using srcML (source code Markup Language) tool in order to analyze the source code, because it adds much of the syntactic information that is found in the Abstract Syntax Tree (Maletic, 2002). It also combines text with both structural and textual information of the source code and provide an easy way to extract information from the source code (Collard, 2002). All this makes the software comprehension directly supported, the main contributions from the proposed work are summarized in the following points:

- The target software artifacts are packages, classes and methods. And the generated summaries are hydride of texts, graphs, and numerical measures.
- The proposed methodology introduces a new approach that aims to generate the both class call graph and method control flow graph that represents a view for both class and method.
- There are also some other important contributions which aim at answering the following research questions:
- Does the generated descriptive summary summarizes, describes, and identifies the source code artifacts (package, class and method) automatically?
- Does the generated descriptive summary reflect the developers understanding of the software?

1.5 Thesis Structure

The remainder of this thesis is organized as follows: Chapter 2 discusses the summarization overview and reviews most of the work in the field of source code artifact summarization techniques and the source code comprehension techniques. This chapter consists of reviewing the methodology of each work, and the results that were achieved and any open source case study that was used. The source code artifacts descriptive summary is discussed in Chapter 3 detail how the new method works, with a number of examples for the generated descriptive summary. Chapter 4 provides the results of the new methodology, conclusion with a summary of the research conducted and recommendations for future work.

Chapter 2
Review of Literature

Introduction

In order to understand the source code many approaches were introduced by following many methods and techniques. In general programmer makes a mental map of the code by looking at and recognizing various knowledge structures by including specific domain knowledge as well as recognized structures in the source code. For example, when programmer wants to understand a while loop in some code he will look for the end of the loop, the condition to exit the loop and how the condition is changed, since that is how while loops are structured in general. Then the mental map of the code used to predict what will happen next in the code.

As a starting point, simply the summary can be defined as producing a text from one or more texts or list of sentences produced from one or more documents that presents the main points in a concise form, which contains a significant portion of the information in the original text(s), where it is not longer than half of the original text(s). When this is done by the means of a computer or automatically, it will be called Automatic Text Summarization (Lloret, 2008).

2.1 Source Code Artifact

Correia in his work (de Figueiredo Correia, 2015), defines the software artifacts as both the products of software development and the things that developers work with. They may be themselves part of the final set of deliverables to be built; they may describe or support the process of developing software, and how it unfolds; and they are capable of describing the function and design of software, and therefore be used in the creation of other software artifacts.

Also (Juergens, 2011), defines software artifact as a file that is created and maintained during the life cycle of a software system. It is part of the system or captures knowledge about it. Examples include requirements specifications, models and source code. From the point of view of analysis, an artifact is regarded as a collection of atomic units. For natural language texts, these units can be words or sentences, for source code tokens or statements. For data-flow models such as Matlab/Simulink, atomic units are basic model blocks such as addition or multiplication blocks.

While (Fisher, 2009), defines the software artifact as something produced during the software development process. The ultimate goal of the process is to produce an operational program that satisfies user's needs. From an end user's perspective, this working program is the artifact of primary interest. Customers also need documentation artifacts that tell them how to use the software. This documentation can include users' manuals,

tutorials, and online program help. The major software engineering artifact is source code.

One of the active research topics in software maintenance is summarizing software artifacts. It can said that the artifacts of a software system includes sotware code and executable files, they also include a hirarical diagrams of the software such as UML class diagram. Table 1, summarizes the kinds of languages and notations that can be used for different software artifacts.

Table 1:
Languages or notations used for software artifacts

Software Artifact	Language or Notation
Requirements	English and pictures, in electronic or paper form.
Specification	A formal specification language, such as SpecL where SpecL manages the logic and date complexity, reporting ambiguities to the user, or by applying a modeling notation, such as UML.
Design	A structured software documentation format, such as Javadoc, or a modeling notation, such as UML
Implementation	A programming language, such as Java or C++, and the graphical program diagramming notation also can apply.

According to Table 1, each phase of software engineering is defined as an artifact, and for each phase it is possible to represent it as graphical notation, structured documents, pictures, or programming language such as C++ or java.

2.2 Software Comprehension

There are different methods to deal with the source code artifacts in order to understand it.

- Some of them deal with the source code as a text since it contain a natural language to introduce a document from it, other methods, to introduce a document deal with the source code as fragments were they invistigate the artifacts from them.
- Some methods aim to find some features.
- On the other hand some comprehension techniques aim to provide quantities that aim to measure the software quality.

Program comprehension is popular area, the idea in this area summarize in breaking a large program into more manageable slices or smaller parts . So, instead of trying to comprehend the program as a whole, the programmer can try to comprehend these slices. were a slice is a set of

statements related by data and control flow. This way can be performed programmatically and can be useful for debugging of computer programs and during program comprehension (O'brien, 2003). Table 2, provides a prife describtion to the methods proposed to comprehence the sotware:

Table 2
Software comprehension

Method	Description
Visualization techniques	Visualization techniques aims to visualize the application using graphs, uml diagram or views (Pierre Caserta, 2011).
Metrics trace techniques	Metrics are useful for analysis purposes, it aims to measure the software project in order to determine the complexity, software size and the qulity of the source code (Sneed, 2006).
Quering techniques	Quering techniques provide a mechanisim to extract the progrm artifacts and the relationship between them. This will help in visualization or take query results as an input for further queries and analyses (Gˆırba, 2008).
Text retrieval techniques	It is designed to work with the documents that are written in natural language, and since source code contain natural language, it can be easily applied (McBurney, 2014).
Heuristic based techniques	This kind of techniques employed for learning or solving problem solutions which are good enough for a given set of data or conditions. It generates a light abstractive summary to the extracted information from the source code (Nazara, 2015).
Dynamic analysis techniques	It means the analysis of data gathered from a running program, it exposes the system's actual behavior so provide an accurate picture of a software system. This technique comprises the analysis of a system's execution through interpretation (for example using the Virtual Machine in Java) (Hamou-Lhadj, 2009).
Static analysis techniques	Is the analysis of computer software without performing the actual execution of the programs built from that software, it is usually applied to the analysis performed with human analysis and by using automated software tool (Gomes, 2009).
Fact collection	According to this technique developers working in the source code in order to search, learn, review, implement and propose facts about the source code in order to serve numerous roles, such as predicting the amount of

العنوان:	Automatic Generation of Descriptive Summary for Source Code Artifact
المؤلف الرئيسي:	البطوش، أماني عبدالسلام
مؤلفين آخرين:	حماد، مصطفى محمد(مشرف)
التاريخ الميلادي:	2016
موقع:	مؤتة
الصفحات:	1 - 86
رقم MD:	951089
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة مؤتة
الكلية:	عمادة الدراسات العليا
الدولة:	الاردن
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، برمجية جافا، برمجية سي
رابط:	https://search.mandumah.com/Record/951089

with both Cyclomatic complexity, and Hallstead complexity measures, that aims to measure method quality, while the previous approaches introduce the method control flow graph in work 9, provides the control flow graph for Java method, with the Cyclomatic complexity measure for each one.

By answering the research questions, and discussing the previous approaches that aims to provide a descriptive summary for source code artifacts, we find that the proposed methodology shows that generating a descriptive summary for the following target software artifacts: packages, classes, and methods in more than one view, make it more easy, and understandable for who concern with the software, to understand the target software in the way that he find it more suitable.

4.3 Conclusion and Future Work

A new approach for generating a descriptive summary for the source code artifacts, in more than one view, is proposed. The generated descriptive views consist of text and graph based information. Textual based reports provide syntactic information project's packages and classes. The class call-graph view is generated for each class with a number of class's metrics, such as LOC, RFC, NOC, and WMC. The method control-flow graph is produced for each method with Cyclomatic complexity measures, and the Hallstead complexity measures. The main metrics in both CCG and MCG hold the semantics information within both class and method.

Summarizing the software and presenting the extracted information for each package, class, and method in more than one view that covers these granularities of the target source code, makes the source code more understandable and maintainable. Two case studies are applied for the proposed approach. The generated reports and views showed that they can support software comprehension and can be seen as a reverse engineering analysis for the source code.

We are planning to extend our work by enhancing the package report to cover the relationships among the package's classes. Moreover, measuring the effectiveness of the proposed approach, and implementing a full feature GUI, for the proposed approach are other goals for the future work.

العنوان:	Automatic Generation of Descriptive Summary for Source Code Artifact
المؤلف الرئيسي:	البطوش، أماني عبدالسلام
مؤلفين آخرين:	حماد، مصطفى محمد(مشرف)
التاريخ الميلادي:	2016
موقع:	مؤتة
الصفحات:	1 - 86
رقم MD:	951089
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة مؤتة
الكلية:	عمادة الدراسات العليا
الدولة:	الاردن
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، برمجية جافا، برمجية سي
رابط:	https://search.mandumah.com/Record/951089

References

Abid, N. J. (2015, September). Using stereotypes in the automatic generation of natural language summaries for C++ methods. *IEEE International Conference on. IEEE*, pp. 561-565.

Aguiar, A. G. (2004). JavaML 2.0: Enriching the markup language for Java source code. *XML: Aplicaçoes e Tecnologias Associadas*, pp. 1-12.

Alimucaj, A. (2009). http://eclipsefcg.sourceforge.net/. Retrieved March 27, 2016, from sourceforge.

Allen, F. E. (1970, July). Control flow analysis. *ACM Sigplan Notices*, pp. 1–19.

Binkley, D. (2007). *Source code analysis: A road map.* Future of Software Engineering. IEEE Computer Society.

Buse, R. P. (2010). *Automatically documenting program changes.* Proceedings of the IEEE/ACM international conference on Automated software engineering. ACM.

Chawla, S., & Kaur, G. (2013). Comparative Study of the Software Metrics for the complexity and Maintainability of Software Development. *International Journal of Advanced Computer Science & Applications*, *4*.

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on, 20(6),* pp.476-493.

CollabNet.(2001).http://argouml.tigris.org/source/browse/argouml/trunk/modules/dev/src/org/argouml/dev/figinspector/. Retrieved March 27, 2016, from argouml.

Collard, M. D. (2011, September). Lightweight Transformation and Fact Extraction with the srcML Toolkit. *IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM'11)*, pp. 173-184.

Collard, M. L. (2002). Supporting document and data views of source code. *ACM symposium on Document engineering*.

Cortés-Coy, L. F. (2014). *On Automatically Generating Commit Messages via Summarization of Source Code Changes.* IEEE 14th International Working Conference on. IEEE.

Daumé Iii, H. J. (2009). *Search-based structured prediction.* Machine learning 75.3.

de Figueiredo Correia, F. A. (2015). Documenting Software With Adaptive Software Artifacts. *PhD Thesis. UNIVERSIDADE DO PORTO*.

Eddy, B. P. (2013). *Evaluating source code summarization techniques: Replication and expansion." Program Comprehension (ICPC).* IEEE 21st International Conference on. IEEE.

Ellina, P. (2007). http://blog.prashanthellina.com/generating-call-graphs-for-understanding-and-refactoring-python-code.html. Retrieved March 27, 2016, from prashanthellina.

Feigenspan, J. (2011). Program Comprehension of Feature-Oriented Software Development. *International Doctoral Symposium on Empirical Software Engineering*, p. Vol. 21.

Fisher, G. (2009). *Software Engineering Formal and Practical.* California Polytechnic State University.

Frakes, W. a. (2005, July 31). Software Reuse Research: Status and Future. *IEEE Transactions on Software Engineering*, pp. 529-536.

Fujita, H. a. (2007). Experience of XML-based source code representation with parsing actions. *New Trends in Software Methodologies, Tools and Techniques*, pp. 330-339.

Gˆırba, M. V. (2008). Query Technologies and Applications for Program Comprehension. *The 16th IEEE International Conference on Program Comprehension*.

Gerald Kaszuba, e. a. (2007). http://pycallgraph.slowchop.com/en/master/. Retrieved March 27, 2016, from slowchop.

Godfrey, M. W. (2008, September). The past, present, and future of software evolution. *Frontiers of Software Maintenance, IEEE*, pp. 129-138.

Gomes, I. e. (2009). An overview on the static code analysis approach in software development. *Faculdade de Engenharia da Universidade do Porto, Portugal*.

Haiduc, S. e. (2010). *On the use of automated text summarization techniques for summarizing source code.* 17th Working Conference on. IEEE.

Haiduc, S. J. (2010). *Supporting program comprehension with source code summarization.* ACM/IEEE International Conference on Software Engineering.

Haiduc, S. J. (2013). *Supporting program comprehension with source code summarization.* Program Comprehension (ICPC).

Halstead, M. H. (1977). Elements of Software Science. Amsterdam: Elsevier North-Holland.

Hammad, M. A. (2016, May). *Summarizing Services of Java Packages.* Lecture Notes on Software Engineering.

Hamish Graham, H. Y. (2004, January.). A Solar System Metaphor for 3D Visualisation of Object Oriented Software Metrics. In Proceedings of the 2004 Australasian symposium on Information Visualisation, pp. 53-59.

Hamou-Lhadj, A. (2009). Techniques to Simplify the Analysis of Execution Traces for Program Comprehension. 11,12.

Juergens, E. (2011). *Why and how to control cloning in software artifacts.* Technische Universität München.

Kanellopoulos, Y. a. (2004). Data mining source code to facilitate program comprehension: experiments on clustering data retrieved from C++ programs. *12th IEEE International Workshop on. IEEE*, pp. 214-223.

LaToza, T. D. (2007, September). Program comprehension as fact finding. *the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pp. 361-370.

Lanza, R. W. (2011). Visualizing Software Systems as Cities. *in Proceedings of the 33rd International Confernce on Software Engieering,ser. ICSE*, pp. 551-560.

License, G. (2014, October 29). http://www.umlgraph.org/download.html. Retrieved January 12, 2016, from umlgraph.

Lloret, E. (2008). *Text summarization: an overview.* Paper supported by the Spanish Government under the project TEXT-MESS (TIN2006-15265-C06-01).

Maletic, J. I. (2002). Source code files as structured documents . *10th International Workshop on. IEEE, p*p. 289-292.

Mamas, E. a. (2000). Towards portable source code representations using XML. *Seventh Working Conference on. IEEE*, pp. 172-182.

Mani, I. a. (1999). *Advances in automatic text summarization.* Cambridge: MA: MIT press.

McBurney, P. W. (2014). *Automatic documentation generation via source code summarization of method context.* the 22nd International Conference on Program Comprehension. ACM.

Moreno, L. (2014, May). Summarization of complex software artifacts. *36th International Conference on Software Engineering. ACM*, pp. 654-657.

Moreno, L. e. (2013). *Automatic generation of natural language summaries for java classes.* IEEE 21st International Conference on. IEEE.

Moreno, L. e. (2013). *Jsummarizer: An automatic generator of natural language summaries for java classes.* IEEE 21st International Conference on. IEEE.

Myers, T. D. (2011). Visualizing call graphs. IEEE Symposium on Visual Languages and Human-Centric Computing.

Nazara, N. Y. (2015). *Summarizing Software Artifacts: Classifications, Methods, and Applications.* oscar-lab.

O'brien, M. P. (2003). Software comprehension–a review & research direction. *Department of Computer Science & Information Systems University of Limerick, Ireland, Technical Repor*.

