Software Architecture Understandability of Open Source Applications

Mohammed Akour
Yarmouk University
Irbid, Jordan
Mohammed.akour@yu.edu.jo

Samah Aldiabat
Yarmouk University
Faculty of Information Technology
Irbid , Jordan
sdiabat1@yahoo.com

Hiba Alsghaier
Yarmouk University
Faculty of Information Technology
Irbid , Jordan
hibaalsghaier@yahoo.com

Khalid Alkhateeb
Yarmouk University
Faculty of Information Technology
Irbid , Jordan
kikhateb@gmail.com

Mamdouh Alenezi
Prince Sultan University
KSA
malenez@psu.edu.sa

***Abstract*—Software Architecture plays the most important phase in designing and developing software systems, which makes the understandability a very important factor in Software Architecture for managing and improving the system and also very important in the reusability and maintainability processes. In our research we tried to answer the following two questions: the first one is how we can measure the understandability of the Software Architecture, and the second question is how understandability evolves over several versions. We found metrics that relate to the understandability of the Software Architecture also we used available tools to measure these metrics on open source software architecture. We will discuss in this paper tools and techniques that can be used to ease the understanding of Software Architecture.**

Keywords—software architecture; understandability; architecture metrics; open source architecture; evolution

I. INTRODUCTION

Open Source Software (OSS) is a free source code or software that is generally public, available for use and modification [34]. OSS is created by the efforts of programmers, who collaboratively create shared community. It is a collaborative experience of developers that develop, test and improve these systems in both academia and industry. People prefer the open source software since it can be found free or with a low cost, wide spread and bugs get fixed immediately. It gets closest to what the users want, they can use part of the OSS and modify on it as they want to match their demands.

Software architecture plays the most important phase in designing and developing a software system, because the maintenance takes the lion share in the software development, a well-known survey shows that the maintenance operation cost about 60-77% of the total cost of software system development, and the maintenance engineers spend around 50% of their time trying to understand the code in order to maintain it. So the software architecture is very important to be understood, plus that the maintenance team will not produce new bugs and errors when they are trying to correct existing errors [34].
Understanding the software architecture might be difficult since dependency is very complex in large systems and there are no available tools to visualize the architectural design in large and complex system development [1]. Understandability is very important in the process of managing and improving the software and useful factor in the reusability and maintainability processes.

The continuing change in the customer's requirements which demand maintaining and modifying the software, which requires the developers to understand the software before modifying it. This leads the programmer to read and understand the software and its

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

documentation. Misunderstanding the software and its quality factors cause failure in delivering this change without affecting the quality of the product. One of the factors that cause misunderstanding of the software is delivering the software on time without checking the quality and testing it, that may cause many problems like, high cost of maintenance, low quality, and insufficient design [2].

One of the techniques to measure the understandability of a software structure is to decompose the system into substructures and then decompose these substructures to another substructure or packages, then measure the coupling and cohesion between substructures, where the cohesion is the strength of the functional relatedness within the same substructure and the coupling is the functional relatedness between other substructures [1].

II. RESEARCH STEPS

A. Research Questions*(Specific Objectives)*

In our research, we are trying to answer the following two questions:

1. How can we measure the software architecture understandability?
2. How understandability evolves over several versions of software?

B. Research Purpose

We will try to find an appropriate method and metrics to measure the understandability of open source software architecture and how the understandability will affect the change over several versions. Understandability of software architecture is an essential attribute in the software development. For this reason we are interested to discover the most important factors that affect in the measurement of the understandability using the appropriate tools to produce high level quality software.

III. BACKGROUND

Software architecture provides a schema for system designing and implementation. The schema focuses on system goodness, such as performance, security, modifiability, usability, and understandability. Software systems should be designed with regard of these main scopes (user, system, and the system goals). For each scope, metrics should be considered to measure the success in each of these scopes. Many concerns should be considered in building the software architecture, like the way users will use the system, requirements of the system such as security, flexibility and maintainability over time and the architectural direction that may impact the system after the system is developed [3].
Software understandability plays a requisite role in measuring the reliability of the software, which means that without understanding the design of the system, the system structure will become progressively worse when system modification happens, so the software understandability needs to be managed [4].

Since the higher-level software architectures are repeatedly utilized to catch the final view of the system, then Understandability of software architectures is very important to designers and developers to provide them with fine details about the whole software architecture [35]. Furthermore, it will help them in all stages of software development life-cycle and cover the hiatus between higher-level and low-level views. In order to estimate the understandability of software architecture, many attribute should be considered such as: modularization, size, coupling, complexity and cohesion. There are 3 different approaches has been developed to help in understanding software architecture:

- Internal structure based metrics: to measure the relationship between package, component and modules such as (classes, interfaces and subsystems that are closely related to the higher-level).
- Graph based metrics: it depends on representing the relationship between system components as nodes and edges.

Finding the values of metrics that just measure architecture component individually is not enough, the overall architecture should be measured too. Somehow there are 2 types of metrics:

1. Metrics related to the component of the software: such as Component-to-component, components, Package, Modules and Graph-node.

2. Metrics related to the attributes of the software such as: Size Metrics, Coupling Metrics, Cohesion Metrics, Complexity Metrics and Quality Metrics [5].

IV. RELATED WORK

Stevanetic and Uwe [5] reported an efficient mapping study on software metrics that measure the understandability idea of the more elevated architectural structures (i.e., level above classes) regarding to the relations in the system implementation. Reconstruction the architecture of the Software is another research direction closed to work [3]. It is dealing with these two

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

problems: (1) software architecture is not explicitly represented in source code; (2) we evolve the successful software applications over time, so their architecture inevitably changes. So we reconstructing the architecture and checking whether it is still valid.

