COMPLEXITY OF RECOVERED ARTIFACTS IN REVERSE ENGINEERING

Safia Sultana[1]*, Mazhar H Malik [2], Nadim Asif [3], Maliha Rafi [4], Hafiz M Haroon [5] Noman Farooq [6]

[1,2,4,6] Department of Computer Science & IT, Institute of Southern Punjab, 9-KM, Bosan Rd, Multan

[3]Department of Computer Science, Bahria University, Lahore, Pakistan

[5]Department of Computer Science, University of Agricultural, Faisalabad Pakistan

*Email: safiasultana09@yahoo.com (Corresponding Author)

***ABSTRACT—**The software system are combination of different types of artifacts, which are required to extract at different levels of abstraction for maintenance purpose. In the new era of software development software system exist in a variety of languages, and that technically make it highly multifaceted. It is incredibly necessary to understand and extract the system documents from these complex systems before maintenance, re-engineer or reuse the software system. Software maintenance activities are require recovering the artifacts from the source code. The source code can also exist in numerous forms. When abstraction is applied to computer programming, program behaviour is emphasize and implementation details are covered up. The knowledge of a software product at different levels of abstraction certainly caused operations regarding the maintenance and reuses the existing software components. It is, therefore natural that there is secure growing interest in reverse engineering, as a capable of extracting information and documents from a software product to present in higher levels of abstraction as that of code. The abstraction as the process of ignoring certain details in order to simplify the problem and so facilitates the specification, design and implementation of a system to proceed in step-wise approach.*

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

I. INTRODUCTION

These software maintenance activities require recovering the artifacts from the source code. The source code also exists in many forms. A source code may be written in different programming languages or have different versions of same language, scripts or may have errors or incomplete and cannot be compile. The size of source code may be very large and implemented in different designs and concepts.

A. Statement of Problem

Artifacts are extracted at different level of abstraction for maintenance purpose. The extraction of these artifacts provides clues about the functionality, structural and behavior of the system. [1] This provides the description of the essential decisions that have been taken in the design of a system. Complexity of recovered artefact depend on the size of source code, degree of source code type, abstraction level and the degree of available document support to recover the artefacts for tasks at hand [2].

Software maintenance is the set of activities that mandatory to providing cost-effective support to software system. Pre-delivery activities consist on planning for post-delivery operations, supportability and rationality willpower. Post-delivery activities consist on software amendment, preparation and operating a help desk.

Reverse Engineering is the process which has the different provisions that making it advance although it is a new and rapidly developing field. Conventionally, Reverse Engineering has been defined in two steps process: (i) information extraction and (ii) abstraction. Information extraction investigation the subject system artefacts mostly the source code for gathering the row data, whereas information abstraction creates user oriented documents and views. The process of reverse engineering developed though six steps:

- Categorization of formal units into source code,
- Semantic explanation of construction of functional units and formal units,
- Clarification of association for each unit of input/output schematics units,
- Manufacturing map for all units and sequences of frequently connected linear circuits,
- Declaration and semantic report of all system applications,
- And at the end creation of scaffold of the all system units.

Above Mention steps, foremost three steps are associated with the local analysis on each module level; whereas the remaining three steps are consider as overall analysis on a system in the large systems. Software maintenance is four types of categories that are corrective, perfective and adaptive fourth one category is preventive

1) Corrective maintenance: is used to correct the software errors that are detected during system operation; it also comprise it also include the system testing with the customized the programs and upgrading the pertinent documentation both within and without program reactive alteration of a software artefact executed after delivery for correct the discovered errors.

2) Perfective maintenance: Is the modification of software system for the improvement of its functionality. The perfective maintenance is used to increase the proficient of the software system.

3) Adaptive maintenance: the adoptive maintenance is the modification of a system because of some changes to its external environment. Whenever hardware or software technology is improved then the existing software needs to change to function with the new technology.

4) Preventive maintenance: where you have to write the extra modules and functions to protect data or to evade process malfunctioning. These definitions commence the idea that software maintenance can be either scheduled or unscheduled and reactive or proactive, as shown in Table 1 depicts the correspondences that exist between these categories [3.4]

Table 1: category of software maintenance

	Unscheduled	**Scheduled**
Reactive	Preventive	Corrective Adoptive
Proactive		Perfective

B. *Software Artifact*

Software is imperceptible therefore software visualization is needed in textually [7,8,9] It is a crucial step to visualize the software. The method of information that is available to the software maintainer or programmer strappingly collides on the efficiency to the design recovery methods. Artifacts are placed at all stages of the software life cycle containing knowledge data, ontology, risks and requirements [8] The complexity of recovered artifacts has styles, patterns and design modules, subsystem, source code, test case tables and revolving aspects of software system. Static and dynamic data, high level and low level information, data and control flow, structural and other software dependencies and software attributes [10]