Pierre Caserta, O. Z. (2011, July). Visualization of the Static aspects of Software: a survey. *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*.

Poulin, J. S. (1993). The business case for software reuse. *IBM Systems Journal*, pp. 567-594.

Rajlich, V. (2014, May). Software evolution and maintenance. *Future of Software Engineering. ACM*, pp. 133-144.

Santhana Megala, S. A. (2014). *Enriching Text Summarization using Fuzzy Logic*. International Journal of Computer Science & Information Technologies.

Selby, R. W. (2005). Enabling reuse-based software development of large-scale systems. Software Engineering, IEEE Transactions on, 31(6), PP. 495-510.

Slashdot. (2016). https://sourceforge.net/projects/junit/files/junit/3.8.1/. Retrieved January 10, 2016, from sourceforge.

Sneed, H. M. (2006, oct 30). Understanding software through numbers: A metric based approach to program comprehension. *Journal of Software Maintenance and Evolution: Research and Practice*, pp. 405 – 419.

Sonal Chawla, G. K. (2013). Comparative Study of the Software Metrics for the complexity and Maintainability of Software Development. *(IJACSA) International Journal of Advanced Computer Science and Applications*, pp. Vol. 4, No. 9.

Sridhara, G. (2012). *Automatic Generation Of Descriptive Summary Comments For Method In Object Orinted programs.*

Sridhara, G. e. (2010). *Towards automatically generating summary comments for java methods.* IEEE/ACM international conference Auomated software engineering. ACM.

Sridhara, G. L.-S. (2011). *"Generating parameter comments and integrating with method summaries.* Program Comprehension (ICPC), 2011 IEEE 19th International Conference on. IEEE.

Stanford NLP Group. (2009). *Stanford log-linear part of speech tagger.* Stanford NLP Group.

Steidl, D. B. (2013). *Quality analysis of source code comments.* Program Comprehension (ICPC), 2013 IEEE 21st International Conference on. IEEE.

Storey, M.-A. (2005). Theories, methods and tools in program comprehension: Past, present and future. *IWPC 2005. Proceedings. 13th International Workshop on. IEEE*, pp. 181-191.

Tjortjis, C. L. (2003, May). Facilitating program comprehension by mining association rules from source code. *11th IEEE International Workshop* , pp. 125-132.

Wilde, N. B. (2003). A comparison of methods for locating features in legacy software. *Journal of Systems and Software*, pp. 105-114.
Wong, E. T. (2015). *CloCom: Mining existing source code for automatic comment generation.* Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference.

العنوان:	Automatic Generation of Descriptive Summary for Source Code Artifact
المؤلف الرئيسي:	البطوش، أماني عبدالسلام
مؤلفين آخرين:	حماد، مصطفى محمد(مشرف)
التاريخ الميلادي:	2016
موقع:	مؤتة
الصفحات:	1 - 86
رقم MD:	951089
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة مؤتة
الكلية:	عمادة الدراسات العليا
الدولة:	الاردن
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، برمجية جافا، برمجية سي
رابط:	https://search.mandumah.com/Record/951089

Appendix 1

The Generated Summary for Case Study 1

```
1   Class AssertionFailedError report
2
3   Class AssertionFailedError declared in  package framework as public, has a super class Error.
4   This class provide the following services:
5
6   The service is:  AssertionFailedError.
7
8   The service is:  AssertionFailedError. The service uses local data:(message with type String).
9   This service use method: super.
10
```

Figure 20
The Generated Class Report for AssertionFailedError Class

```
1   Class ComparisonFailure report
2
3
4   Class ComparisonFailure declared in  package framework as public, has a super class AssertionFailedError.
5   This class provide the following services:
6
7
8   The service is:  getMessage. This service returns String. The service uses the attributes:(fExpected with type String,
9   fActual with type String ).
10  This service use methods: format, min, charAt, substring, and length.
```

Figure 21
The Generated Class Report for ComparisonFailure Class

```
1   Class RepeatedTest report:
2
3   Class RepeatedTest declared in package extension as public. Has a super class Canvas.
4   This class provide the following services:
5
6   The service is:  countTestCases.  This service returns int. The service use attribute: (ftimesRepeat  with
7   type int).
8   This service use local method: countTestCases.
9
10  The service is: run.  This service returns void. The service uses local data :(result with type TestResult).
11  The service use attribute: (ftimesRepeat  with type int ).
12  This service use local method: run.
13  This service use method: shouldStop.
14
15
16  The service is: toString.  This service returns String.
17  This service use local method: toString.
```

Figure 22
The Generated Class Report for RepetedTest Class

```
1   Class TestResult report
2
3   Class TestResult declared in  package framework as public, has a super class Object.
4   This class provide the following services:
5
6   The service is:  addError. This service returns void. The service use the attributes:(fFailures with type Vector). The
7   service uses local data:(test with type Test, and t with type Throwable ).
8   This service use local method: cloneListeners.
9   This service use methods: hasMoreElements, nextElement, addError, and elements.
10
11  The service is:  addFailure. This service returns void. The service use the attributes:(fFailures with type Vector). The
12  service uses local data:(test with type Test, and t with type AssertionFailedError).
13  This service use local method: cloneListeners.
14  This service use methods: TestFailure , hasMoreElements, elements, and addFailure.
15
16  The service is:  addListener. This service returns void. The service use the attributes:(fListeners with type Vector).
17  The service uses local data:(listener with type TestListener).
18  This service use method: addElement.
19
20  The service is:  addListener. This service returns void. The service use the attributes:(fListeners with type Vector).
21  The service uses local data:(listener with type TestListener).
22  This service use method: removeElement.
23
24  The service is:  cloneListeners. This service returns vector. The service use the attributes:(fListeners with type
25  Vector).
26  This service use method: clone.
27
28  The service is:  endTest. This service returns void. The service use the attributes:(fListeners with type Vector). The
29  service uses local data:(listener with type TestListener).
30  This service use local method: cloneListeners. And endTest.
31  This service use method: elements, hasMoreElements, and nextElement.
32
33  The service is:  errorCount. This service returns int. The service use the attributes:(fErrors with type Vector). The
34  service uses local data:(listener with type TestListener).
35  This service use method: size.
36
37  The service is:  errors. This service returns Enumeration. The service use  attributes: (fErrorswith type
38  Vector). The service uses local data:(test with type Test, and t with type Throwable ).
39  This service use methods: elements.
40
41  The service is:  failureCount. This service returns int. The service use local attributes:(fFailures with type
42  Vector).
43  This service use method: size.
44
45  The service is:  failures. This service returns Enumeration. The service use local attributes:(fFailures with
46  type Vector).
47  This service use method: elements.
48
49  The service is:  run. This service returns void. The service use local data:(test with type TestCase).
50  This service use local method: runProtected.
51  This service use method: Protectable, Throwable, runBare, and endTest .
52
53  The service is:  runCount. This service returns void. The service use the attributes:(test with type).
54  This service use local method: cloneListeners, and startTest.
55  This service use method: countTestCases, elements, hasMoreElements, and nextElement.
56
57  The service is: stop. This service returns void. The service use the attribute :(fStop with type Vector).
58
59  The service is:  wasSuccessful. This service returns boolean. The service use local data :(test with type
60  Vector).
61  This service use local method: failureCount, errorCount.
```

Figure 23
The Generated Class Report for TestResult Class

```
1   Class AboutDialog report:
2
3   Class AboutDialog declared in package swingui as public. Has a super class JDialog.
4   This class provide the following services:
5
6   The service is:  createLogo.  This service returns JLabel.
7   This service use method: getIconResource, and JLabel.
8
```

Figure 24
The Generated Class Report for AboutDialog Class

```
1   Class ProgressBar report:
2
3   Class ProgressBar declared in package awtui as public Has a super class Canvas.
4   This class provide the following services:
5
6   The service is:  getStatusColor. This service returns Color. The service use attribute: (fError with type
7   boolean).
8
9   The service is: paint.  This service returns void. The service uses local data :(g with type Graphics).
10  This service use local method: paintBackground, paintStatus.
11
12  The service is: paintStatus.  This service returns void. The service uses local data :(g with type Graphics).
13  This service use methods: setColor, getBounds, fillRect, and drawLine.
14
15  The service is: paintStep.  This service returns void. The service uses local data :(startX with type int,
16  and endX with type int).
17  This service use method: repaint.
18
19  The service is: reset.  This service returns void. The service uses attributes :(fProgressX with type int,
20  fProgress with type int, and fError with type boolean).
21  This service use local method: paint.
22
23  The service is: scale.  This service returns int. The service uses local data :(value with type int). The
24  service uses attributes :(fTotal with type int).
25  This service use method: max, and getBounds.
26
27  The service is: setBounds.  This service returns void. The service uses local data :(x with type int, y with
28  type int, w with type int, and h with type int). The service uses attributes :(fProgressX  with type int,
29  and fProgress with type int ).
30  This service use local method: setBounds, and scale.
31  This service use method: super .
32
33  The service is: start.  This service returns void. The service uses local data :(total with type int). The
34  service use attribute: (fError with type boolean).
35  This service use local method: reset.
36
37  The service is: step.  This service returns void. The service uses local data :(successful with type
38  boolean). The service use attribute: (fProgress with type int, fProgressX with type int , and fError with
39  type boolean).
40  This service use local method: paintStep.
```

Figure 25
The Generated Class Report for ProgressBar Class

```
1   Class LoadingTestCollector report:
2
3   Class LoadingTestCollector declared in package runner as public. has a super class ClassPathTestCollector.
4   This class provide the following services:
5
6   The service is:  isTestClass.  This service returns boolean. The service use local data : (classFileName  with type String).
7   This service use local method: classFromFile, and isTestClass.
8    This service use method: endsWith.
9
10  The service is: isTestClass.  This service returns boolean. The service uses local data :(testClass with type Class).
11  This service use local method: hasSuiteMethod, and hasPublicConstructor.
12  This service use method: isAssignableFrom, and isPublic.
13
14  The service is: hasSuiteMethod.  This service returns boolean. The service uses local data :(testClass with type Class).
15  This service use method: getMethod.
16
17  The service is: hasPublicConstructor.  This service returns boolean. The service uses local data :(testClass with type
18  Class).
    This service use method: TestSuite.
```

Figure 26
The Generated Class Report for LoadingTestController Class

```
1   Class TestFailure report:
2
3   Class TestFailure declared in  package framework as public, has a super class Object.
4   This class provide the following services:
5
6   The service is:  failedTest. This service returns Test.
7
8   The service is:  thrownException. This service returns Throwable.
9
10  The service is: toString. This service returns String.
11  This service use local method: toString.
12  This service use methods: StringBuffer, and append.
13
14  The service is: trace. This service returns String.
15  This service use local method: toString.
16  This service use methods: StringWriter, PrintWriter, printStackTrace, and getBuffer.
17
18  The service is: exceptionMessage. This service returns String.
19  This service use local method: thrownException.
20  This service use methods: getMessage.

    The service is: isFailure. This service returns boolean.
    This service use local method: thrownException.
```

Figure 27
The Generated Class Report for TestFailure Class

```
1   Class RepeatedTest report:
2
3   Class RepeatedTest declared in package extension as public. Has a super class Canvas.
4   This class provide the following services:
5
6   The service is:  countTestCases.  This service returns int. The service use attribute: (ftimesRepeat  with
7   type int).
8   This service use local method: countTestCases.
9
10  The service is: run.  This service returns void. The service uses local data :(result with type TestResult).
11  The service use attribute: (ftimesRepeat  with type int ).
12  This service use local method: run.
13  This service use method: shouldStop.
14
15  The service is: toString.  This service returns String.
    This service use local method: toString.
```

Figure 28
The Generated Class Report for Repeated Test Class

```
1   Class TestSuite report:
2
3   Class TestSuite declared in  package framework as public.
4   This class provide the following services:
5
6   The service is:  addTest. This service returns void. The service use local data :(test with type Test)
7   This service use method: addElement.
8
9   The service is:  addTestSuite. This service returns void. The service use local data :(testClass with type Class)
10  This service use local method: addTest.
11
12  The service is: addTestMethod. This service returns void. The service use local data :(m with type Method, names with
13  type Vector, and theClass with type Class)
14  This service use local method: addTest, getName, and isPublicTestMethod.
15  This service use methods: contains, isTestMethod, and addElement.
16
17  The service is: createTest. This service returns public. The service use local data :(theClass with type Class, name with
18  type String)
19  This service use local method: warning.
20  This service use methods: newInstance, and getParameterTypes.
21
22  The service is: exceptionToString. This service returns String. The service use local data :(t with type Throwable)
23  This service use local method: toString.
24  This service use methods: printStackTrace.
25
26  The service is: countTestCases. This service returns  int.
27  This service use local method: countTestCases.
28  This service use methods: hasMoreElements, and nextElement.
29
30  The service is: isPublicTestMethod. This service returns  boolean. The service use local data :(m with type Method)
31  This service use local method: isTestMethod.
32  This service use methods: isPublic.
33
34  The service is: isTestMethod. This service returns  boolean. The service use local data :(m with type Method)
35  This service use local method: getName.
36  This service use methods: getReturnType, and startsWith.
37
38  The service is:  testCount. This service returns int.
39  This service use method: size.
40
41  The service is: toString. This service returns String. The service use local data :(m with type Method, names with type
42  Vector, and theClass with type Class)
    This service use local method: getName, and toString.

    The service is: warning. This service returns Test. The service use local data :(message with type String)
    This service use method: fail.
```