Software architecture style (SAS's) model was developed to design software in a way that the components and the relationship between them are well described, so that many methods were developed to estimate the software architecture such as measurement-based evaluation, which focuses on using metrics to measure the internal and external attributes and the relationship between them. Many research studies confirm that complexity, cohesion and coupling between components have a huge impact on software understandability and maintainability [6].

Fabio Palomba [7] performed an Extract Package Refactoring approach by using ARIES tool that uphold developers in the correspondence of packages that need to be re-arranged. His experiment comprises 16 Master students who resolved refactoring process proposed by the approach towards realizing if the packaged that were extracted have a higher quality than the original packages, and weather the extracted packages are significant. The experiment pointed out that the decomposed packages have better cohesion without a deterioration of coupling and the re-modularization proposed by the tool are meaningful from a functional point of view.

Tianlin Zhou [8] proposed a new cohesion metric for package called SCC, which is supposed to determine whether a package is cohesive or not. On the hypotheses that two components are very attached if they have similar contexts. SCC uses the context of two components to conclude if they are similar or not. And he also contrasts the SCC with RC and EEC by using open source projects. Both of them contain more than 780 components, and these components are gathered into more than 80 nonempty packages. As a result, most of these packages are cohesive.

Gupta [9] introduced another approach to measure the cohesion of packages which is dependent on the relationship between elements of the package, like sub-packages, classes, interfaces the package cohesion refers to the relationship between all sorted and individual package elements divided by most possible number of relationship among them. The experiment, based on choosing 25 packages chosen from six open source software projects, confirmed that the reusability and the submitted package level cohesion measurement shown in his study is significant in order to design a software with high quality.

Hani Abdeen [10] provided a set of metrics that estimate some modularity principles for packages in large object-oriented software. These metrics are defined with regard to object-oriented dependencies as a reason of inheritance. He proved the developed metrics theoretically through a study of the mathematical properties of each metric. Two of the developed metrics deal with package coupling and the others with package cohesion, also his metrics measure *"to which extent a given OO software modularization is well-organized"*, and he showed that all his metrics achieve the mathematical properties that cohesion and coupling metrics should follow. However, he didn't conduct an experimental study the metrics with real large software systems to approve their usefulness with independent Software supervisors.

Mojtaba Shahin [11] produced a survey on tools that uphold the architecture design of software and the verification of how compend tool can be used as a general tool to envision the software architecture. And also he produced how the envisaging by compend can progress of understandability and uphold the connection of software architecture procedure.

Srdjan Stevanetic [12] made a questionnaire for a 7 components of a Soomla android store system distributed into 42 master students. The students were asked if they understand these components and then answer the questions. The students have different level of programming experience; the first group includes dependent variables: time required to study the 7 components the percentage of the correct answers. The second group includes independent variables related to the following metrics: number of classes (NC), number of incoming dependencies (NID), number of outgoing dependencies (NOD), and number of internal dependencies (NIntD). Regarding the values of the two groups of variables, they make some analysis to find the relationships between the values and the understandability of Smoola according to the students. The results show statistically significant relation between 3 metrics (NC, NID, and NIntD).

Walt Scacchi [13] developed an approach for open source software architecture to understand the relationship between components and how these component are integrated to each other and also he elaborated on the ways in which a collected component in the architecture can be improved over time, and how a software architecture promotion can change the old versions of the software architecture.

Taimur Khan [14] provided an overall survey in the scope of software architecture and his study offer that the software architecture conception developed rapidly in the last decade by improving many tools to understand, estimate, and develop

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

software architecture and serving administrators to observe the architecture conception and its various issues. He concluded that due to the problem of inability to try all tools and using them from stakeholders, investigator should work sincerely with experts, to meet the requirements, and to attitude a thorough assessment to get better quality, minimize time and cost influence.

Ananya Kanjilal [15] offered a new graph metric in order to constitute the unified modeling language component diagram and using it to measure many levels that are related to the software architecture. The CAG metrics will qualify developers to focus on the unused components, highly coupling and complexity.

Abdul Azim Abdul Ghani [16] surveyed and reported few findings in complexity metrics from software engineering researchers and gather opinions from them for some metrics and which of them are extent analogous that can be defined for business process models and they also proposed a Goal-Question-Metric (GQM) framework for measuring the understandability and maintainability of BPMs which it can be used when measuring a software project as a whole.

GUO Wei [17] used the graph theory to estimate the properties of software Architecture based on the components assembly. He divided the component – based the architecture on components and connectors. The components responsible for functions and connectors responsible for control and communications for the system. His algorithm for optimizing the component based architecture based on the number of components (N), number of connections attached to the component (Δi), the distance between components i.e. the shortest path between a pair of component (u), and the Diameter (D) which is the largest number of connectors to transmit a message between any two components (maximum inter-component distance). According to the metrics, estimating the best structures of the component assembly can be used to explore theoretical optimum for the architecture.

V. SOFTWARE ARCHTECTURE COMPONENT

Software architecture consists of a number of components; these components should communicate with each other to produce the service that the system aims to produce. In our paper we concentrate and discuss two of these components.

A. The module as a compnent of software architecture :

In large-scale software systems, developers and managers should pick out and emphasize the understandability according to the complexity and size of them. The module in the large-scale object oriented system is referenced as a package which contains large number of classes. These classes must be situated in a group to achieve a specific target. The module is considered as a black box, since it hides the information.

The modularization works to make the system easier to understand, which leads to minimize the complexity and make the system design in high abstract level. Therefore, the large-scale software modularization must be managed in the understandability view to avoid any disadvantage of the system modification over time.

