IMPLEMENTATION OF SOFTWARE ARTIFACT RECOVERY COMPLEXITY TOOL

Safia Sultana[1],Nadim Asif[2], Mazhar Hussain Malik*[3], Maliha Rafi [4]
[1,4]Department of Computer Science & IT, Institute of Southern Punjab, 9-KM, Bosan Rd, Multan
[2]Department of Computer Science, Bahria University,Lahore, Pakistan
[3]Department of Informatics, Universita Ca Foscari, Venezia, Italia
*Email: malik@dsi.unive.it (Corresponding Author)

ABSTRACT*—Modern software development projects are getting larger in size, are being developed using different software languages and using many third party libraries as well as frameworks. ARC is the tool that measurse the complexity of recovered artifacts. Artifacts are generating automatically during developing the software at different steps. In this research paper, four types of abstraction layer is defined as grouping principle, where a hierarchy is remained to with higher levels of abstraction near the top with more concept beneath. These artifacts are recovered and analyzed how much they are complex with the help of Artifact Recovery Compexity (ARC) tool. Artifact Recovery Complexity tool has some weight points at every level of abstraction layer. The software artifacts also have some dependencies whom which a source code complexity is depend upon. These dependencies are Size of source code (SSC), Source Code Type (SCT), Abstraction Level (AL) and Documentaiton Type Support (DTS). We used ARC tool, to measure and understand the code complexity impact using thirty one format string using fixed vulnerabilities.We analysed the impact of code complexity on the quality of results, which includes the successful detection along with false positive rates. Artifact Recovery Complexity tool is able to detected 90% of the format string vulnerabilities, result shows direct relation of code complexity with detetcion rate, when code complexity increased, error detetcion automatically decreases.*

Keywords-ARC, SSC, SCT, AL, DTS, Artifacts

I. INTRODUCTION

The software maintenance is the tumult where performances of all the activities need to keep a software system proactive after it is acknowledged and placed in the production. The software maintenance activities are classified into four major types, perfective, adoptive, corrective and preventive. This classification based on modifying program to produce new outputs, to change executing logic, new features amalgamation, to change or correct the existing features, to correct errors in the existing code during the system code optimization to meet different hardward and software enviroments.

Software maintenance is very extensivebustle and it instigate after deliverance of the correct faults, Performance improvement or adopting a change change environments. Software maintenance is the post-delivery activity and which makes the maintenance hard[15].

Software maintenance consist of those mandatory activities which leads to provide cost-effective support to software system. Pre-delivery activities based on planning for the post-delivery operations, supportability and rationality willpower. Post-delivery activities consist on software amendment, preparation and operating a help desk.

A. Software Artifact

The software systems are composed of different types of artifacts which are compulsory to extract different abstractions levels for the maintenance activities[4]. Software systems can be designed in different language which lead to high techincal complexity. It is very necessary to understand and extract the system documents from these complex systems for maintenance, reuse or re-engineer the systemsSoftware are imperceptible therefore software visualization is needed in textually [11].

*1) Software Artifacts Types:*A software system has a lot of artifacts types like source code, design documents, specification documents and programmer knowledge and experience that are most important for the reverse engineering efforts.

2) Abstraction Levels of Artifacts: Software consists of several layers of abstraction built on top of raw hardware; the lowest-level software abstraction is machine code, or object code (Figure 1). In the context of software maintenance, four levels of reverse engineering abstraction are defined: implementation abstraction, structural abstraction, functional abstraction and domain abstraction [7].

B. Software Complexity

Complexity arises in software in many forms, here is a little analysis of software complexity and how developers come across in software development[17]:

- At the domain level complexity in the form of lack of deep knowledge about domain and Analysis-paralysis,
- At the team level complexity is in form of many sub-modules and may be a project has many stakeholders.
- At the level of chaotic code flow problems in the form of algorithmic complexity.
- A bundle of ambiguous user interface design that causes the design coding complexity that also consisting the hundred of table columns, designing the hundred of documents.
- Another form of complexity is arises in coding languages. Code is not written in extraordinary language type.
- Real time and embedded system software are another big issue of software complexity.