Figure 29
The Generated Class Report for TestSuite Class

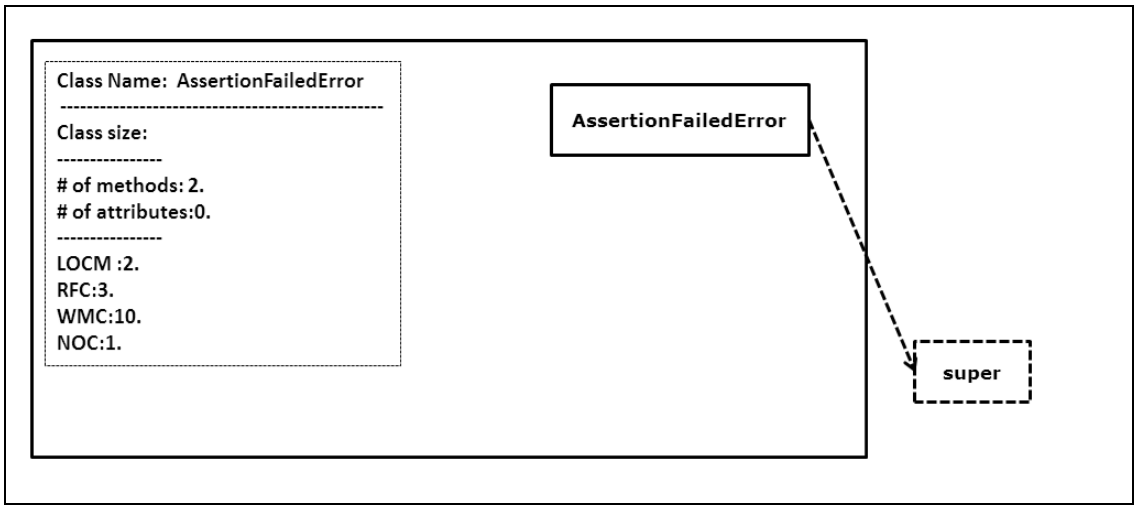

Figure 30
The Generated Class Call Graph For AssertionFailedError Class

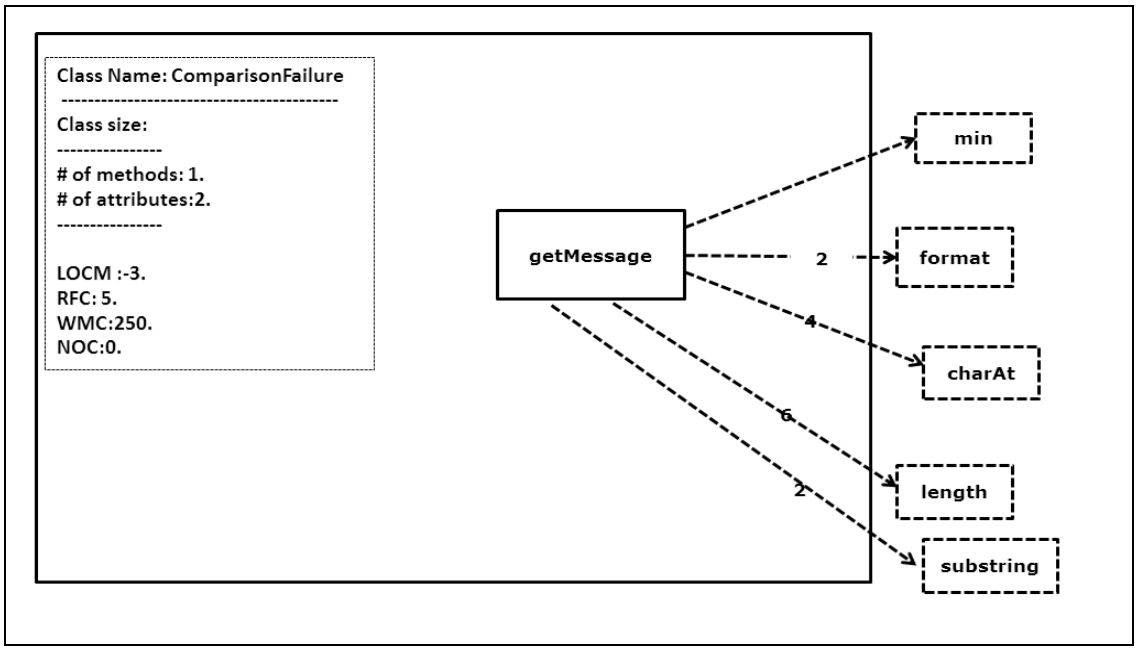

Figure 31
The Generated Class Call Graph For ComparisonFailure Class

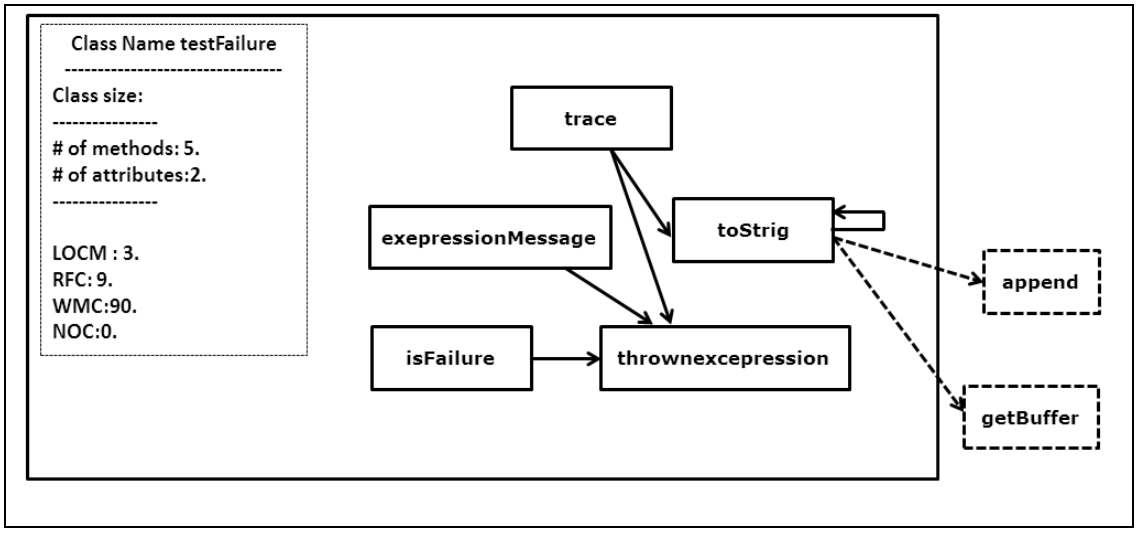

Figure 32
The Generated Class Call Graph For TestFailure Class

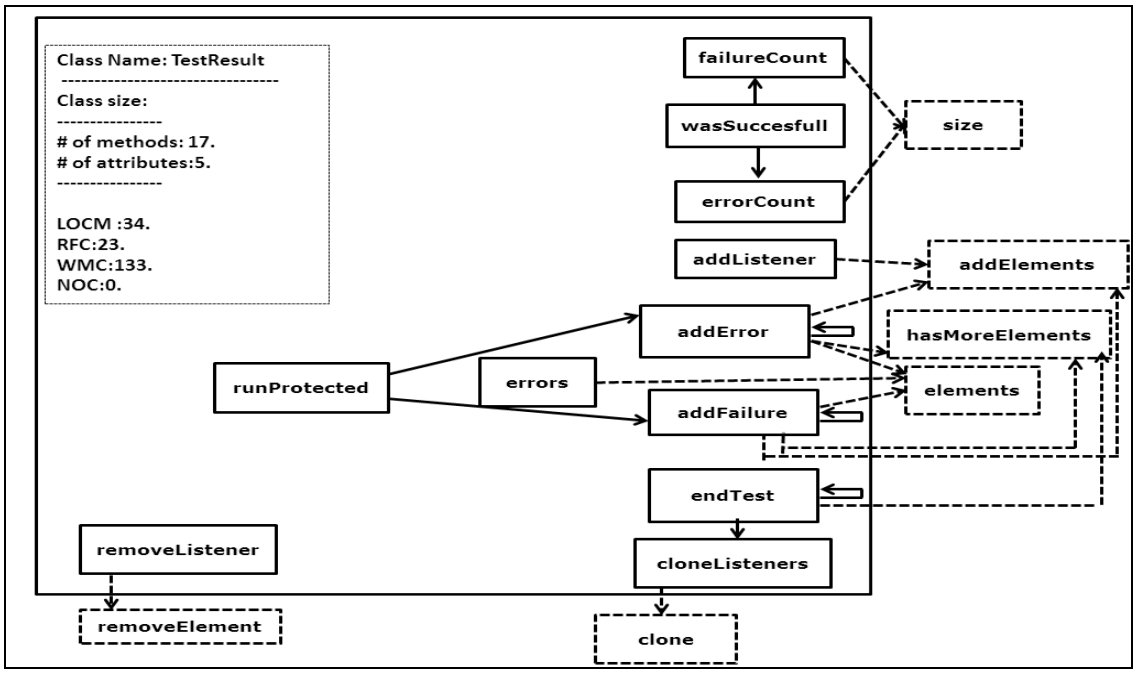

Figure 33
The Generated Class Call Graph For TestResult Class

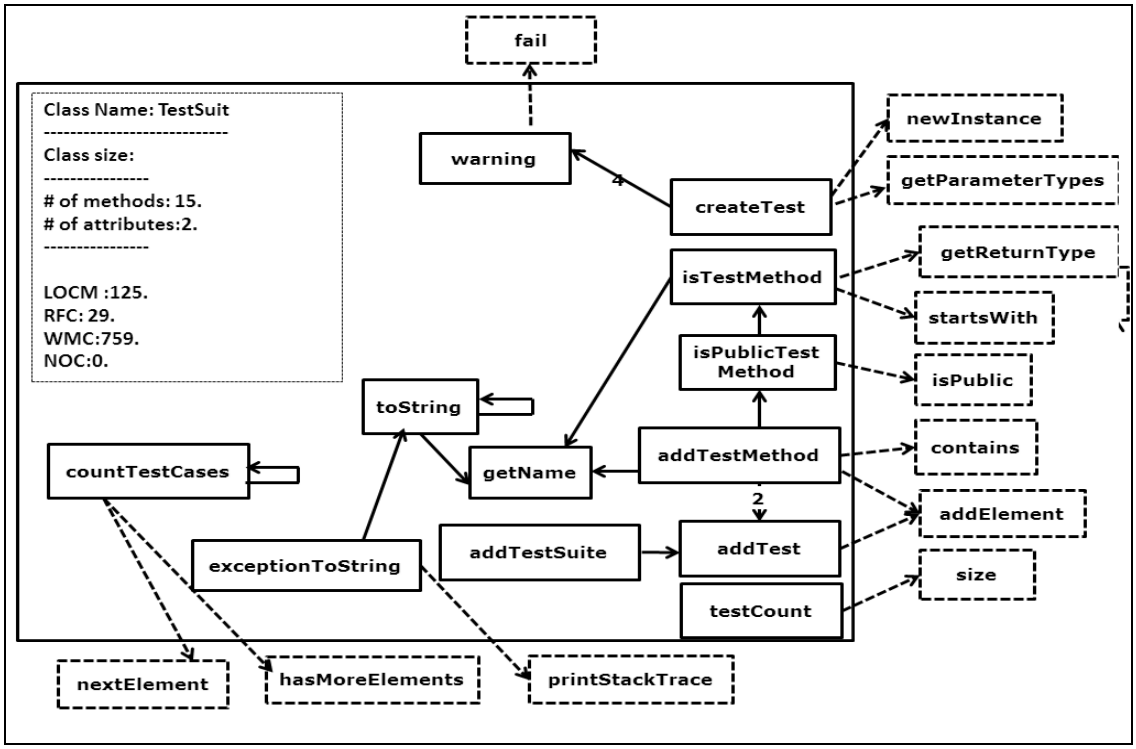

Figure 34
The Generated Class Call Graph For TestSuit Class

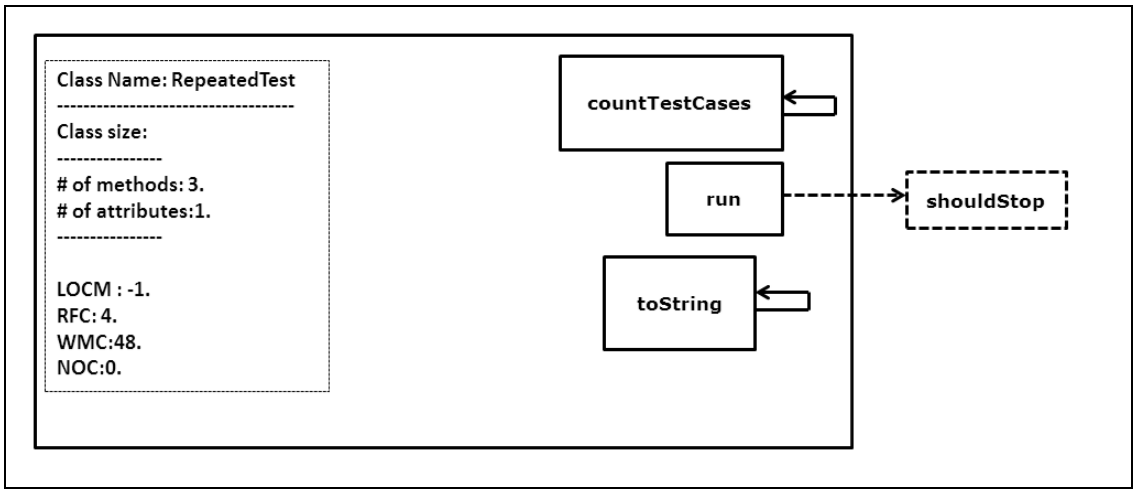

Figure 35
The Generated Class Call Graph For RepeatedTest Class

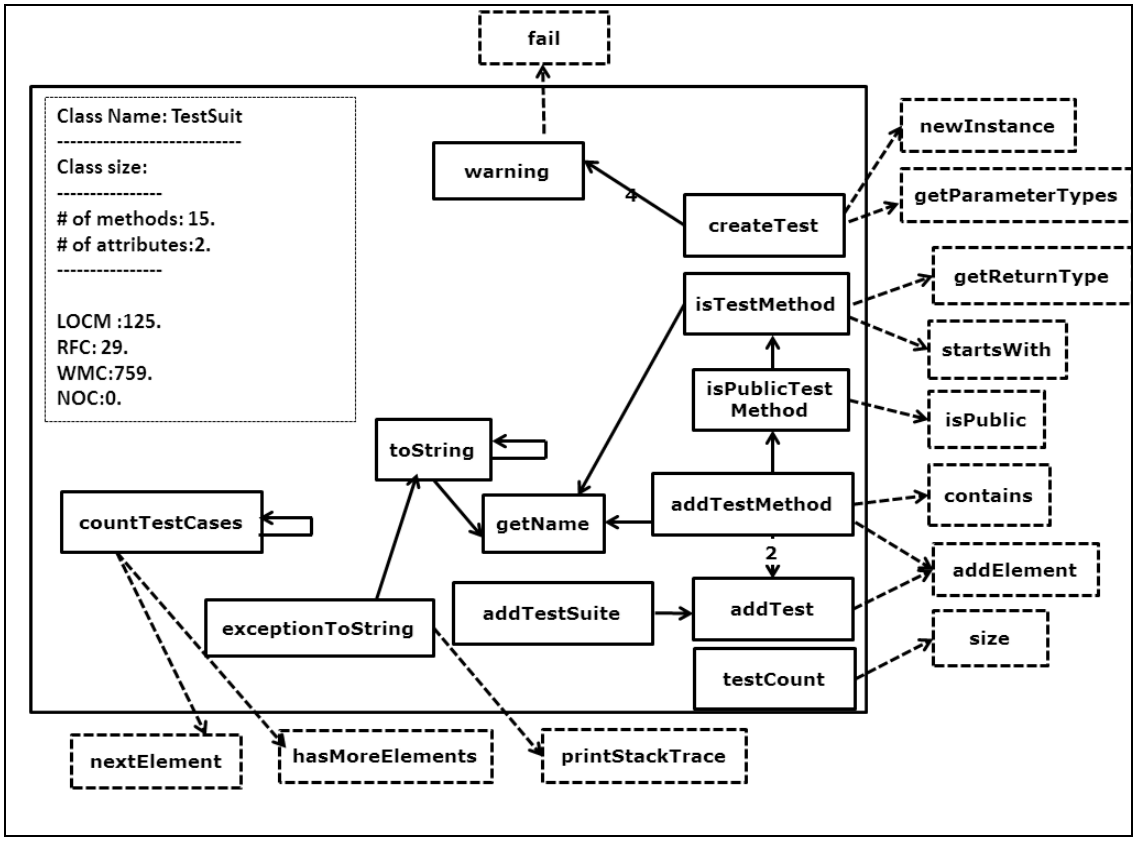

Figure 36
The Generated Class Call Graph For TestSuit Class

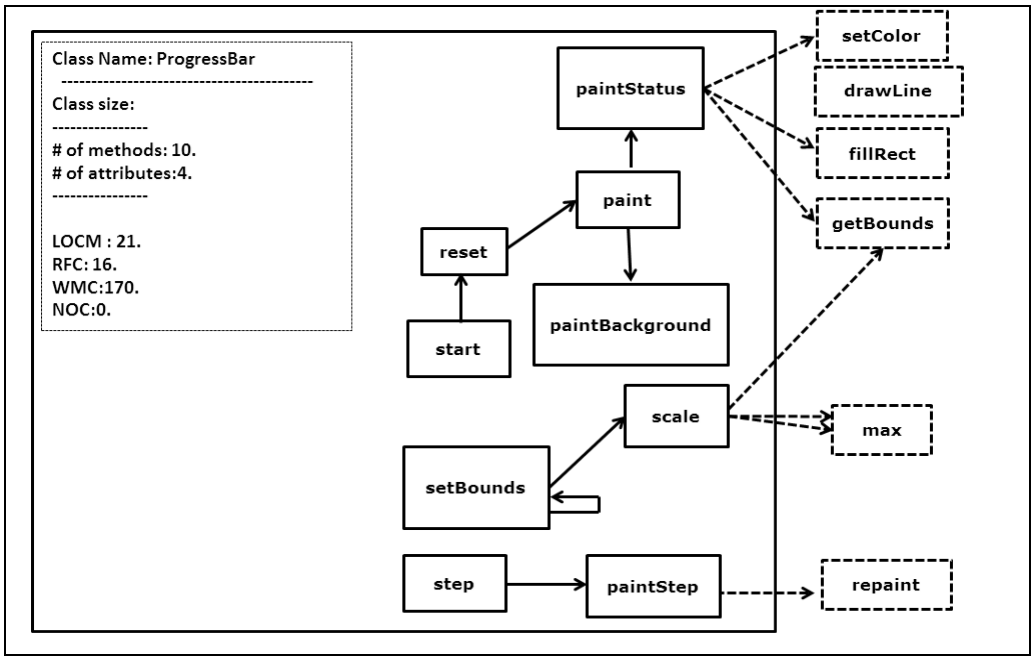

Figure 37
The Generated Class Call Graph For ProgressBar Class

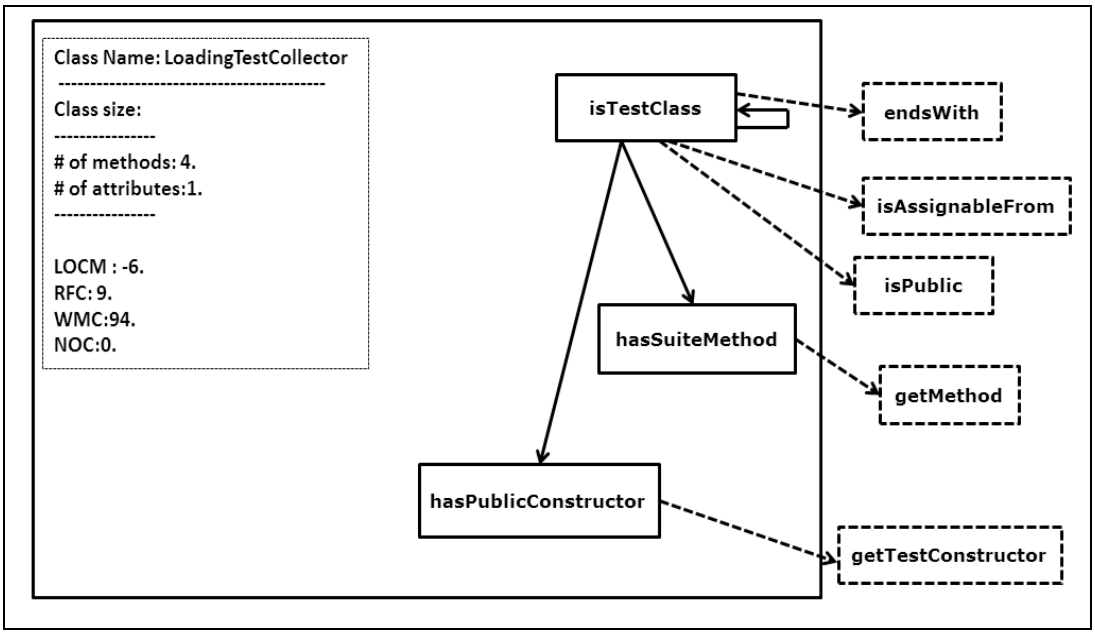

Figure 38
The Generated Class Call Graph For LoadingTestCollector Class

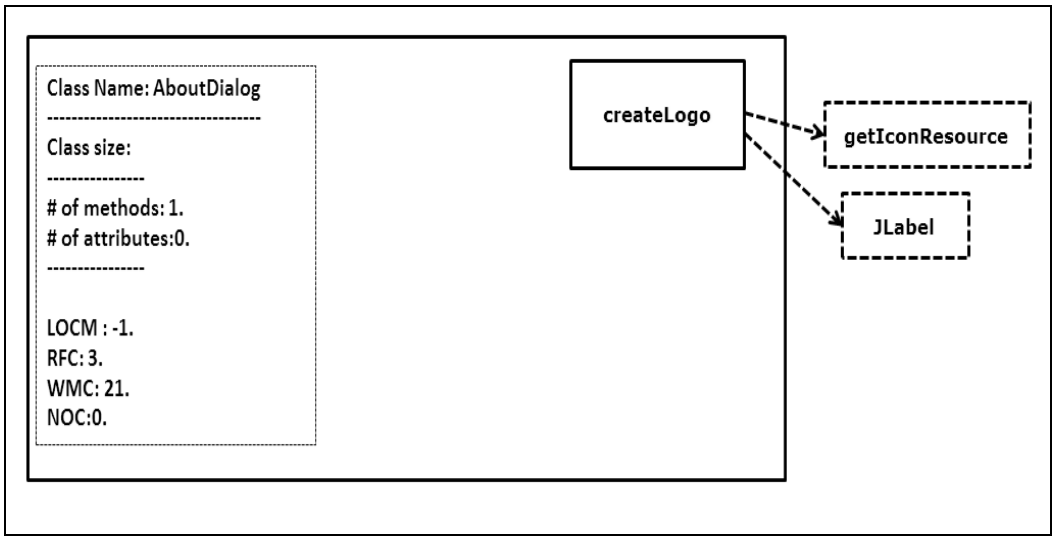

Figure 39
The Generated Class Call Graph For AboutDialog Class

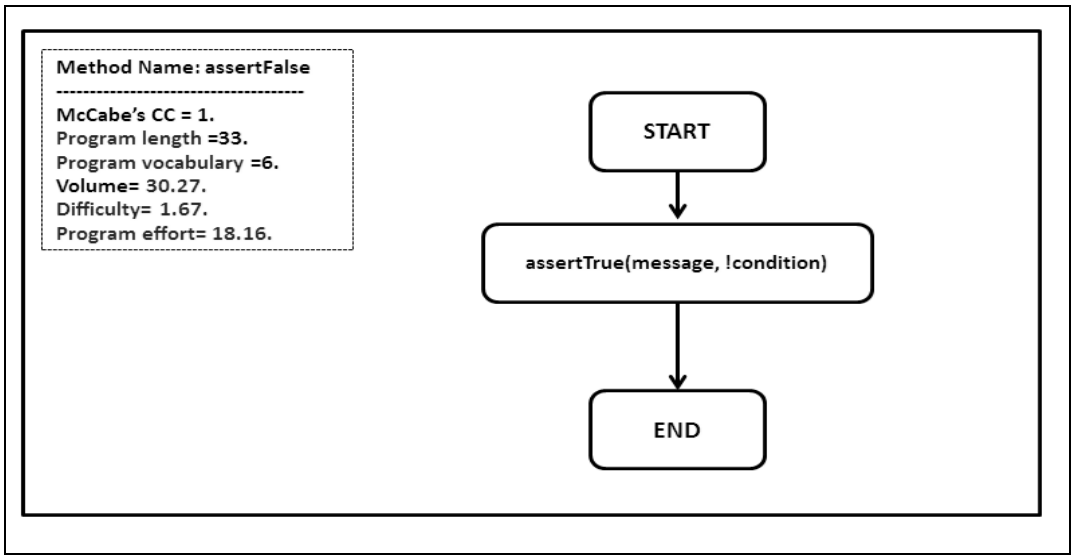

Figure 40
The Generated Method Control Flow Graph For assertFalse Method

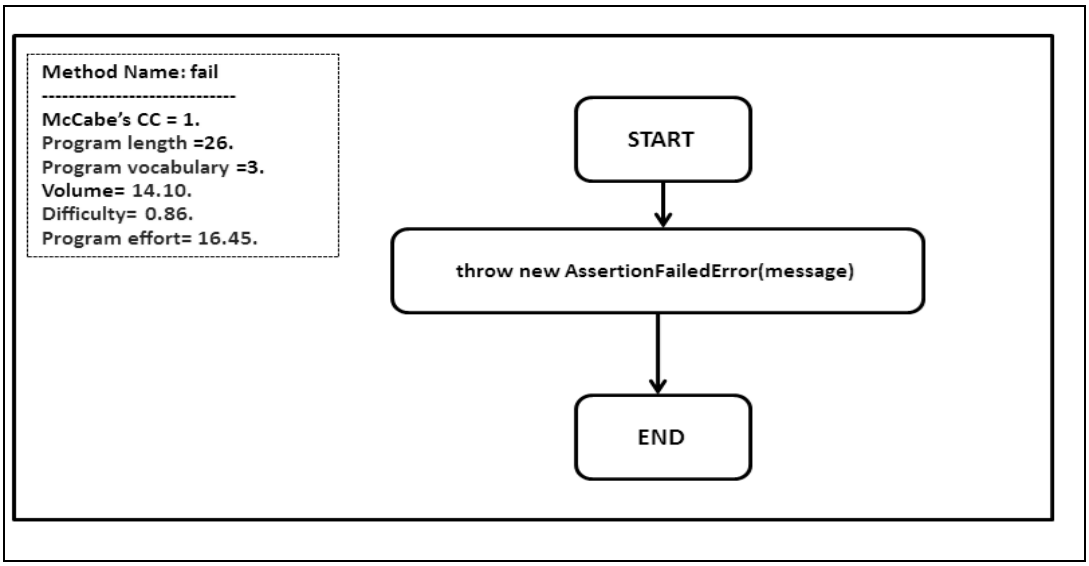

Figure 41
The Generated Method Control Flow Graph For fail Method

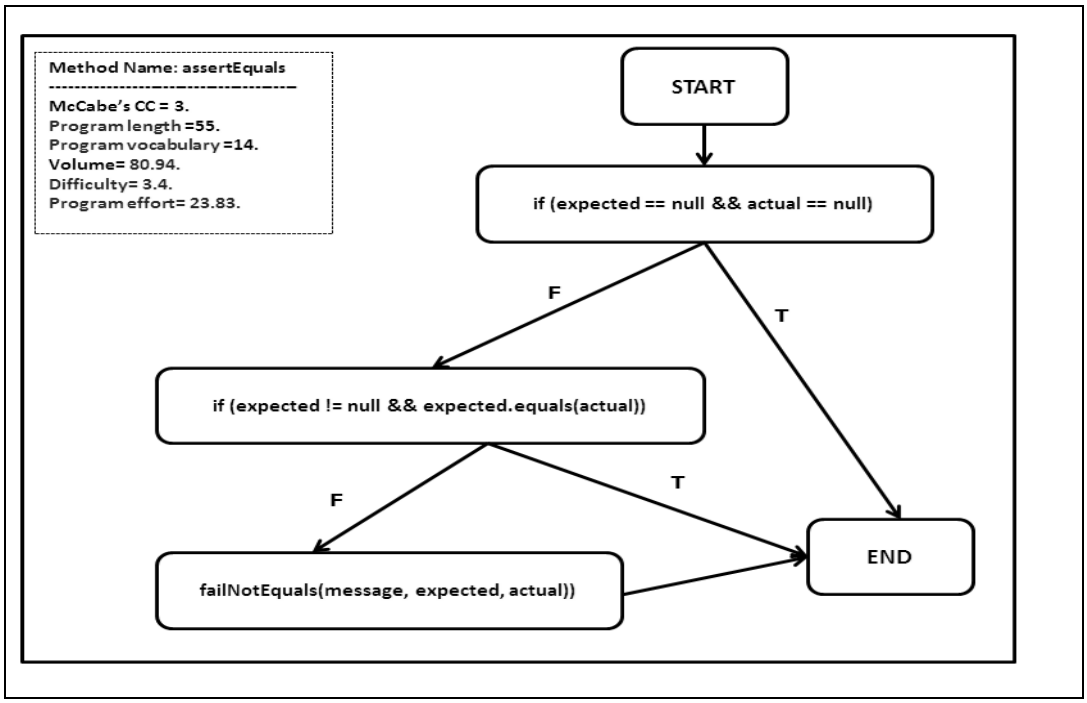

Figure 42
The Generated Method Control Flow Graph For assertEquals Method

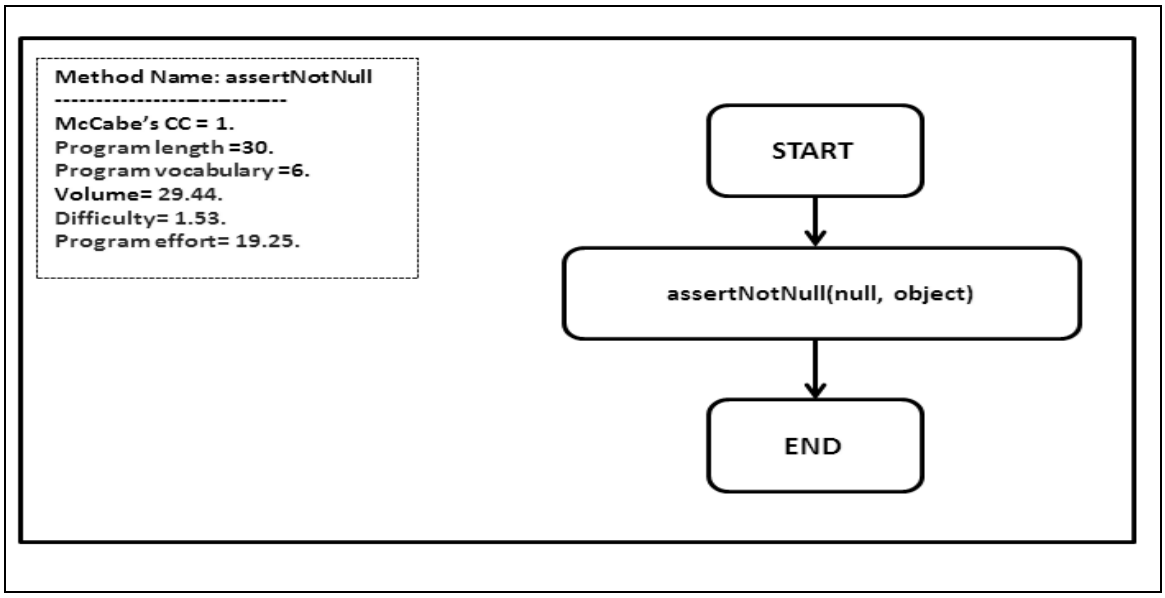

Figure 43
The Generated Method Control Flow Graph For assertNotNull Method

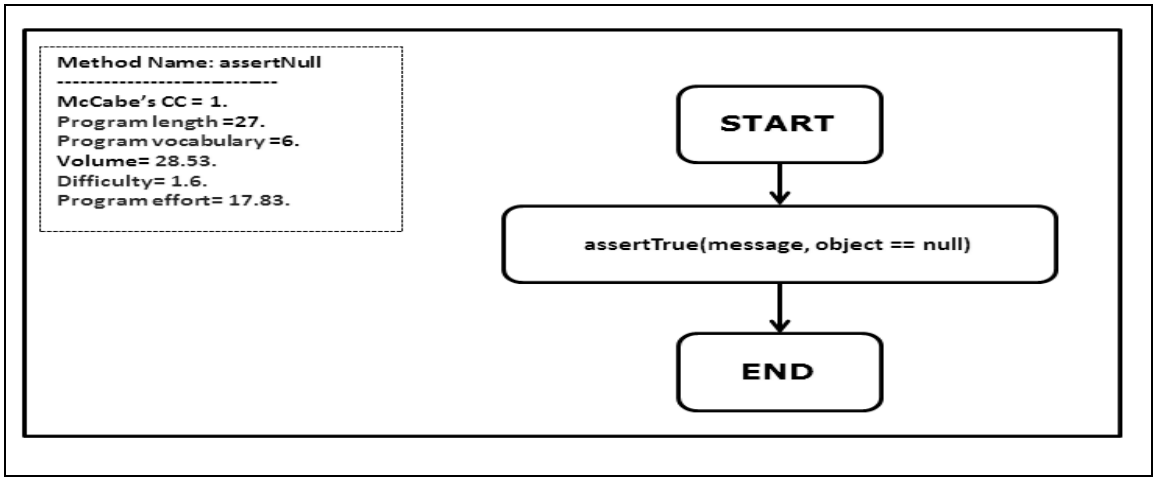

Figure 44
The Generated Method Control Flow Graph For assertNull Method

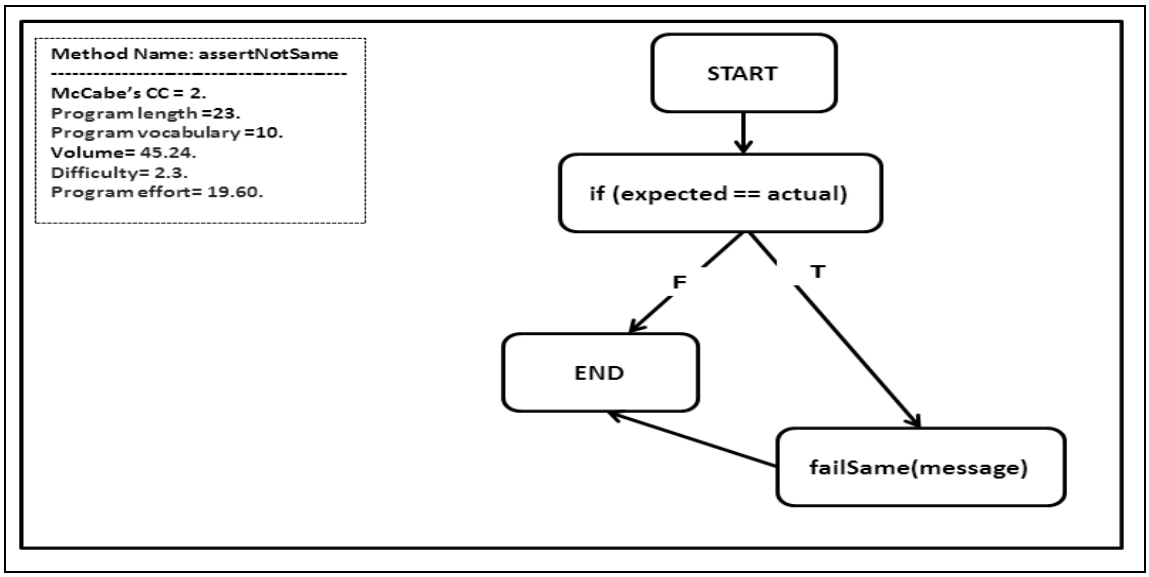

Figure 45
The Generated Method Control Flow Graph For assertSame Method

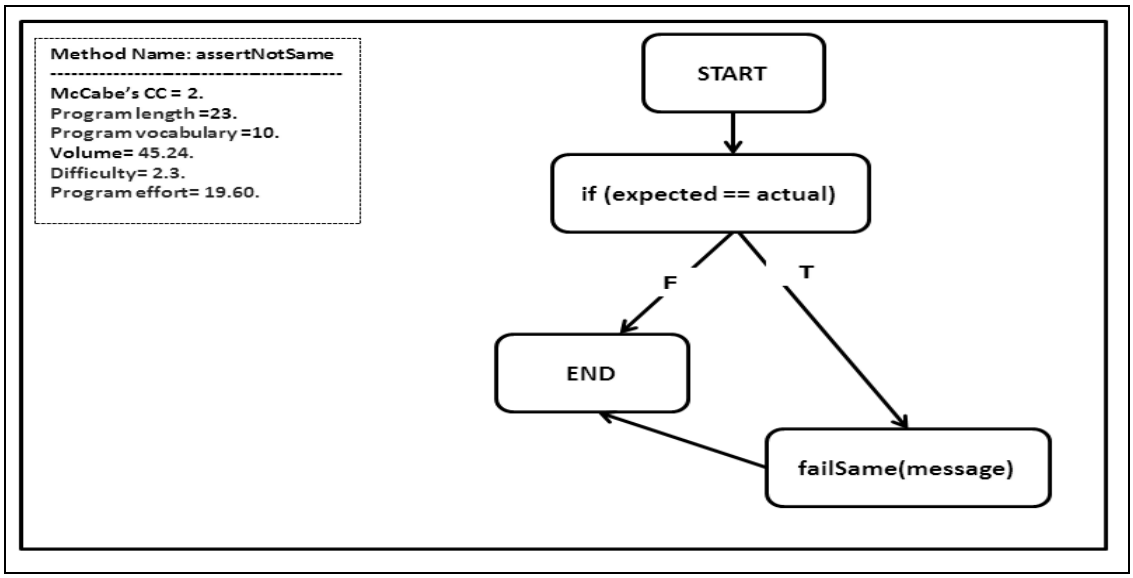

Figure 46
The Generated Method Control Flow Graph For assertNotSame Method

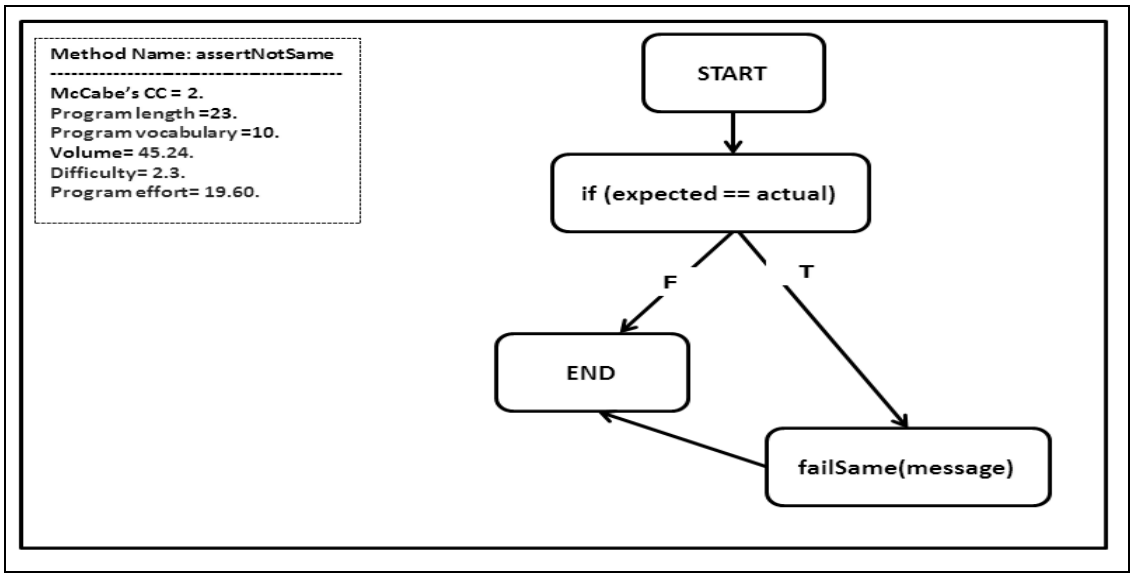

Figure 47
The Generated Method Control Flow Graph For assertNotSame Method

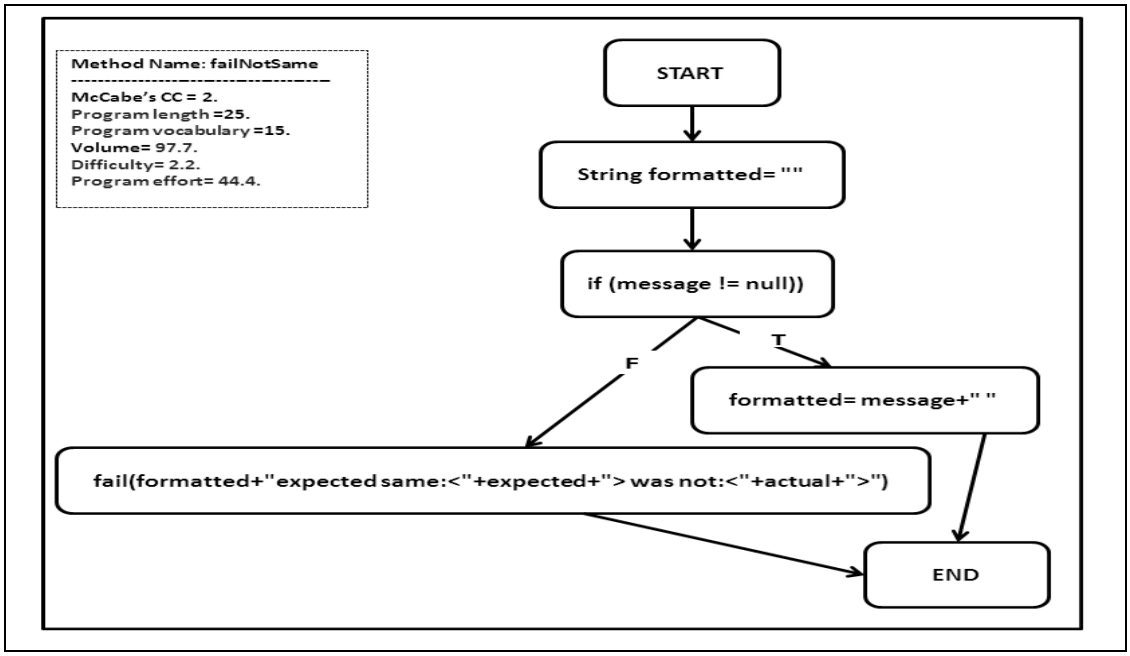

Figure 48
The Generated Method Control Flow Graph For failNotSame Method

Appendix 2:

The Generated Descriptive Summary for Case Study 2

```
1  Class ClassGraphHack report:
2
3  Class ClassGraphHack declared in package doclet as Private. Has a super class ClassGraph.
4  This class provide the following services:
5