The coupling and cohesion metrics can estimate the quality of the software and the design properties. However, they are inadequate to manage the understandability of the software in the package level. To manage it, the quality model should have the power to manage the dependencies between the modules and its constitution. The quality model that manages the dependencies is defined by decomposing the quality properties of the high level into perceptible quality attributes of the components of the product such as implementation and design. Finally, the connection between the quality properties of the high level component will be established. Following are the six design properties that have an effect on the understandability of the modular system 1) design, 2) complexity, 3) design size, 4) encapsulation, 5) cohesion, 6) coupling and modular abstraction. The hierarchy of the software design module supports the relationships between parent and child modules. The aim of decomposing the package is recognizing how to make packages more manageable and understandable [4].

B. The Package as a compnent of software architecture:

Gupta et al [9] defined the package as a set of elements (classes, interfaces or packages) and relations between elements. The presence of a package within a package leads to the hierarchical organization of the package." While the Sub package is a package that locates inside another package and also it denotes to high hierarchy level. Disjoint packages: refer to the package in the system that doesn't share classes or interfaces between each other, for example class A belongs just to one package, the Empty package is a package that does not have any element so that it does not share any relationship with other packages. The existence of various types of element in a package such as classes, interfaces or sub packages leads to various types of relation within the

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

package, for example, Class to Class relation, Package to Package relation, sub package to Class relation, class to Sub package relation.

In OO languages like java, C++ and Smalltalk, the structure of the package enables people to arrange their programs as subsystems. The good modularized system can replace parts of it without any effect on the system. When organizing the classes into collaborating and recognized packages make the understanding, testing, maintaining and evolution the software easy. With system modularization, the code will decay because of the modification over time (removing, adding of classes and relationships between inter-class dependencies). Progressively, modularization causes quality loose because of modification where some of the classes may be at the wrong package and some of the packages require restructuring. To enhance the software modularization quality, the packages and relationships need to be organized and assessed. There are some modularity principles that related to the reusability, changeability and packages encapsulation [10].

Understanding the structure of the packages in software architecture component is very important in large-scale software systems. So that cohesion metrics are used to measure the quality of packages, it is a significant part in the organization of the code and structure. So it is simple to understand the architecture by package structure rather than reading the code. Logical package structure reforms the maintainability, understandability, extensibility and reusability of software.

As shown in Zhou et al work [8], a p*ackage cohesion* means “reflecting how tightly-related the internal components of a package are to contribute to a single task" so that the complexity of the package is low if its cohesion is high. also the components in a cohesive package are similar according to changes. This makes the development of software much easier, which means higher cohesion of a package is better if the package is understood. As well as the volume of the package is too large, then the understandability is more complex, also when there is a high dependency between packages then the understandability will be more complicated. Furthermore, if the package is not balanced then the understandability will be more sophisticated [18].

VI. Software architecture evolution

Software architecture evolution means how to make changes on, or add new features to the architecture; this process depends on many factors. One of them is the good evaluation of the architecture in order to reach into a protective evolution method and in order to reduce the consuming time in evolution process. Furthermore, the architecture quality of the attribute plays a significance role in evolution, when the architecture begins with understandable design then adding or making changes will be easier.

Software architecture evolution is a difficult technique due to the hardness of picking out a good quality of requirements and making change on it. Moreover, the hardness of making changes on the high-dependency architecture. [19]
Satisfying customer’s requirements might demand keep maintaining and modifying the software, which is done by programmers, where sometimes these programmers have never programmed that software. This could lead the programmer to read and understand the software and its documentation.
Another factor that may affect the understandability of the software is the gap between the years of programming the software and modification it.
Understandability is very important in the process of managing and improving the software, and also it is a very useful factor in the reusability and maintainability processes.
One of the factors that may affects the understandability is the low of cohesion of the class, so the highly cohesion and loosely coupled class (“good software design with manageable complexity”) can increase the understandability.
Misunderstanding the software architecture causes the false interpretations that lead to the ambiguity and the false software development results.
Some of the metrics that affects the understandability (total num of classes, total num of attributes, total num of methods, total num of associations, total num of aggregation relationships within a class diagram, total num of dependency relationships, total num of generalization relationships within a class diagram, total num of aggregation hierarchies in a class diagram, total num of generalization hierarchies in a class diagram, maximum between the total num of aggregation relationships within a class diagram values obtained for each class of the class diagram, maximum between the DIT values obtained for each class of class diagram).[20]

The best way to reduce the cost and time in developing the software is to reuse it, and for reusing the software the targeted software must be well understood to ensure that the components of that software in a high quality.
As shown in the Figure (1) below, to enhance the reusing of the software, the programmer should understand the old version of the software and evolves it to the next version without faults and errors that may cause creating the software again with the new demands.
The understandability of software evolving is a complex and not easy process since it depends on humans. [21]

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

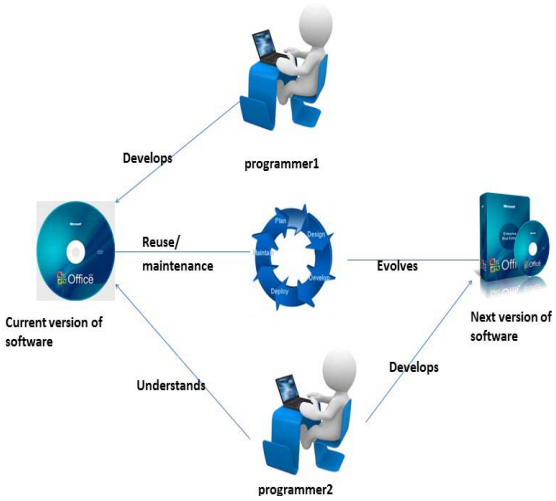

Figure (1): software architecture evolution

VII. Software Architecture Visualization

As mentioned previously that Understanding the software architecture is the most important step for building and maintaining software systems, because it is hard to comprehend the software architecture. Visualizing software architecture has been one of the most important techniques to understand and make a visual mapping for that architecture.
Imran Ghani papered al [22] derive and construct a new method called "Tiny-Notational Approach", to make a good visualization of software architecture by determining a particular component or the activities by the other components. They present a scenario of web based client server with its architecture. The usefulness of the "Tiny-Notational " approach is to improve understandability and mutual communication of software architecture visualization between stakeholders.
"Tiny-Notation" can be applied in any layer of the system architecture i.e. Application Layer, Presentation Layer, Business Layer, Resource Access Layer, Resources Layer.
"Tine-Notations" can be applied to most of Architecture Description Languages (ADLs). The proposed Tiny-Notational approach can be used as a tool to allow architects to add understandability into their architectures. [22]