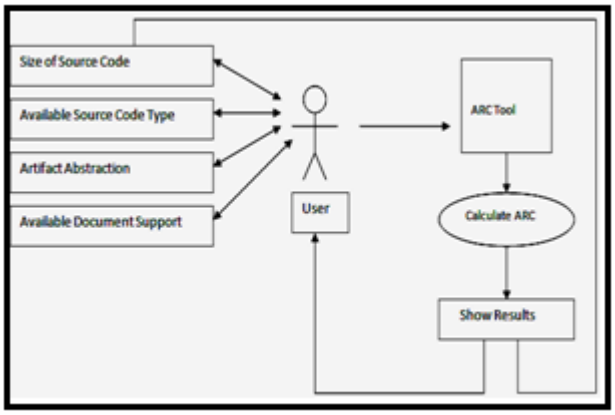

Figure 1: Artifact Arbstraction Levles X

C. Recovery of Artifacts

We recovered the artifacts from the source code just to plan, resign, re-implement the system or just make changes in the current system to meet the current requirements [4]. However it is well known fact that even in organization and developing project with mature development processes, software artifacts created as part of these processes at the end to be disconnected with each other.

The software artifacts require for maintenance purpose are different at the every level of abstraction. We must need to know that:

- The software developers have specific aim for maintenance tasks at hand.
- Which type of artifacts are required and at what level of abstraction?

D. Designing the ARC Tool

It has been describe that Artifact Recovery Complexity depend upon the four type of source code dependencies that are; (i) size of source code, (ii)level of source code type, (iii) artifact abstraction levels (iv) documentation type support to recover the artifact for maintenance purpose as shown in Figure 2. Here we used the term "Degree" which is two type "Recoverness" that is degree to which the artifact recovery related with other artifact for maintenance task and another type is "Recovership" that is the degree to which the artifact recovery related with other artifact for maintenance task.

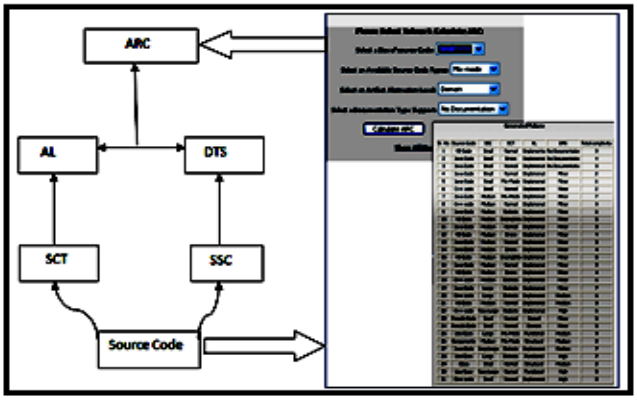

Figure 2: Designing the ARC Tool

II. MATERIALS AND METHODS

The artifact recovery complexity is the formulated method by which we can measure the complexity of software artifact. This research is about to implement an ARC tool that measure the complexity of recovered artifacts. A bundle of devices and methods have been developed for the software programmers.

Software reflexion models can help programmers to define the large systems by highlighting, how the actual system is associated with high level explanation of prospect [13]. Response models are as precious as developers that analyze the structure of large software systems. The Rigi system also uses visual techniques to provide programmers with a graphical presentation of the structure of a software system [13]; Above all Rigi is designed to support reverse engineering responsibilities where a programmer must accomplish a working understanding of an unfamiliar software artifact [13]. These tools provide the software developer precious imminent into the structure of the system under examination, but goal mentioned here is rather different. Here we are going to implement a tool which measure the complexity of software artifact which generate during developing the software.

The artifacts recovery complexity depends, degree of source code type, abstraction level, on the size of source code and the degree of available document support to recover the artifacts for task at hand. First I explain source code, artifacts for maintenance, available source code and documentation nature, extraction of artifacts, presentation of artifacts.

A. Architecture of Artifact Recovery Complexity Tool

In the world of software industry; software tools are play an important function to achieving high self assurance that a software system convinces critical properties. The Figure 2 describe the four different functions where tool interact to improving the quality of both software system and software system artifacts.

B. Source code

Source code is a piece or code which is written any designated programing to fulfill the requirement to design and run the system. Normally, these codes are wriiten in high languages which are understand by the computer using a compiler/ interpretation steps which translate the high languages computer languages files into machine understanable form. Source code can also be stored in a database as is common for stored procedures or elsewhere. The conception of source code may also be in use more broadly to include machine code and notations in graphical languages neither of which are textual in nature[11].