6  The service is:  prologue.  This service returns void.
   This service use  methods: PrintWriter.
```

Figure 49
The Generated Class Report for ClassGraphHack Class

```
1  Class ContextMatcher report:
2
3  Class ContextMatcher declared in package doclet as public.
4  This class provide the following services:
5
6  The service is:  setContextCenter.  This service returns void. The service uses local data :(pattern with
7  type Pattern). The servise use the attributes: (pattern with type Pattern, matched with type List)
8  This service use local method: addToGraph.
9  This service use  methods: classes, matcher, and add.
10
11 The service is: addToGraph.  This service returns void. The service uses local data:(cd with type
12 ClassDoc). The servise use the attributes: (visited with type Set, cg with type ClassGraphHack, and opt
13 with type Options).
14 This service use methods: contains, add, printClass, printRelations, and printInferredRelations.
15
16 The service is: matches.  This service returns boolean. The service uses local data:(cd with type
17 ClassDoc). The servise use the attributes: (opt with type Options, and matched with type List).
18 This service use local method: addToGraph.
19 This service use methods: matchesHideExpression, contains, and matches.
20
21 The service is: matches.  This service returns boolean. The service uses local data:(name with type
22 String). The servise use the attributes: (pattern with type Pattern,  matched with type List, with type
23 ClassGraphHack, and opt with type Options).
24 This service use methods: toString, matcher, matches, getClassInfo, getRelation, and matchesOne.
25
26
```

Figure 50
The Generated Class Report for ContextMatcher Class

```
1  Class DevNullWriter report:
2
3  Class DevNullWriter declared in package doclet as Private. Has a super class Writer.
4  This class provide the following services:
5
6  The service is:  write.  This service returns void. This service use local data(cbuf with type char, off with
7  type int, and Len with type int).
8
9  The service is:  flush.  This service returns void.
10
   The service is:  close.  This service returns void.
```

Figure 51
The Generated Class Report for DevNullWriter Class

Class ContextView report:

Class ContextView declared in package doclet as public.
This class provide the following services:

The service is: setContextCenter. This service returns void. The service uses local data :(contextCenter with type ClassDoc). The servise use the attributes: (cd with type ClassDoc, myGlobalOptions with type Options, and matcher with type ContextMatcher)
This service use methods: containingPackage, setOption, and setContextCenter.

The service is: getDisplayName. This service returns String. The servise use the attributes: (cd with type ClassDoc).

The service is: getGlobalOptions. This service returns Options. The servise use the attributes: (myGlobalOptions with type Options).

The service is: getOptionsFor. This service returns Options. The service uses local data:(cd with type ClassDoc). The servise use the attributes: (globalOptions with type Options, hideOptions with type Options, cd with type ClassDoc, centerOptions with type Options, and packageOptions with type Options).
This service use local method: overrideForClass.
This service use methods: matchesHideExpression, equals, containingPackage, and clone.

The service is: getOptionsFor. This service returns Options. The service uses local data:(name with type String). The servise use the attributes: (matched with type List, hideOptions with type Options, cd with type ClassDoc, centerOptions with type Options, and globalOptions with type Options).
This service use local method: overrideForClass.
This service use methods: matches, name, equals, clone.

The service is: overrideForClass. This service returns void. The service uses local data :(opt with type Options, and cd with type ClassDoc). The servise use the attributes: (matcher with type ContextMatcher, HIDE_OPTIONS with type String, cd with type ClassDoc, and nodeFillColor).
This service use methods: setOptions, matchesHideExpression, setOption, and equals.

The service is: overrideForClass. This service returns void. The service uses local data :(opt with type Options, and className with type String). The servise use the attributes: (matcher with type ContextMatcher, and HIDE_OPTIONS with type String).
This service use methods: matches, and setOption.

Figure 52
The Generated Class Report for ContexView Class

Class RunDoc report:

Class RunDoc declared in package test as public.
This class provide the following services:

The service is: main. This service returns void. The service uses local data :(args with type String). The servise use the attributes: (docFolder with type String, and sourcesFolder with type String).
This service use local method: runDoclet.
This service use methods: File, exists, and mkdirs.