There are too many existing approaches for software architecture visualization. One of them is DiffArchViz for visualizing the software architecture of network-based large-scale systems, the DiffArchViz tool Visualize Correspondence between Multiple Representations of Software Architecture. [23]
The second one is Axivion Bauhaus Suite which supports couple of programming languages and platforms. This tool allows developers to browse their software at a high-level and also to show whether some functions are necessary [31].
The third one is Structure 101 which is a Java based tool that makes software structure easy to understand, define, communicate, control and keep it simple [32].
The fourth one is Software Architecture Visualization and Evaluation (SAVE) [24]. The SAVE tool creates an actual architecture from source code and compares it with planned architecture and identifying architectural violations [25], and the last one is Ecospecies Visualization Tool which explores Software Architecture in 3D which is a reverse tool that provides an architectural level visualization of software systems in a virtual environment [26].

VIII. Table of Metrics and Tools

In our research, we summarized a table that contains metrics, their definitions, the tools used to measure them and the reference for both the metric and the tool. Table 1 shows the studied metrics in this study.

Metric name	Metric definition	Reference
Component-to-Component	the number of connecters between components	[1]
Package	the number of classes in the package and the number of other packages that are related to the classes in a specific package	[1]
Modules		[1]
CAG metric	measure usability, understandability functionality, and complexity of a component or an interface	[15]
graph node:	this metrics perform the software system in summarized way	[1]

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

Size Metrics	it is affined to the number of components in the whole system structure	[1]
Coupling Metrics:	It is related to the indirect and direct relationship between design components, like the number of paths, interfaces and links between components. Furthermore there are 2 types of coupling (import and export).	[1], [4],[7]
Complexity Metrics:	it is related to the connectivity degree between elements in one unit and the connectively degree between units in the whole design	[1]
Stability Metrics:	It measure if it is simple and possible to make modification without impressing the other element	[1]
Cohesion Metrics :	there are 2 types of cohesion :internal and external, both of these types are related to the relationships between elements in one unit, and the elements in other unit in the same design	[1], [4] , [7], [33]
Quality Metric:	measuring the quality of component according to its attribute	[12] , [27], [29]
Number of Classes (NC):	The total number of classes inside a component.	[12]
Number of Incoming Dependencies (NID):	The NID defined as a total number of dependencies between classes outside a component and classes inside a component that has been used by those outside classes.	[12]
Number of Outgoing Dependencies (NOD):	The NOD defined as a total number of dependencies between classes inside a component and classes outside of a component that has been used by those inside classes.	[12]
Number of Internal Dependencies (NIntD):	The NIntD defined as a total number of dependencies between the classes within a component.	[12]
Internal structure based metrics:	to measure the relationship between package, component and modules such as (classes, interfaces and subsystems that are closely related to the higher-level	[5]
Graph based metrics:	representing the relationship between system components as nodes and edges.	[5] , [11] , [30]
Autonomy Ratio (AR)	to find a way to potential understandability of some piece of software by characterize the functional independence of a substructure whatever its level in the system	[1]
SCC metric	determine whether a package is cohesive or not	[8]
Afferent Couplings	"the number of other packages that depend upon classes within the package. It measures the incoming dependencies (fan-in)".	[18]
Efferent Couplings	"the number of other packages that the classes in the package depend upon. It measures the outgoing dependencies (fan-out)".	[18]
Instability (I):	s is an indicator of the package's resilience to change. The range for this metric is zero to one, with zero indicating a completely stable package and one indicating a completely unstable package".	[18]
Distance (D):	This metric is an indicator of the package's balance between abstractness and stability".	[18]
Package cohesion	"reflecting how tightly-related the internal components of a package are to contribute to a single task"	[8], [7]
CC	Concrete Class Count	[28]
AC	Abstract Class (and Interface) Count	[28]
Ca	Afferent Couplings (Ca)	[28]
Ce	Efferent Couplings (Ce)	[28]
A	Abstractness (0-1)	[28]
I	Instability (0-1)	[28]
D	Distance from the Main Sequence (0-1)	[28]
V	Volatility (0-1)	[28]

Table 1: Studied Metrics

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

IX. The Experimental Test

In this research paper experiment tests are conducting to provide an evidence that understandability is very important to the software architecture. We used two tools to measure many metrics that are related to the software architecture. The first tool we used is SDmetric tool (reference) . At the beginning we conduct the experiment on an open source code and convert it into xmi file by using ALTOVA tool (reference). Then the XMI file(which is a software architecture file) is imported inside the Sdmetric tool and the results are shown in the Appendix , Figures (2…12).The results clarify the metrics measurement on the high level software architecture such as package, component and interface.

The second tool we used is the Understand tool (reference), where we convert a java open source code into XML diagram and import it into the Understand tool, the results are shown in The Appendix Figures (13….17). The results clarify the dependency between components in software architecture.

As a result, from the both used tools we noticed that the software architecture component with high coupling decrease the understandability because developers need to know all the services the element relies on and how to use them, versus software architecture with low cohesion considered to be difficult for refactoring, reusing and understanding, also the software architecture with high dependency leads to strictness in understandability and evolution

Furthermore, the experiments are conducted on brought another four source codes, and convert them into architecture, and repeat the above experiment test. The results are given to 20 students and 10 developers. We asked them to identify the most influential metrics to the understandability. Most of them concerned about coupling and cohesion, where 90% of them ensure that when there is a high dependency between classes in the architecture it will be difficult to understand the architecture and also hard to maintain and reuse it.