For the meticulousness 'source code' is taken to signify any fully executable description of a software system. It is hence construed as to include machine code very high level languages and executable graphical representations of systems.

C. Dependency of Artifact Recovery Complexity

Recovery of software artifacts is a complex factor. The recovery of software is easily done with the help of different tools but the recovery of software artifacts which are written in different languages (human languages and programming languages or in visual format) is a complex factor as compare to software recovery [17]. The following dependencies factors are proposed to measure the artifacts recovery complexity:

- Size of available source code
- Degree of available source code type
- Abstraction level of source code
- Degree of available documentation Type support

These are the four dependency levels which are required to recover the artifacts for task at hand. Now we explain them in detail.

i. Size of source code (SSC): Size of source code is very important. The code written in large size for large system and a small type module has a small size of source code. The increasing the source code makes it more complex. A large size of source code consists in thousand lines. Recovery of these artifacts is a big issue. In fact size is an important factor and plays an important role to recover the software artifacts. Size of source code from which the artifacts are recovered for maintenance purpose, and the size of source code falls into four categories, every category of source code size have a specific weight points. as shown in the table 1.

Table I: Source Code Size (Ssc)

Size of source code	Description	Weight
Small	Few thousand code lines	1
Medium	above 10,000 code lines	2
Large	Above 100,000 code lines	3
Very Large	Above one million code lines	4

ii. *Available Source Code Type (SCT):* The source code exist in many forms, it is consist of more then one programming languages. It could happen that available source code is written in different version of one language. e.g. source code written in version C language and next time when system module is updated for enhance purpose, remaining code written in C++ version and next time in visual C++.

So the dialect of this type of source code makes it complex and its recovery is another big issue. It could be happen that source code is incomplete or there are errors in source code[23]. Recovery of such type of source code that is incomplete or consisting errors is easy rather then source code is in mix-mode or in dialects form. So we assign the weight regarding their available source code type discussed in table 2.

Table 2: Source Code Type (Sct)

Source Code Type	Description	Weigh t
Mix-Mode	The Source Code in multiple languages	5
Dialects	The source code is in different dialects	4
Incomplete	Incomplete source code	3
Errors	Error in Code compilation	2
Normal	Source code written in single language	1

iii. *Abstraction Level (AL):* Each abstraction level has specific weight points, so we assign it for computing the ARC. These abstraction levels and regarding weights are shown in table 3.

Table 3: Artifact Abstraction Level (Al)

Abstraction Levels	Description	Weight Point
Domain	High level entities describe the system software	4
Functional	Artifacts like in Use Cases and Scenarios	3
Structural	Components,/ classes type artifacts	2
Implementation	Files, Function definition and procedure call etc.	1

iv. *Documentation Type Support (DTS):*Documentation is very important in any case. The software documents are also plays an important role to measure the complexity of recovered artifacts. The last dependency which is needed to measure the Artifact Recovery Complexity is available Documentation Type supports that recover the software artifact for maintenance purpose. The table given below show that the document type and the weight that we assign to the appropriate document.

These are four types of dependency levels that are very important to calculate the Artifact Recovery Complexity of available source code. Following is the formula for calculating the ARC.

In this formula, SSC (Size of source code), SCT (Source Code Type) and AL (Abstraction Level) are added and DTS (Documentation Type Support) is subtracted from all of this. So known about size of a source code, its type and its abstraction level is important other then for ARC calculating DTS (Documentation Type Support) is very important.

$$ARC = SSC + SCT + AL - DTS$$

Table 4: Document Type Support (Dts)

Documentation type support	Documents Description	Weight
Documentation	There is no documentation support	0
Minor	There is only system and components information.	1
Medium	Requirements, Design and implementation details subsist	2
High	Complete Requirements, Design and implementation details for recovery task	3

III. RESULTS AND DISCUSSIONS

To comprehend the produce of code complexity on ARC Tool, thirty one design string susceptibilities were considered. A class is the just name of object and how these objects interact each other, but when we design this class. A class must be explain by its attributes and the methods how this class working. We analyzed different case studies for every susceptibility. We observe the effect of code complexity on the quality of results, including successful detection and false positive rates. ARC Tool detected 90% of the format string vulnerabilities, with detection rates decreasing with increasing code complexity. When the tool failed to identify the code complexity, it was for one of four reasons:

- The size of source code is small
- Available source code is normal
- Artifact abstraction level is implementation
- And documentation support is high in the ARC tool.