C. *Complexity of Software Artifacts*

Another bog dilemma in the IT world is software complexity. Most of the software perhaps has some problem in the distinct stage of life cycle. Software develops steadily by time and enhances its functionality by adding new features. Every evaluation in the source code increases the software size that is another aspect of complexity: Every new feature may be the beginning of complexity of recoered artifactshitecture degradation. Complexity risks in non functional requirements, also including maintainability, quality and productivity. In this way small application becomes library, library becomes platform, platform becomes system, system becomes large system and large system becomes ultra large system (ULS). Then we think about this wild beast of complexity that how to overcome this. It is possible to avoid software complexity, it is possible to reduce the software complexity, is it accidentally or inherent? There are so many questions arises about complexity with the evolution of software.

The software are not recognize by its purposes, behaviors, structure, algorithms and environment of domain or modification of software becomes a vital part of science for us and nobody can figure out how its code is written like this [5,6]

1) Formal Symptoms of Complexity

- Complete detail of problem domain that has the zillions of requirement.
- A complete list of requirements and specification that consist on poor designing, feature creep, over engineering.
- Highly coupling module that extremely interacted modules or subprogram
- Increasing the software size, doubling and tripling every year.
- Inseparable concerns and low cohesion consisting in the random functions,
- Abstraction has no protection or wrong,
- Proportional primitives that has strong hiercomplexity of recovered artifactshy,
- The software developer has less knowledge about domain and less capability of software development.

The software artifacts are written in different languages (natural language as well as programming languages). At each step the software artifact also describe the software abstraction level (Domain, Functional, Structural and implementation levels) [11]. Most probably it is happen that in an organization supporting artifact are not linked up. This causes a great maintenance problem. Now a day it becomes a major challenge for reverse engineering activities. As a result reverse engineering have spend a large amount of efforts on manufacturing and integrating the information system to make a links between these software artifacts. Software design documentation and source code are two major parts that used as software artifact during the process of reverse engineering [12]. The reverse engineering processes that recover the system artifacts at different levels of abstraction are depend upon the following factors:

2) Require Artifacts for Maintenance

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?

Available Source code and Documentation Type Size of the source code, Mix-mode source code of different languages and scripts or have different dialects, cannot be compiled or source code has some errors.

- Source code,
- Textual descriptions
- Existing available artifacts (i.e. complexity of recovered artifact architecture , design diagrams)
- Functional specifications & different available types of documentation of different formats.

3) Extraction of Artifacts

Completely extraction of source code is called the information about the system; this information is presented at each level of abstraction as graphs with different granularity. The maintenance analysis are base on the information at the low level is completely automated.

- Reverse Engineering activities requires to extracts the artifacts at different levels of abstraction.
- The extraction heavily depends on
- the nature of available source code
- Existing documentations
- And artifacts require for the maintenance errands at hand.
- The extraction of artifacts also depend on the require artifact specification and tracing process.

4) Presentation of Artifacts as Visual Model

- In a specific format or diagrams (i.e. UML diagram)

- At different levels to perform the maintenance activities at hand.

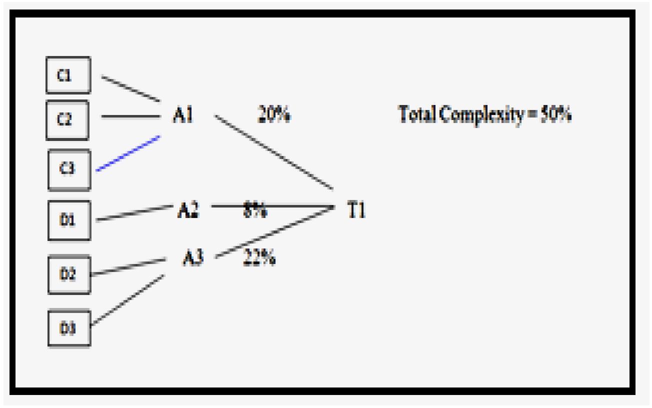

Figure 1: Measuring the complexity within a task

II MATERIALS AND METHODS

An abstraction for a software artifact concise description is suppressed the detail; that is insignificant to developer and the important information highlight (Asif et al.). For example in high level programming the abstraction allow a programmer to construct the algorithm without containing the detail about hardware register allocation. The Software artifact has normally a number of layers. When maintenance problems occur, the levels of abstraction layer is applied to recover the software artifacts. The Software artifact has normally a number of layers. When maintenance problems occur, the levels of abstraction layer is applied to recover the software artifacts. These layers are:

- Domain Abstraction Layer
- Functional Abstraction Layer
- Structural Abstraction Layer
- Implementation Abstraction Layer

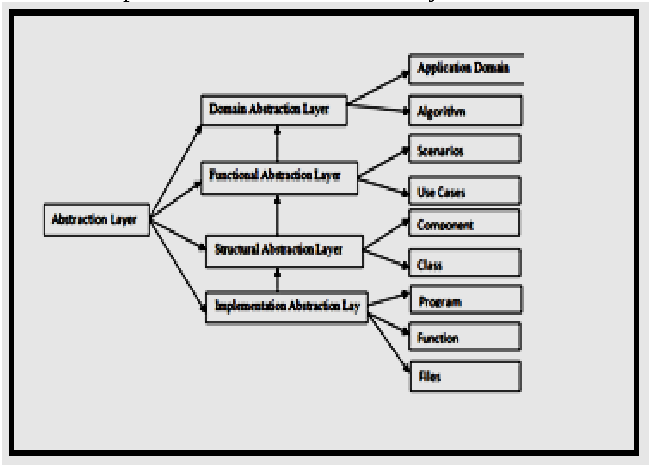

Figure 2: Abstraction Levels

At the every level of abstraction layer, it has the different type of artifacts and every abstraction layer has the specific values. When we calculate the complexity at every abstraction level it generates a different value.

1) Domain Abstraction Layer

Domain Abstraction further abstracts the functions by replacing its algorithmic nature with concepts and specific to the application domain. Application Domain in abstraction layer is a set of interrelated software system that contributes to common design features. Domain in this context has been defines as:

- An area of application
- An area of business
- An area of software business
- An area of software intensive application
- Areas of application which have the similar software systems have been built**.**

2) Functional Abstraction Layer

Functional abstraction level is a further higher abstraction level, it usually achieve by further abstraction of components or sub-components (programs or modules or class) to reveal the relations and logic, which perform certain tasks e.g. use cases and scenarios.

3) Structural Abstraction Layer

Structural abstraction level is a further abstraction of system components (program or modules) to extract the program structures, how the components are related and control to each other. The artifacts at this level data flow diagram, processes and complexity of recovered artifacts.

4) Implementation Abstraction Layer

Implementation abstraction is a lowest level of abstraction and at this level the abstraction of the knowledge of the language in which the system is written, the syntax and semantics of language and the complexity of recovered artifact of system components (program or module tree) rather than data structures and algorithms is abstracted. Artifacts at this level are program, function and files.

III RESULTS AND DISCUSSIONS

The software evolves to meet the requirement of new worlds, an obsolete functionality is removed and the new module is added, so the design gradually diverges from its original design. The alteration initiates the system's evolution due to variety of reasons, adding the new feature in the system on the user request, Adding the new hardware and software technologies and business decision to improve the source code. Software evolution and maintenance depends upon the several factors including the existing documentation of system design. In some case, the original system design has not any type of existing documentation; as a result the decision at implementation level makes problems.

The source code does not contain the much information about the original design information, which must be reconstructed from available sources. This makes the system complex. The artifact recovery complexity depend on the size of source code, degree of source code type, abstraction level and the degree of available document support to recover the artifacts for task at hand.

The table is also show by graph that how the values are varies at different levels.

A) Complexity at Level 2

Task 2, task 6 and task 29 have the same complexity level. Here SSC for these tasks are in small form that size of source code consists on only few lines of code. SCT for these tasks are in normal form, it mean source code exist for these tasks are in a single language. AL for task 4 and 6 are in implementation level. Source code for these two task are in the form of files function definition and call procedures. The task 29 has the structural level, its mean source code for here is in components or in complexity of recovered artifacts. The DTS for task 4 and 6 is in minor form that documentation at this level is only system or component details. So the artifact recovery complexity is 2 that easily recovered because difference only occur in task 29 between AL and DTS which is structural level and DTS is in medium level. Although we have artifact at AL in component or complexity of recovered artifactshitecture but we know about the requirements, design and implementation details that support to understand the existing artifacts. So we can easily recovered the artifact and calculate that how much complicated.