X. The Conclusion

The purpose of our paper is to answer these two questions

1. How can we measure the software architecture understandability?
2. How understandability is doing over several versions of software?

we tried hardly to answer the above 2 questions by collecting more than 60 papers that are related to the understandability of software architecture - included 33 references in our research and developed an experimental test based on 2 tools: Sdmetric and understand tool. Our experimental test and the survey that includes 10 developers and 20 students opinions about the understandability of software architecture and the most important metrics related to it . we conclude that understandability of software architecture plays a significant role in all the phases of software development life cycle. Also software architecture understandability can be measured depending on measuring the metrics of software architecture. For example: the metrics that measure the dependency between software architecture components show that the high dependency will leads to low understandability and hardness in evolution . Furthermore understandability is very important in the process of managing and improving the software and useful factor in the reusability and maintainability processes. The increasing of Software architecture understandability will decrease the difficulty of software architecture evolution.

References

[1] Philippe Dugerdil and Mihnea Niculescu, “Visualizing Software Structure Understandability”, published in 23rd Australasian software engineering conference (ASWEC) 2014.

[2] Mohd Nazir, Raees A. Khan and Khurram Mustafa, “A Metric Based Model for understandability Quantification”, JOURNAL OF COMPUTING, VOLUME 2, ISSUE 4, APRIL 2010, ISSN 2151-9617.

[3] “Varinder Pabbi Asst. Prof. Ramgarhia Institute of Engineering & Technology, Phagwara,India :" Software Architecture and its Various Tools",International Journal for Science and Emerging Technologies with Latest Trends” 3(1): 36-45 (2012) ~ 36 ~

[4] Hwa, Jimin, Sukhee Lee, and Yong Rae Kwon. "Hierarchical understandability assessment model for large-scale OO system." Software Engineering Conference, 2009. APSEC'09. Asia-Pacific. IEEE, 2009.

[5] Stevanetic, Srdjan, and Uwe Zdun. "Software metrics for measuring the understandability of architectural structures: a systematic mapping study." Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering. ACM, 2015.

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

[6] ShahMohammadi, Gholamreza. "Evaluation of the Software Architecture Styles from Maintainability Viewpoint." int. Journal of computer science & information technology 6.1 (2014).

[7] Palomba, Fabio, et al. "Extract package refactoring in ARIES." Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on. Vol. 2. IEEE, 2015.

[8] Zhou, Tianlin, et al. "Measuring package cohesion based on context." *Semantic Computing and Systems, 2008. WSCS'08. IEEE International Workshop on*. IEEE, 2008.

[9] Gupta, Varun, and Jitender Kumar Chhabra. "Package level cohesion measurement in object-oriented software." *Journal of the Brazilian Computer Society* 18.3 (2012): 251-266.

[10] Abdeen, Hani, Stéphane Ducasse, and Houari Sahraoui. "Modularization metrics: Assessing package organization in legacy large object-oriented software." *Reverse Engineering (WCRE), 2011 18th Working Conference on*. IEEE, 2011.

[11] Shahin, Mojtaba, Peng Liang, and Mohammad Reza Khayyambashi. "Improving understandability of architecture design through visualization of architectural design decision." *Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge*. ACM, 2010.

[12] Stevanetic, Srdjan, and Uwe Zdun. "Exploring the relationships between the understandability of components in architectural component models and component level metrics." *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2014.

[13] Scacchi, Walt, and Thomas A. Alspaugh. "Understanding the role of licenses and evolution in open architecture software ecosystems." *Journal of Systems and Software* 85.7 (2012): 1479-1494.

[14] Khan, Taimur, et al. "Visualization and evolution of software architectures." *OASIcs-OpenAccess Series in Informatics*. Vol. 27. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.

[15] Kanjilal, Ananya, Sabnam Sengupta, and Swapan Bhattacharya. "CAG: A Component Architecture Graph." *TENCON 2008-2008 IEEE Region 10 Conference*. IEEE, 2008.

[16] Azim, Abdul, et al. "Complexity metrics for measuring the understandability and maintainability of business process models using goal-question-metric." (2008).

[17] Wei, Guo, Xiong Zhong-Wei, and Xu Ren-Zuo. "Metrics of graph abstraction for component-based software architecture." *Computer Science and Information Engineering, 2009 WRI World Congress on*. Vol. 7. IEEE, 2009.

[18] Elish, Mahmoud O. "Exploring the relationships between design metrics and package understandability: A case study." *Program Comprehension (ICPC), 2010 IEEE 18th International Conference on*. IEEE, 2010.

[19] Breivold, Hongyu Pei, Ivica Crnkovic, and Magnus Larsson. "A systematic review of software architecture evolution research." *Information and Software Technology* 54.1 (2012): 16-40.

[20] Nazir, Mohd, Raees A. Khan, and Khurram Mustafa. "A metrics based model for understandability quantification." *arXiv preprint arXiv:1004.4463* (2010).

[21] Srinivasulu, D. "Evaluation of Software Understandability Using Software Metrics." (2012).

[22] Ghani, Imran, Bhaskar Prasad Rimal, and Seung Ryul Jeong. "Tiny-Notational Approach for Software Architecture Visualization." *International Journal of Computer Applications* 43.4 (2012): 38-42.

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

[23] Sawant, Amit P. "Diffarchviz: A tool to visualize correspondence between multiple representations of a software architecture." *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on*. IEEE, 2007.

[24] Stratton, William C., et al. "The SAVE tool and process applied to ground software development at JHU/APL: an experience report on technology infusion." *Software Engineering Workshop, 2007. SEW 2007. 31st IEEE*. IEEE, 2007.