Complex code is more likely to hold complex code constructs and ambiguous format string functions, resulting in lower finding rates.

A. Test Procedures

Thirty one format string vulnerabilities source code written in C or C++, VB Code, Java, Pseudo Code, Component, Use Case, Classes are randomly selected from Vulnerability Database are used. For a test case to be evaluated, source codes are available. The Test cases that did not meet these criteria were replaced with another test cases selected randomly.

For each source code, four types of artifact recovery complexity dependencies are calculated. The first type whom which artifact recovery complexity is depend is size of source code, second is degree of source code type, third is abstraction level and the fourth one is available document type support.

The first artifact recovery complexity dependency is size of source code. The size of source code must be small, medium, large and very large. If the size of source code is consist on few hundred lines, it is consider small and its weight age value is '1'. If the size of source code is more then 1000 lines, it is considered medium and its weightage value is 2. If the size of source code is more then 100,000 lines, it is considered large and its weightage value is 3. If the size of source code is more then 1 million lines, it is considered vary large and its weightage value is 4. In this way size of source code is calculated.

The second artifact recovery dependency is available source code type. Available source code is five types. The first type is Mix-mode, its mean that source code is written in multiple languages and weight given to mix-mode is highest value that is 5. It's most complicated to analyze this type of code, that's way its value is highest. The second type of available source code is Dialects. The dialects type of code is also complicated because it consists of different version of a language. That's way it is complicated and value given to it is 4. The third type of available source code is Incomplete where you have code that is not complete the weightage value is 3. The fourth type of available source code is Errors. Source code is available but consists of errors and not successfully compiled. So its weightage value is 2.the fifth and last type of available source code is Normal. The source code exists in a single language and its weightage value is 1. So our available source code is calculated.

The third dependency of artifact recovery complexity is artifact abstraction level. Artifact abstraction levels are four types. The first type of abstraction level is Domain where the high level entities describe the system. These high level entities are application domain and algorithm. Analyze the application domain and algorithms are most complicated that's way its weightage value is highest that is 4. The second type of abstraction level is Functional where artifacts are like use cases and scenarios. The recovery of use cases and scenarios are completed so here is also complexity is high and functional weightage vale is 3. The third type of abstraction level is Structural where artifacts are in the form of component and architecture classes and its weightage value is 2. The fourth type of artifact abstraction level is Implementation. Here source code artifacts are in the form of files, functions definition and program calls etc. The recovery of implementation level is easy so its weightage value is only 1. Now we have calculates the abstraction level.

The fourth artifact recovery complexity dependency is Documentation Support type. There are four type of documentation. First one is, it may be happen that you have source code but have not documentation. If a source code has no documentation support, it has no weightage value and it given to a 0 value. The second type of documentation support is minor. Here we have only system/component detail and no other document is available. The weightage value of minor is 1. The third type of documentation support is Medium, where we have some requirements, design and implementation details and its weightage value is 2. The fourth type of documentation support is High. In the high documentation included requirements, design and implementation details support fully the recovery the tasks. The weightage value of documentation support is 3.

These are the four types of artifact recovery complexity dependencies, which applied to recover the artifacts. The formula which calculate the artifact recovery complexity is Artifact Recovery Complexity:

$$(ARC) = SSC + SCT + AL - DTS$$

B. RESULTS

The four dependency of artifact recovery complexity is measured. And every source code has its own individual result. The highest value show that recovery complexity for this is most complicated and the lowest value indicate that recovery complexity for this artifact is easiest. Artifact recovery complexity for any type of software artifact can be measure by ARC Tool. The results is shown by graph that how the values are varies at different levels.