The service is: runDoclet. This service returns String. The service uses local data :(options with type String). The servise use the attributes: (pw with type PrintWriter).
This service use method: execute.

Figure 53
The Generated Class Report for RunDoclet Class

```
1   Class TestUtils report:
2
3   Class TestUtils declared in package test as public.
4   This class provide the following services:
5
6   The service is:  textFilesEquals.  This service returns boolean. The service uses local data :(pw with type
7   PrintWriter, refTextFile with type File, and outTextFile with type File).
8   This service use local method: runDoclet.
9   This service use  methods: println, BufferedReader , FileReader , refReader, readLine, outReader,
10  startsWith, equals, close, and print.
11
12  The service is: dotFilesEqual.  This service returns boolean. The service uses local data :(pw with type
13  PrintWriter, dotPath with type String, and refPath with type String).
14  This service use  method: println, DotDiff, differ, graphEquals, printList, getExtraLines1, getExtraLines2,
15  getNodes1, getNodes2, getArcs1, and getArcs2.
16
17  The service is: printList.  This service returns void. The service uses local data :(pw with type
18  PrintWriter, message  with type String, and extraOut with type List).
19  This service use  method: size, and println.
20
21  The service is: cleanFolder.  This service returns void. The service uses local data :(folder with type File,
22  and  recurse with type boolean).
23  This service use local method: cleanFolder.
24  This service use  method: listFiles, isDirectory, getName, equals, and delete.
```

Figure 54
The Generated Class Report for TestUtils Class

```
1   Class RunOne report:
2
3   Class RunOne declared in package test.
4   This class provide the following services:
5
6   The service is:  main.  This service returns void. The service uses local data :(args with type String). The
7   service use attribute: (testDestFolder with type String).
8   This service use local method: runSingleClass.
9   This service use  methods: File, exists, and mkdirs.
10
11  The service is: runView.  This service returns void. The service uses local data :(viewClass with type
12  String). The service use attribute: (testDestFolder with type String).
13  This service use local method: runDoclet.
14
15  The service is: runSingleClass.  This service returns void. The service uses local data :(className with
16  type String). The service use attributes: (testDestFolder with type String, and testSourceFolder with
17  type String ).
18  This service use local method: runDoclet.
19
20  The service is: runDoclet.  This service returns void. The service uses local data :(options with type
21  String). The service use attributes: (pw with type PrintWriter).
22  This service use method: execute.
```

Figure 55
The Generated Class Report for RunOne Class

```
1   Class Shape report:
2
3   Class Shape declared in package doclet.
4   This class provide the following services:
5
6   The service is:  graphvizAttribute.  This service returns String.
7   This service use  methods: equals.
8
9   The service is:  landingPort. This service returns String
10  This service use  methods: equals.
11
12  The service is:  extraColumn.  This service returns String . The service uses local data: (nRows with type
    int).
13
14  The service is:  cellBorder. This service returns String
15  This service use  methods: equals.
```

Figure 56
The Generated Class Report for Shape Class

```
1   Class RelationPattern report:
2
3   Class RelationPattern declared in package doclet.
4   This class provide the following services:
5
6   The service is:  addRelation. This service returns  void. The service uses local data: (defaultDirection
7   with type RelationType, and direction with type RelationDirection ).
8   This service use  methods: ordinal, and sum.
9
10  The service is:  matchesOne.  This service returns void. The service uses local data: (relationPattern
11  with type RelationPattern).
12  This service use  methods: contains.
13
```