[25] Duszynski, Slawomir, Jens Knodel, and Mikael Lindvall. "SAVE: Software architecture visualization and evaluation." *Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on*. IEEE, 2009.

[26] Alam, Sazzadul, and Philippe Dugerdil. "EvoSpaces: 3D Visualization of Software Architecture." *SEKE*. Vol. 7. 2007.

[27] Nenonen, Lilli, et al. "Measuring object-oriented software architectures from UML diagrams." *Proc. 4th Intl. ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering*. 2000.

[28] Cazzola, Walter, Alessandro Marchetto, and Fondazione Bruno Kessler. "AOP-HiddenMetrics: Separation, Extensibility and Adaptability in SW Measurement." *Journal of Object Technology* 7.2 (2008): 53-68.

[29] Telea, Alexandru, Lucian Voinea, and Hans Sassenburg. "Visual tools for software architecture understanding: A stakeholder perspective." *IEEE software* 27.6 (2010): 46.

[30] Jansen, Anton, et al. "Tool support for architectural decisions." *Software Architecture, 2007. WICSA'07. The Working IEEE/IFIP Conference on*. Ieee, 2007.

[31] Axivion Bauhaus Suite, http://www.software-acumen.com accessed on 5th December 2011

[32] Structure 101. Available online from: http://www.headwaysoftware.com/ accessed on 4th August 2011.

[33] Lincke, Rüdiger, Jonas Lundberg, and Welf Löwe. "Comparing software metrics tools." Proceedings of the 2008 international symposium on Software testing and analysis. ACM, 2008.

[34] M. Alenezi and F. Khellah, “Evolution impact on architecture stability in open-source projects,” International Journal of Cloud Applications and Computing (IJCAC), vol. 5, no. 4, pp. 24–35, 2015.

[35] Mamdouh Alenezi, “Software Architecture Quality Measurement Stability and Understandability” International Journal of Advanced Computer Science and Applications(IJACSA), 7(7), 2016.