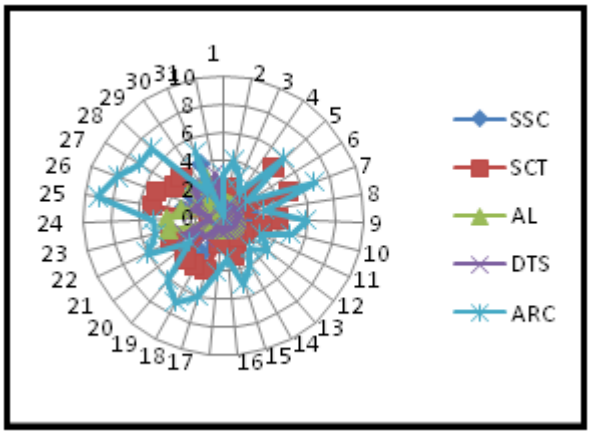

Figure 3: ARC for 31 Case studies

The graph shows the variation for each artifact recovery complexity dependency. The highest complexity is found in Task 25 that is 9 and lowest complexity found in Task 31 that is 0. Which mean that to recover of this task is very easy. Now artifact recovery complexity is shown below for every dependency by graph.

C. *ARC at Different Levels of Abstraction*

i. ARC at Level 9

Task 25 found a high artifacts recovery complexity, here is analysis of this task. The first dependence is size of source code (SSC) which is 3; its mean size of source code lies in more then 100,000 lines of code. The second dependency is available source code type (SCT); its value is 5 which mean that source code is written in multiple languages. So analyze them and recovered it most complicated. The third dependency is artifact abstraction level (AL) which calculated value is 3: its mean artifact at this level is in the form of use cases and scenarios. The recovery of use cases and scenario is most complicated because if the system modification due to any internal or external reason, its scenario will also change. So at the functional level artifact recovery complexity is high. The fourth dependency of artifact recovery complexity is documentation support. Here calculated value is 2, so the complexity is medium type here. The medium documentation support has some requirements; design and implementation details exist for support. The artifact recovery complexity is maximum here due to these four types of dependencies.

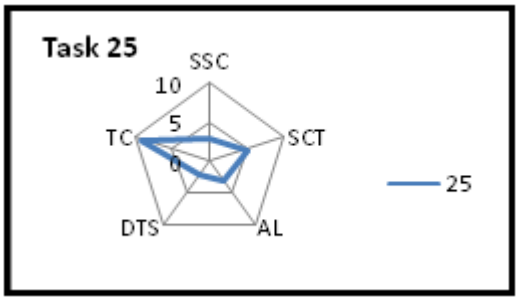

Figure 4: ARC at level 9

ii. ARC at Level 0

Task 31 has a lowest complexity and that is 0; its mean that artifacts are easily recovered. Now we analyze what are the dependencies values are found here. First artifact recovery dependency is size of source code which has 1. So here is small size of source code that consisting on few lines. These lines are easily recovered, count and reconstruct. The second dependency is source code type. The source code type is normal and consisting on single language. The value found here is 1. The fourth dependency is available abstraction level. Abstraction level is implementation type that consists on program files, function definition and function calls etc. This abstraction level value has 3. So the artifact recovery complexity is 0.

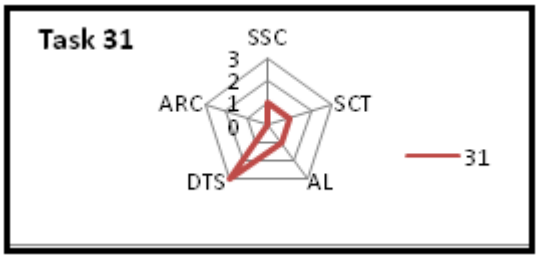

Figure 5: ARC at level 0

We can easily compare the artifact recovery complexity (ARC) of both tasks. So if SSC in small, SCT in normal, AL in implementation level and DTS is in medium form. We have the artifact recovery complexity is 0, and is we have SSC in large, SCT in Mix-mode, AL in Functional level and DTS is in medium form. We found the high level complexity to recover the artifacts.