B) *Complexity of Recovered Artifacts at Level 3*

The artifact recovery complexity of tasks 1, 3,8,11,13,16,21 is at level 3. The SSC value of task 1and 3 are 1 and for task 8, 11, 13, 16, 21 are 2. The values shows that the task 1 and 3 are in small size of code that consist on few lines while other tasks are medium type of code that consist of 1000 lines of code. The second dependency of COMPLEXITY OF RECOERED ARTIFACTS is SCT. All source codes type are in normal form that source code exist in a single language. Third dependency of complexity is AL. the abstraction level for all tasks is also same and exists at the implementation level. The fourth dependency is DTS. The tasks 1and 3 has no documentation and tasks 8,11,13,16 has consist only system/component details. The task 21 has medium type of DTS; that consist on some requirements, design and implementation details. The complexity at level is 3.

C) *Complexity at Level 4*

The SSC for task 2 is small and DTS for task 2 is not available but there are errors in the source code type at the abstraction level of implementation. So recovery of artifacts is not easy with error code. The other task 12, 14 and 17 are also place in the same complexity value 4. Here we have the SSC medium type but all source code type consist on the errors and there abstraction level is lowest level. Although we have minor type of DTS but recovery of artifact is not easy. For recovering artifacts first we must remove the errors from code then we can overcome its complexity.

D) *Complexity at level 5*

Here we have 5 tasks for measure the artifact recovery complexity. The task 10 has SSC value 2, its mean size of source code for this is medium and source code type is incomplete here. Abstraction level is in the form of files, function and procedure calls. So with incomplete source code it is complicated to recover. The value for task 15 is same as task 10. Now discuss about task 23 and 24 which have the same dependency values. SSC is small form and source code type is normal here. The abstraction level is at the highest level where the high level entities describe the system. Although we have minor documentation type support but it's difficult to recover the artifact. The task 30 has very large size of source code that is in normal form but here we have use cases and scenarios at the abstraction level. The recovery of functional layer is not so easy with very large size of source code. Although we have medium type of documentation support for task 30 but it's difficult to recover.

E) *Complexity at Level 6*

Here we have 5 tasks. The task 5 has few lines of source code but SCT is in Mix-mode here that consist on multiple languages. The abstraction level is implementation type with minor documentation support. The tasks 9 and 18 are has the same dependency values. SSC is medium type SCT is in dialect form and abstraction level is lowest with minor type of documentation support. The task 20 is in large size of source code which is written in the different version of same language. DTS for task 20 is medium type. Now about task 22; which consist on very large size of source code with dialect form? Abstraction level is lowest but has the high documentation support that help to recover the artifact.

F) *Complexity at Level 7*

Here we have 4 tasks for discuss, the task 7 has medium type of SSC with value 2 and SCT is in Mix-mode. Abstraction level is lowest with minor type of DTS. So recovery is difficult due to SCT which consist on multiple languages. The task 19 has large size of source code in dialect form. Abstraction level is lowest but has the minor type of DTS. The task 27 consists on very large with dialects form. Abstraction level is implementation with medium type of documentation support. Now about task 28 which is large size of source code in multiple languages and this source code is in form of use cases and scenarios but DTS for this task is highest.

G) *Complexity at Level 8*

Now we discuss about task 28 that it's Complexity of recovered artifacts is up to 8. Task 28 is medium type of source code which is written in multiple languages. The abstraction level is in functional form and DTS is in some requirements, design and implementation details exits for support.

IV CONCLUSION

The software maintenance is distinct as the concert of all activities required to keep a software system operational and approachable after it is accepted and placed into production. The software maintenance activities are classified into four major types, perfective, adoptive, corrective and preventive. This classification based on modifying program to generate new outputs, to change executing logic, to integrate new features, to improve the existing features, to correct errors in the existing code when they are detected during the meting out of the system, to optimize the code, and to adapt the software to a different hardware/software environment.

REFERENCES

[1]. S. Sultana, N. Asif, M. H. Malik, and M. Rafi, “IMPLEMENTATION OF SOFTWARE ARTIFACT RECOVERY COMPLEXITY TOOL,” SCIENCE INTERNATIONAL-LAHORE, no. 06, pp. 6033–6038, Dec. 2015.

[2]. Asif, N., Dixon, M., Finlay, J., & Coxhead, G. (2002, June). Recover the design artifacts. In proceedings of International Conference of Information and

Knowledge Engineering (IKE02) (pp. 656-662). Asif, Nadim, et al. "Recover the design artifacts." proceedings of International Conference of Information and Knowledge Engineering (IKE02). 2002.

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

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

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

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

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

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

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

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

[11]. 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.

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

[13]. 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.

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

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

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

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

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

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

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

[21]. 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.

[22]. 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.

[23]. 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.