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

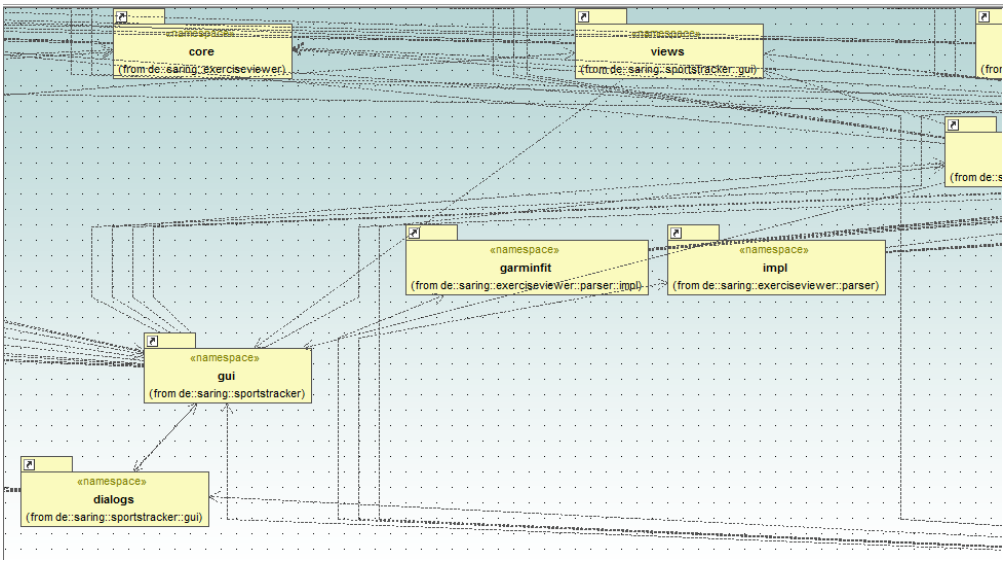

Figure 1: Altova tool to convert open source code into an xmi diagram

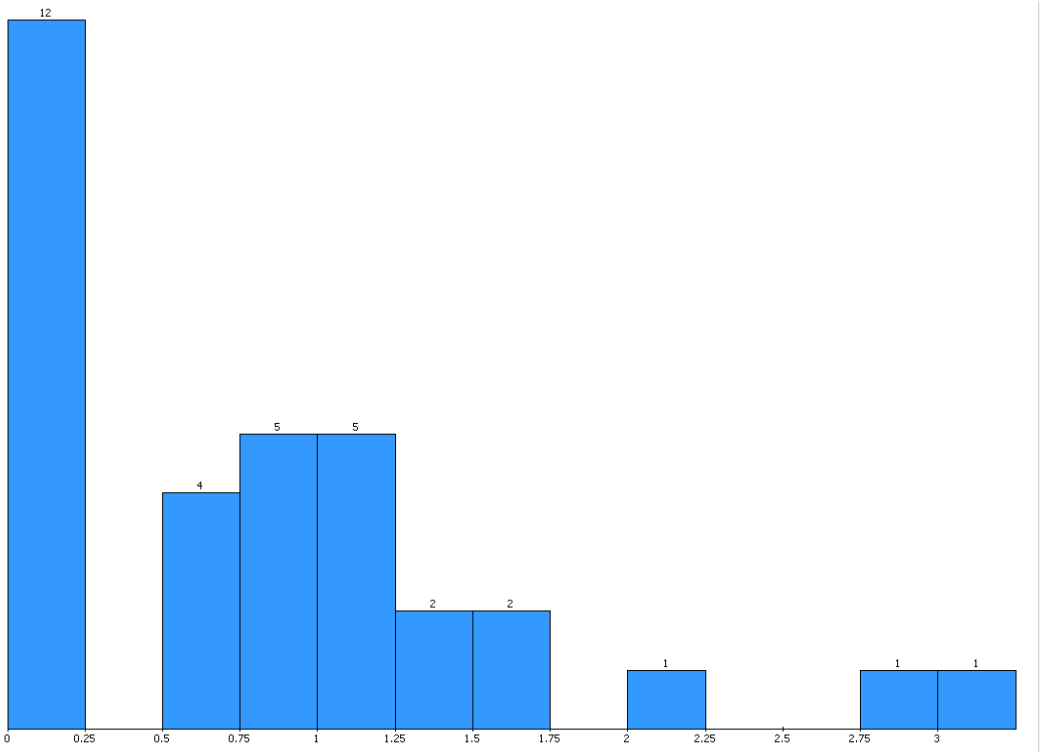

Figure 2: Sdmetric tool that measure the relational cohesion metric

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

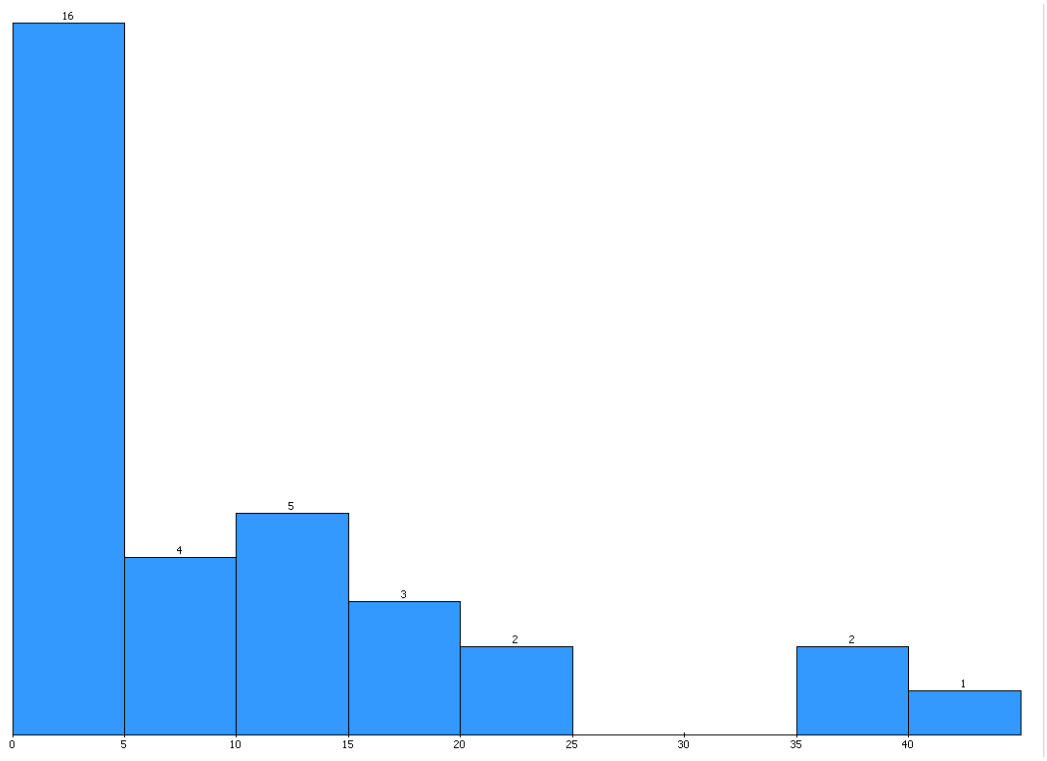

Figure 3: Sdmetric tool that measure the efferent coupling metric