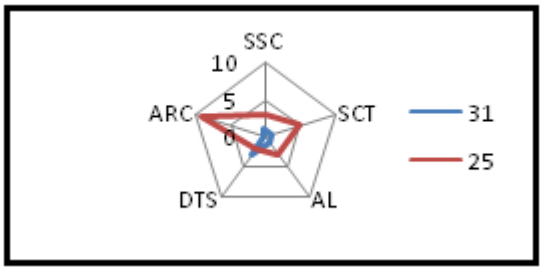

Figure 6: Comparison level 0 and level 9

IV. CONCLUSION

The ARC Tool calculates the complexity of every recovered artifact and return a value. With the implementation of this tool, it has been concluded that with very large size of source code is available and in the mix-mode of source code type, the recovery of artifacts is most complex. At the domain level of abstraction layer if source code has not any type of documentation, the recovery of artifacts is also most complicated. ARC must be high at these dependencies levels. So if the size of source code is small with normal type of source code and its abstraction level is at implementation and if documents are fully available the ARC complexity is at zero level because we have full documents support so recovery of artifact is easily recovered.

IV. REFERENCES

[1]. Asif, Nadim, et al. "Recover the design artifacts." proceedings of International Conference of Information and Knowledge Engineering (IKE02). 2002.

[2]. Asif, Nadim, et al. "Clustering the source code." WSEAS Transactions on Computers 8.12 (2009): 1835-1844.

[3]. Asif, Nadim, et al. "Recover the design artifacts." proceedings of International Conference of Information and Knowledge Engineering (IKE02). 2002.

[4]. Asif, Nadim. "Reverse Engineering Methodology to Recover the Design Artifacts: A Case Study." Software Engineering Research and Practice. 2003.

[5]. Asif, Nadim. "Artifacts Recovery at Different levels of Abstraction." Information Technology Journal 7.1 (2008): 1-15.

[6]. Micro, Bianco at el. " Extracting and analyzing software code metrics from C# source code." Center for Applied Software Engineering Free University of Bolzano-Bozen.

[7]. Chikofsky, Elliot J., and James H. Cross. "Reverse engineering and design recovery: A taxonomy." Software, IEEE 7.1 (1990): 13-17.

[8]. Koschke, Rainer. "Atomic architectural component recovery for program understanding and evolution." (2000).

[9]. Koschke, Rainer. " Atomic Architectural Component Recovery for Program Understanding and Evolution." Proceedings of the International Conference on Software Maintenance (2002).

[10]. Koschke, Rainer. "Software Visualization in Software Maintenance, Reverse Engineering, and Re-engineering: A Research Survey." Journal of Software Maintenance and Evolution: Research and Practice J. Softw. Maint. Evol.: Res. Pract.: 87-109.

[11]. Eichberg, Michael. Open Integrated Development and Analysis Environments. Diss. TU Darmstadt, 2007.

[12]. Murphy, Gail C., David Notkin, and Kevin Sullivan. "Software reflexion models: Bridging the gap between source and high-level models." ACM SIGSOFT Software Engineering Notes 20.4 (1995): 18-28.

[13]. Murphy, Gail C., and David Notkin. "Reengineering with reflexion models: A case study." Computer 30.8 (1997): 29-36.

[14]. Pigoski, Thomas M. Practical software maintenance: best practices for managing your software investment. John Wiley & Sons, Inc., 1996.

[15]. Rasool and Asif. Design Recovery Tool. International Journal of Software Engineering. 2007

[16]. Banker, Rajiv D., et al. "Software complexity and maintenance costs."Communications of the ACM 36.11 (1993): 81-94.

[17]. Banker, Rajiv D., et al. "Software errors and software maintenance management." Information Technology and Management 3.1-2 (2002): 25-41.

[18]. Hennicker, Rolf, and Nora Koch. "Systematic design of Web applications with UML." Unified Modeling Language: Systems Analysis, Design and Development Issues (2001): 1-20.

[19]. Swanson, B. E., and I. S. Maintainability. "Should It Reduce the Maintenance Effort." Conference on Maintenance, New Orleans. 1999.

[20]. Sarma, Anita, Zahra Noroozi, and André Van Der Hoek. "Palantír: raising awareness among configuration management workspaces." Software Engineering, 2003. Proceedings. 25th International Conference on. IEEE, 2003.

[21]. Singh, Paramvir, and Hardeep Singh. "DynaMetrics: a runtime metric-based analysis tool for object-oriented software systems." ACM SIGSOFT Software Engineering Notes 33.6 (2008): 1-6.

[22]. Dean, Thomas R., Andrew J. Malton, and Ric Holt. "Union Schemas as a Basis for a C++ Extractor." Reverse Engineering, 2001. Proceedings. Eighth Working Conference on. IEEE, 2001.