Figure 57
The Generated Class Report for Class RelationPattern

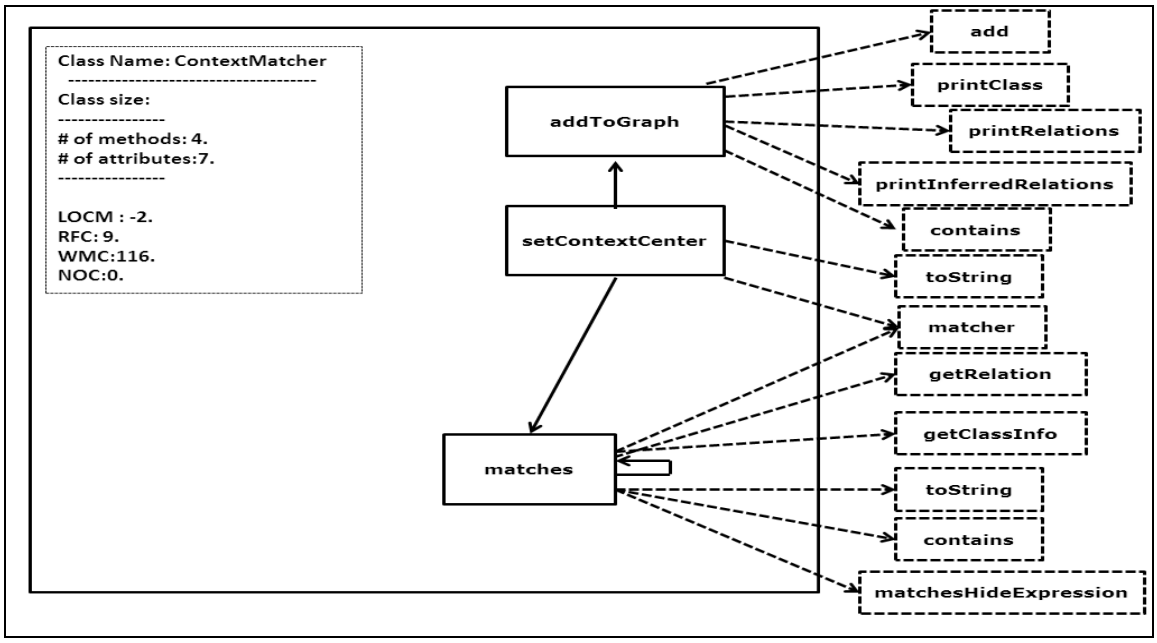

Figure 58
The Generated Class Call Graph for ContextMatcher Class

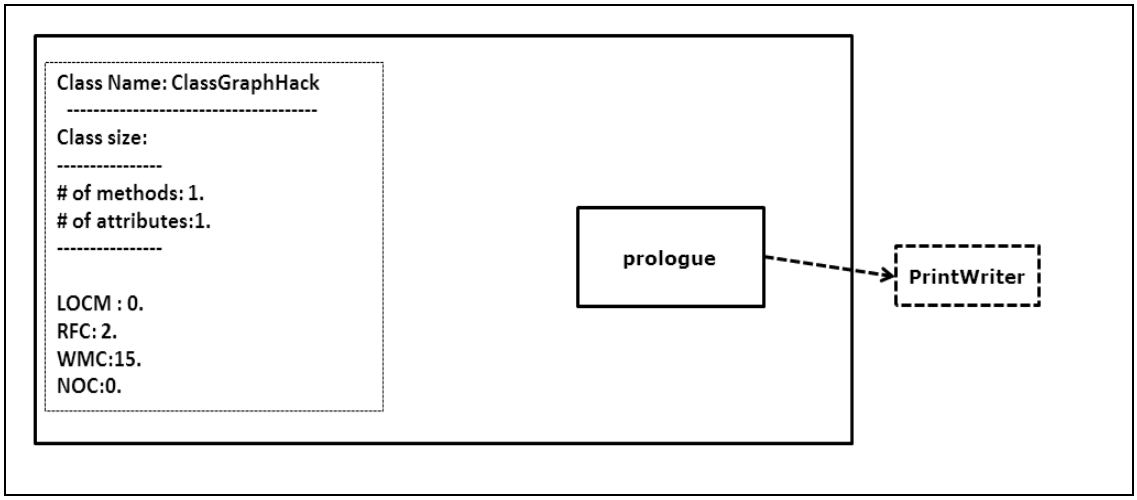

Figure 59
The Generated Class Call Graph for ClassGraphHack Class

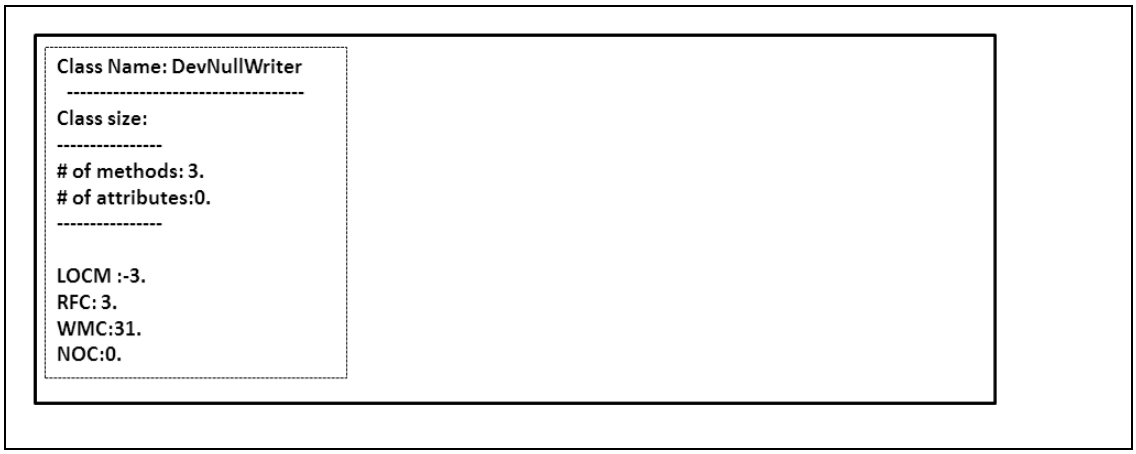

Figure 60
The Generated Class Call Graph for DevNullWriter Class

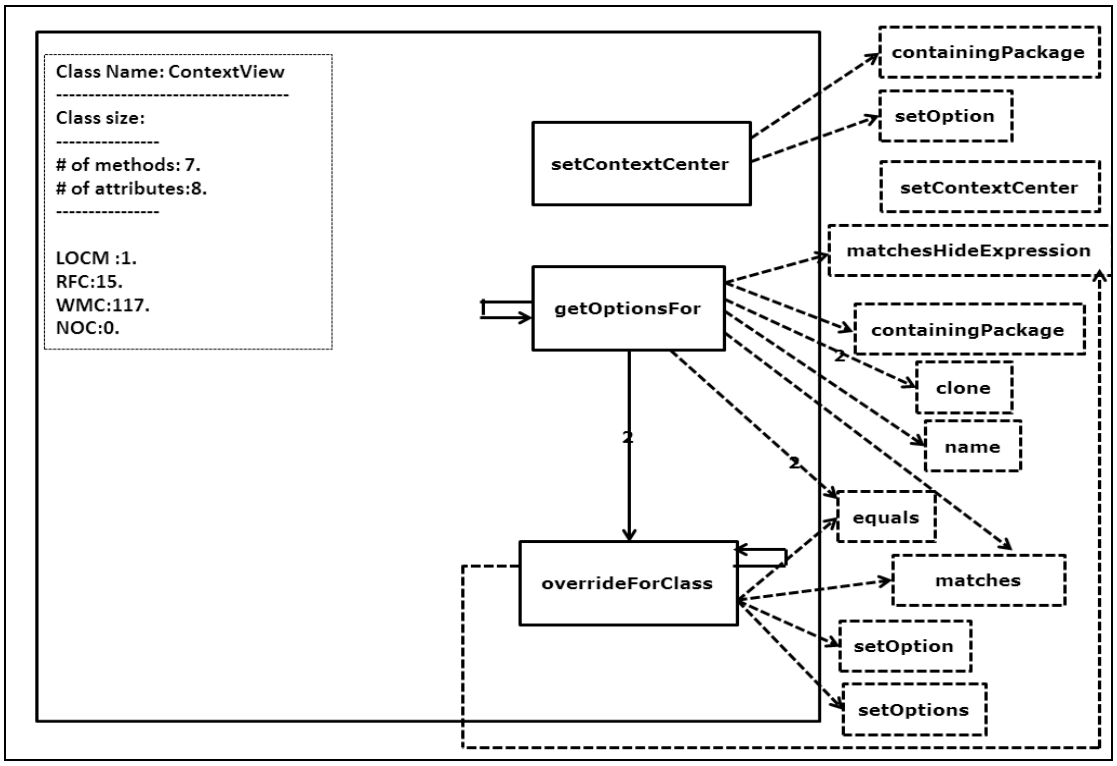

Figure 61
The Generated Class Call Graph for ContexView Class

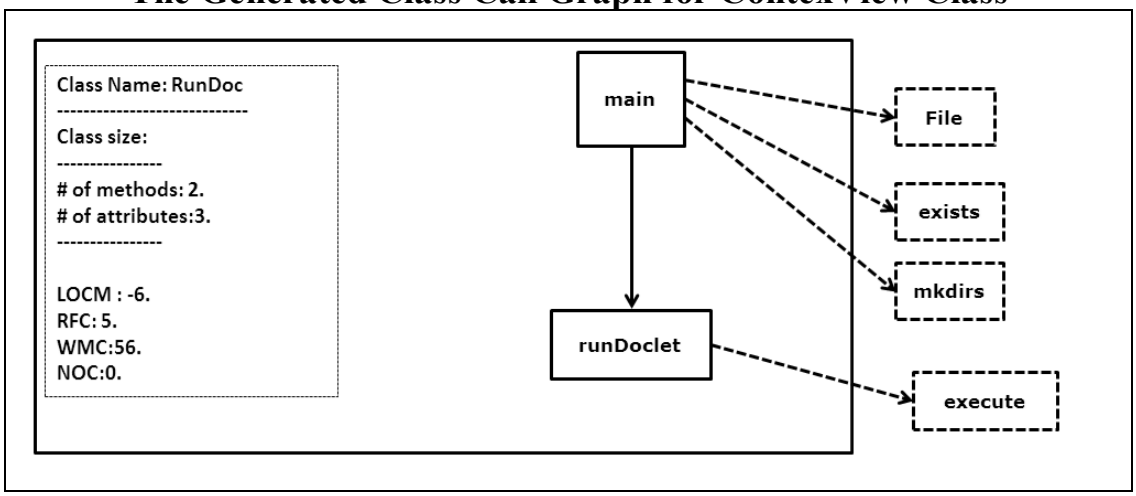

Figure 62
The Generated Class Call Graph for RunDoc Class

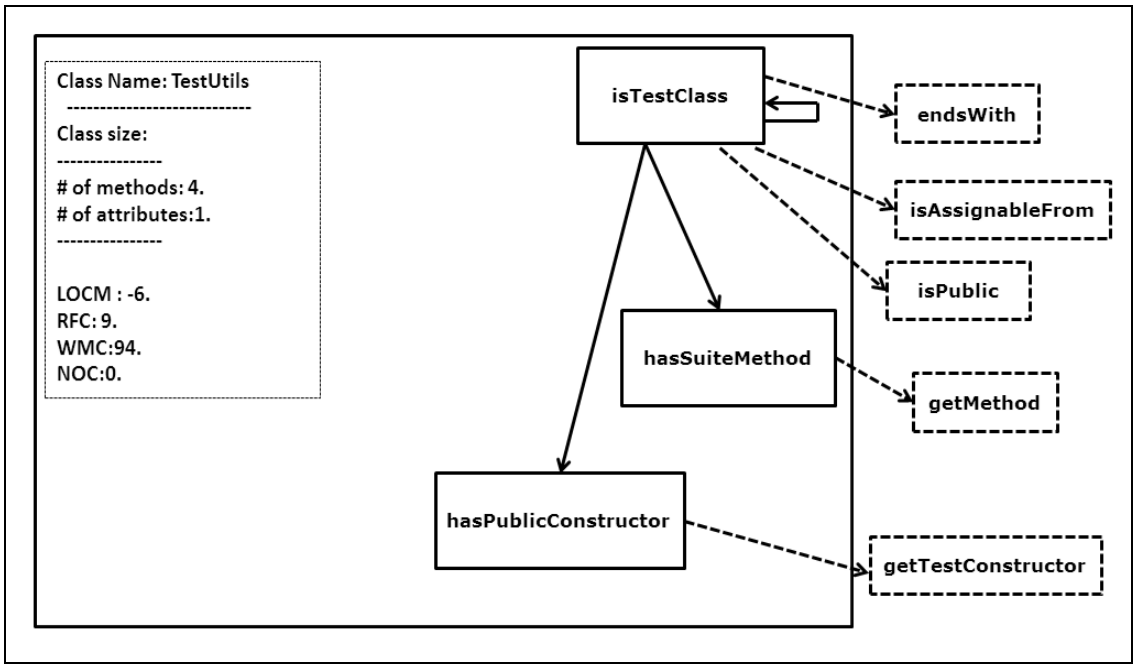

Figure 63
The Generated Class Call Graph for TestUtils Class

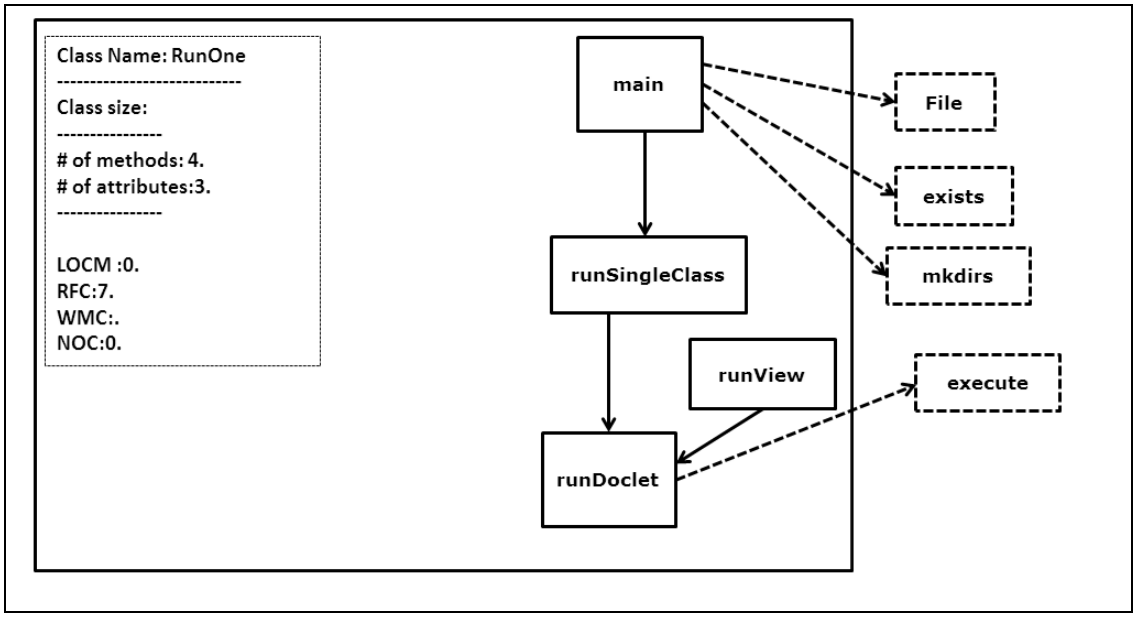

Figure 64
The Generated Class Call Graph for RunOne Class

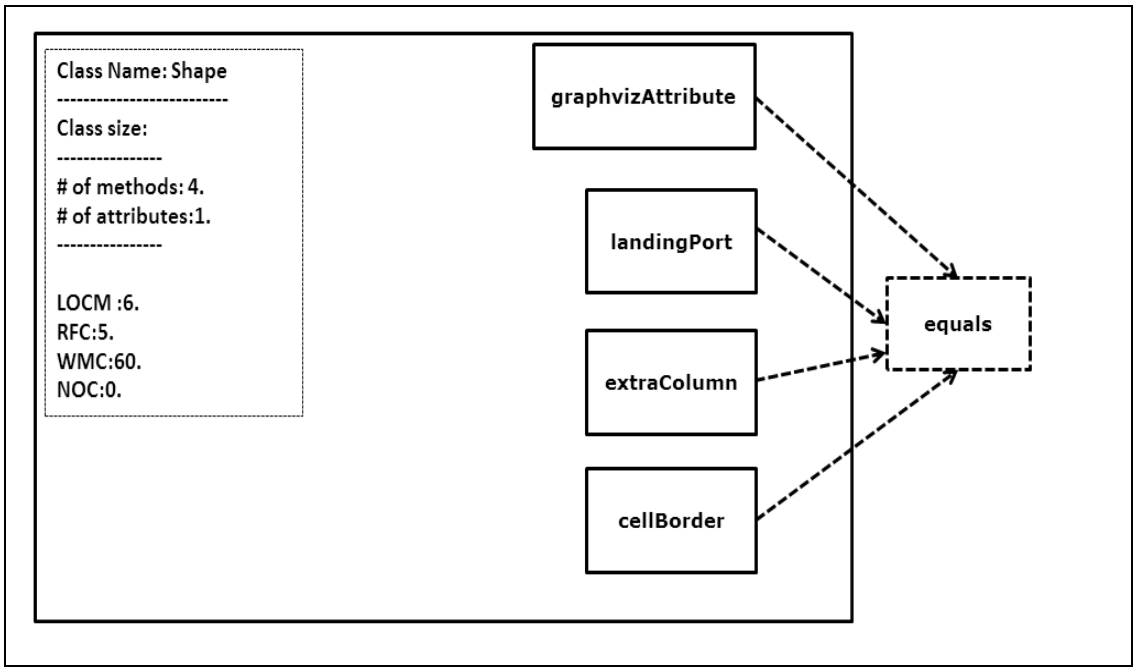

Figure 65
The Generated Class Call Graph for Shape Class

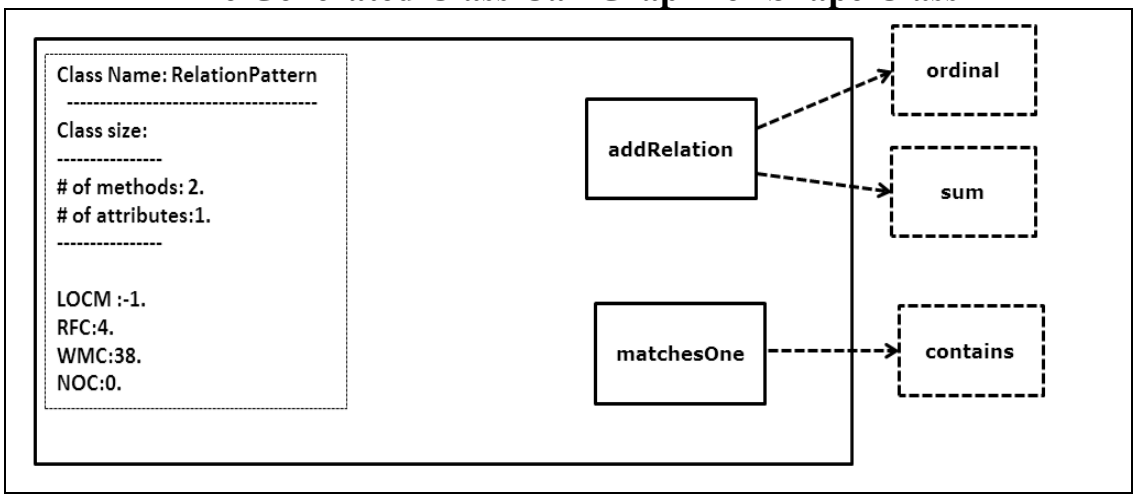

Figure 66
The Generated Class Call Graph for RelationPattern Class

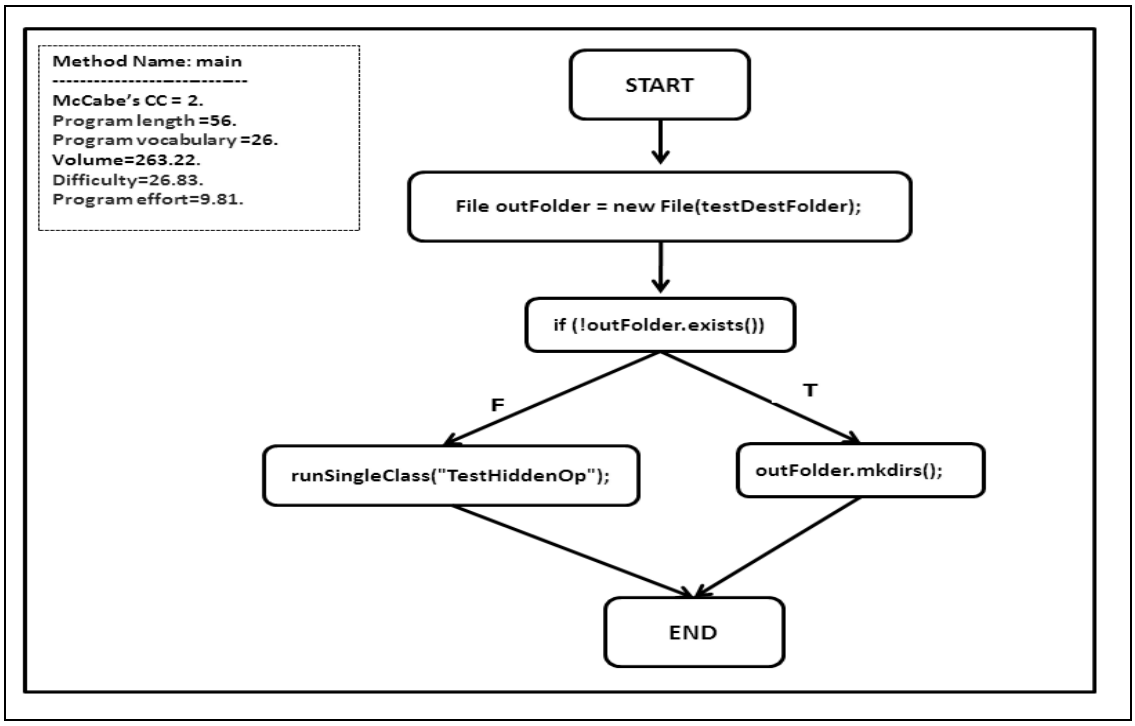

Figure 67
The Generated Method Control Flow Graph For main Method

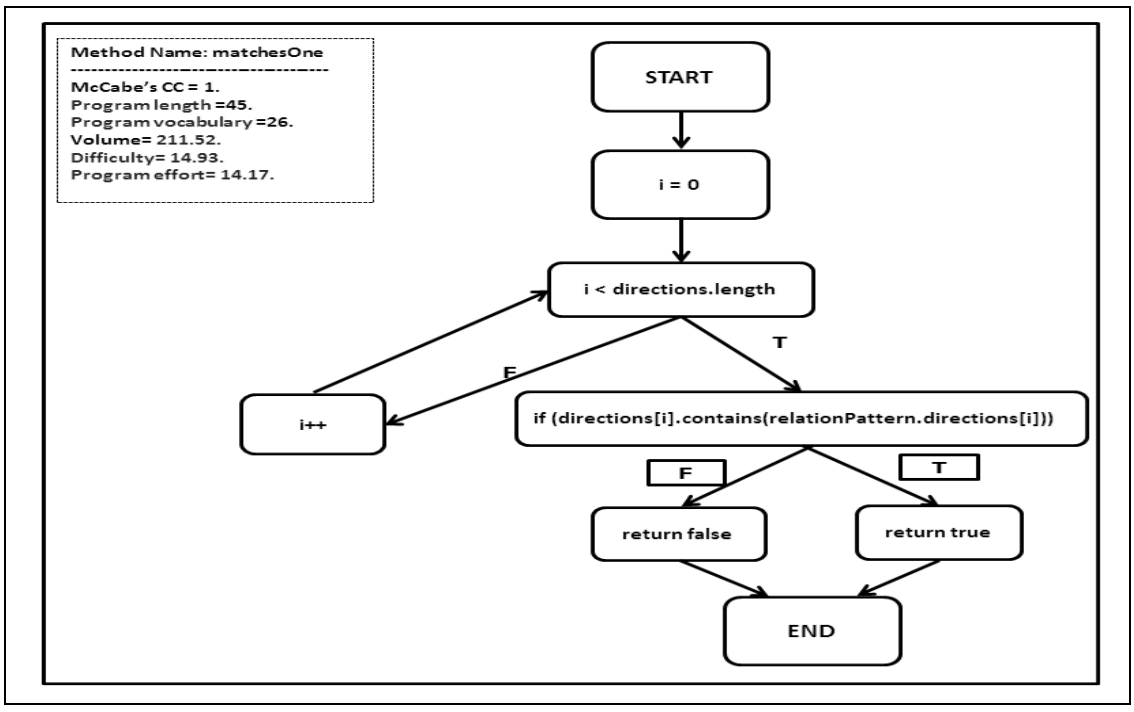

Figure 68
The Generated Method Control Flow Graph For matchesOnes Method

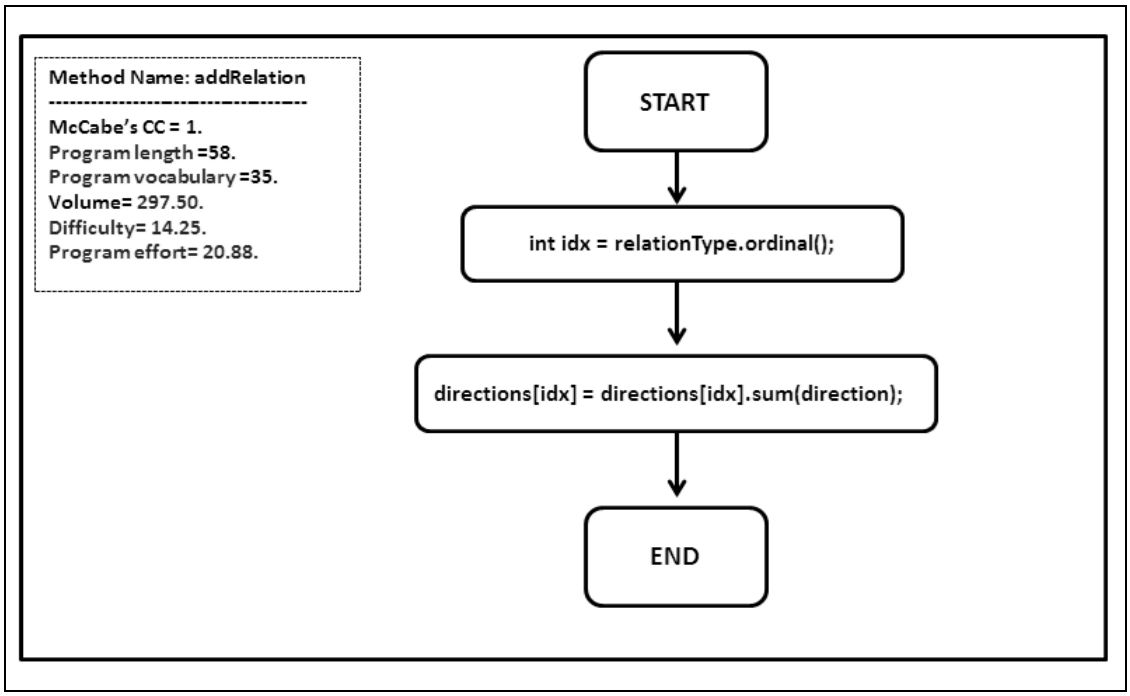

Figure 69
The Generated Method Control Flow Graph For addRelation Method

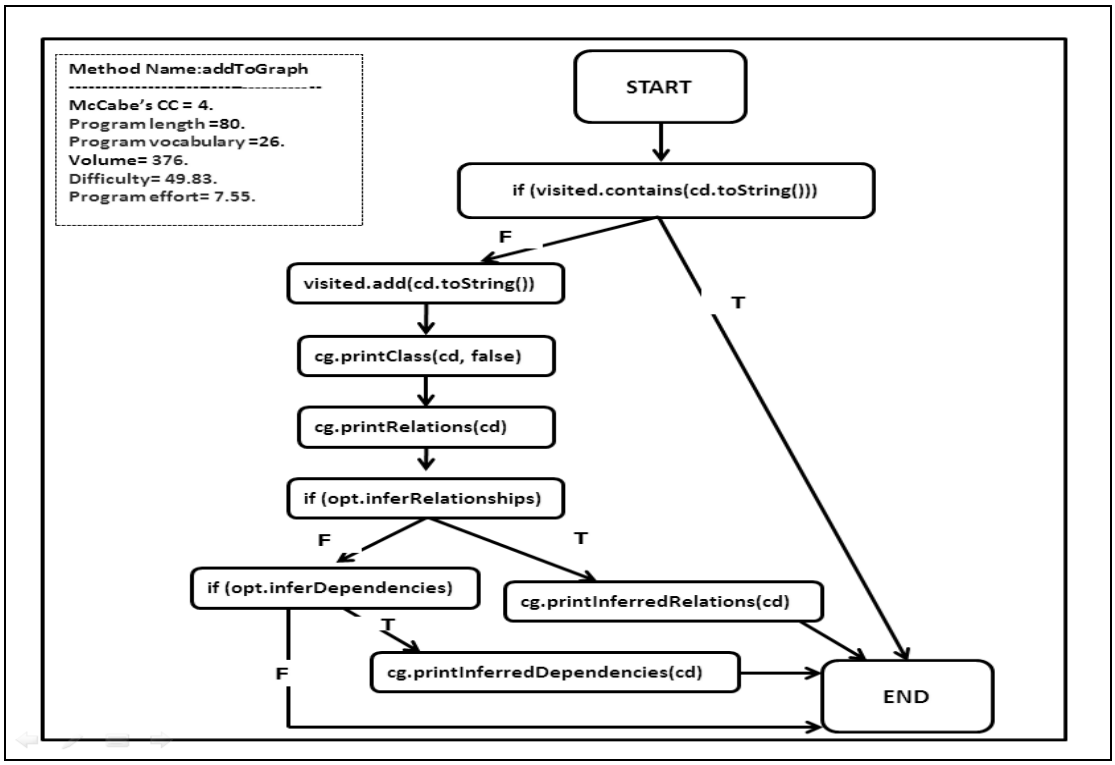

Figure 70
The Generated Method Control Flow Graph For addToGraph Method

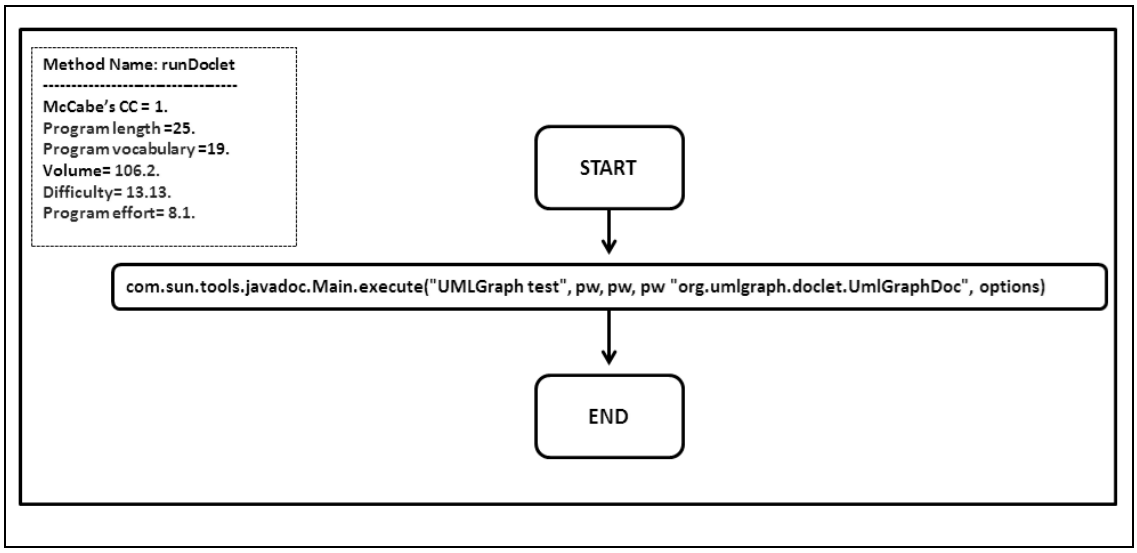

Figure 71
The Generated Method Control Flow Graph For runDoclet Method

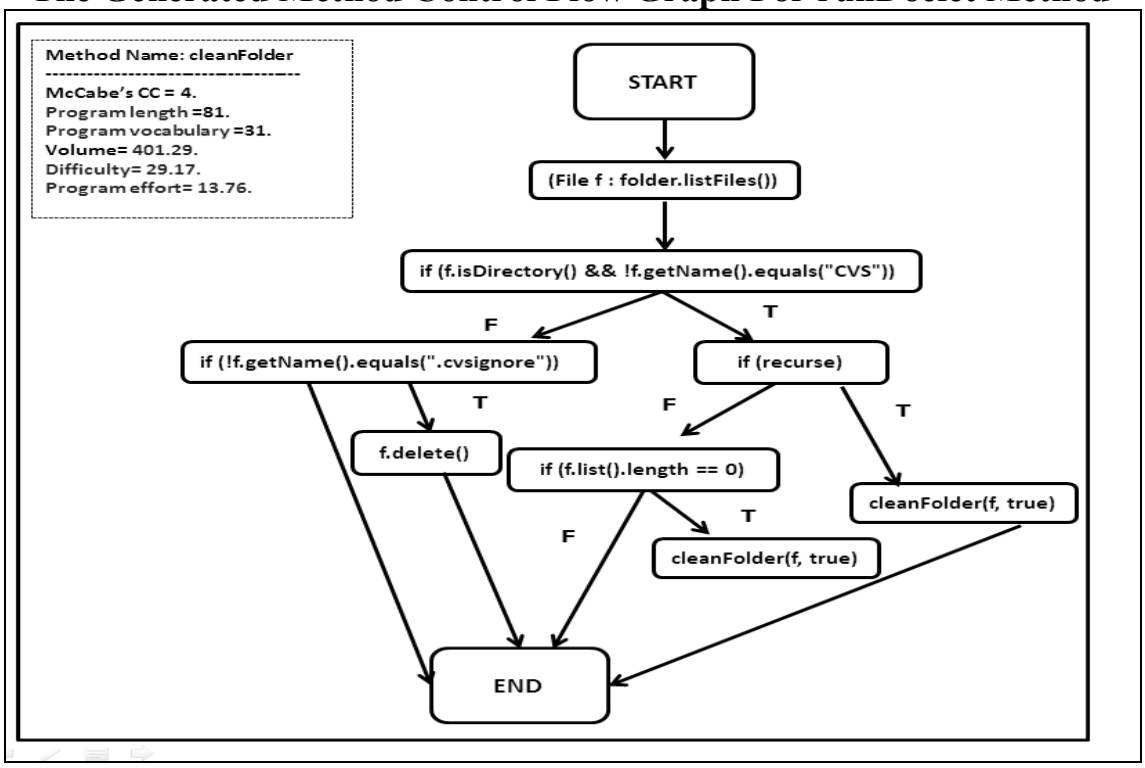

Figure 72
The Generated Method Control Flow Graph For cleanFolder Method

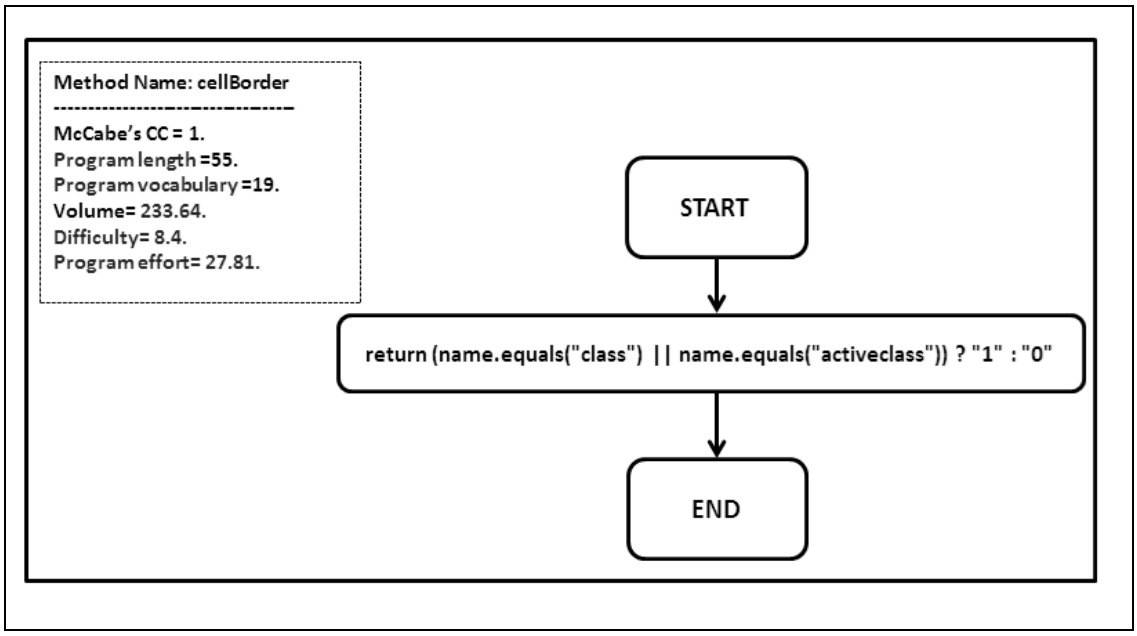

Figure 73
The Generated Method Control Flow Graph For cellBorder Method

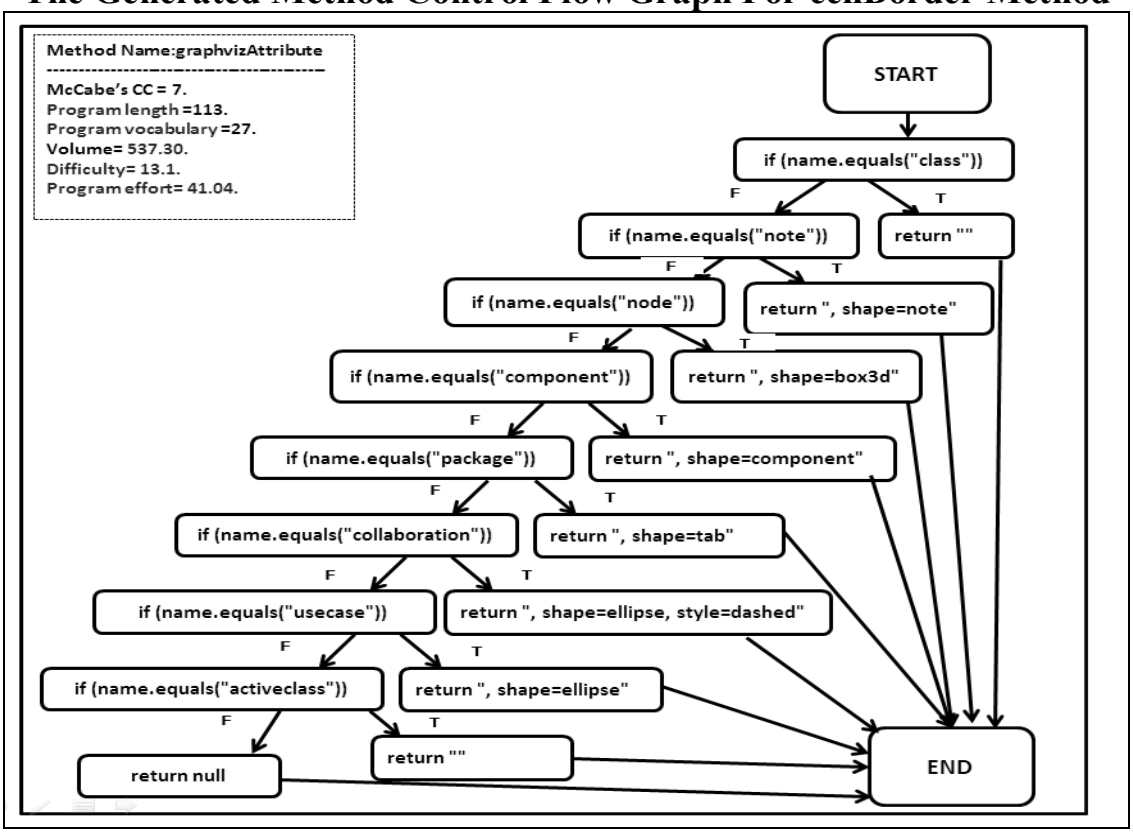

Figure 74
The Generated Method Control Flow Graph For graphvizAttribute Method

العنوان:	Automatic Generation of Descriptive Summary for Source Code Artifact
المؤلف الرئيسي:	البطوش، أماني عبدالسلام
مؤلفين آخرين:	حماد، مصطفى محمد(مشرف)
التاريخ الميلادي:	2016
موقع:	مؤتة
الصفحات:	1 - 86
رقم MD:	951089
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	جامعة مؤتة
الكلية:	عمادة الدراسات العليا
الدولة:	الاردن
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات، برمجية جافا، برمجية سي
رابط:	https://search.mandumah.com/Record/951089

Mutah University
College of Graduate Studies

Automatic Generation of Descriptive Summary for Source Code Artifact

By
Amani Abdel-Salam Al-Btoush

Supervisor:
Dr. Mustafa Hammad

A thesis Submitted to the College of Graduate Studies in partial fulfillment of the requirements for the Master's degree in Computer science to the Department of Information Technology, University of Mutah.

Mutah University, 2016