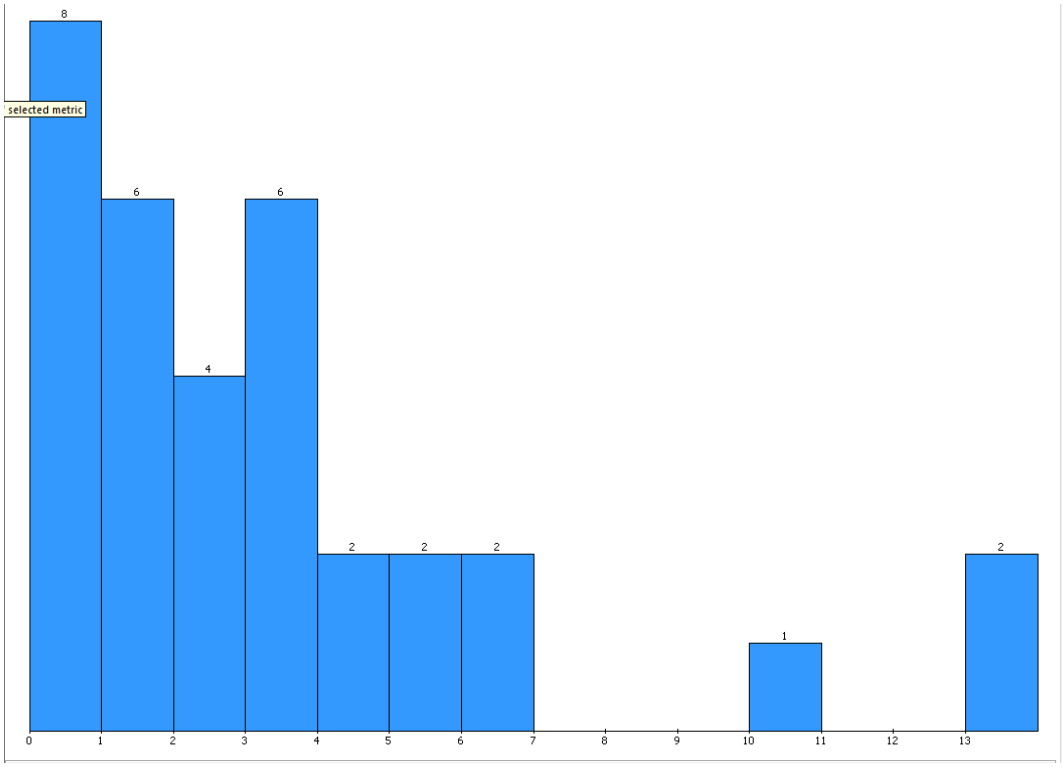

Figure 3: Sdmetric tool that measure the number of UML dependency

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

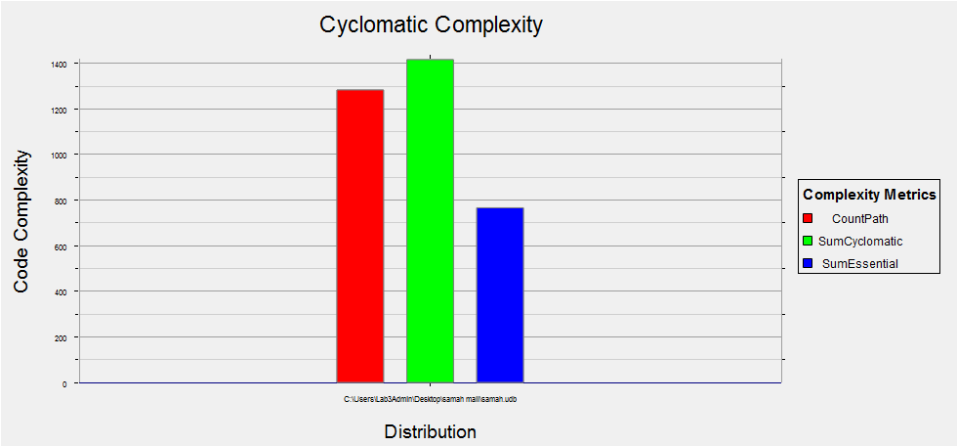

Figure 4: Understand tool that measure the complexity of software architecture

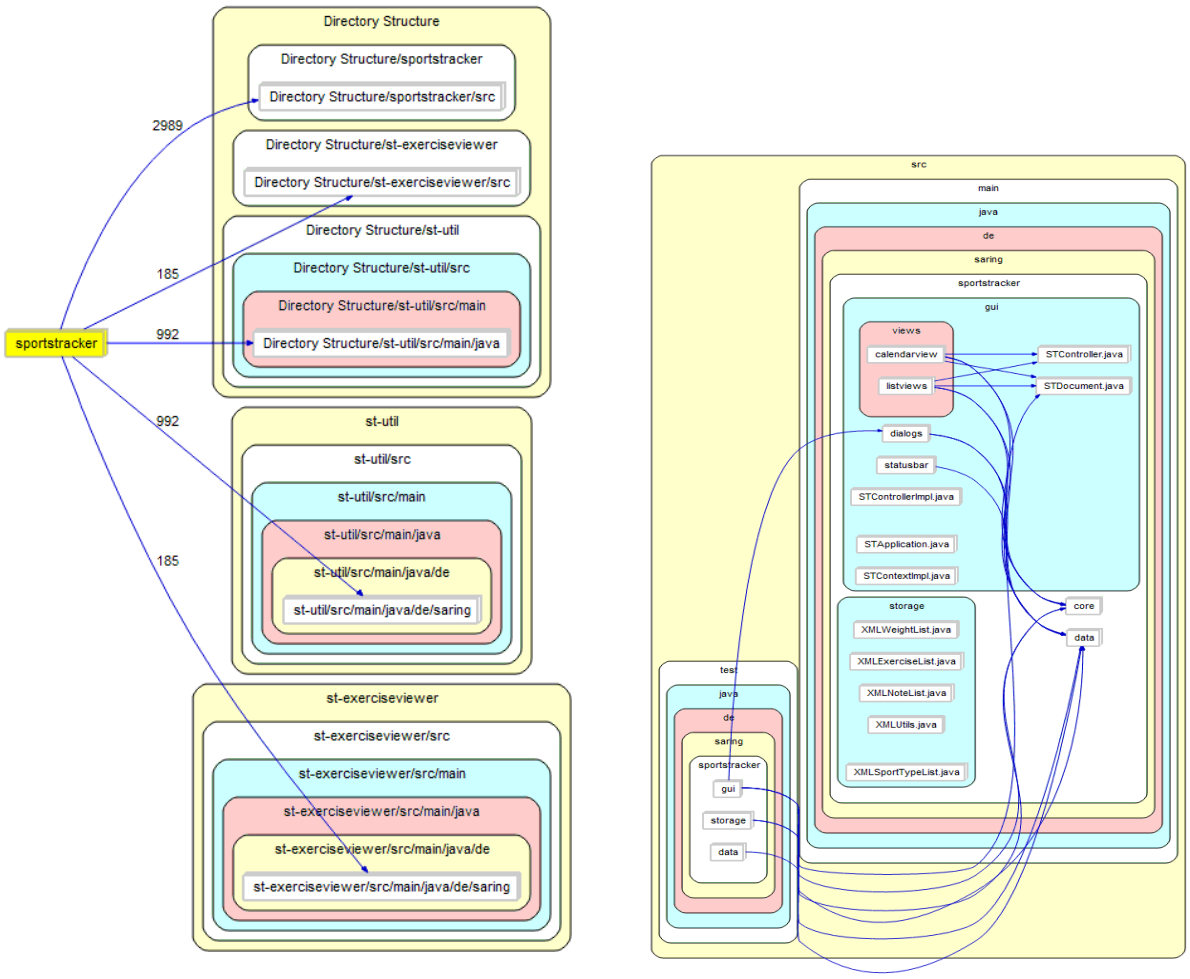

Figure 5: The dependency of software architecture

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

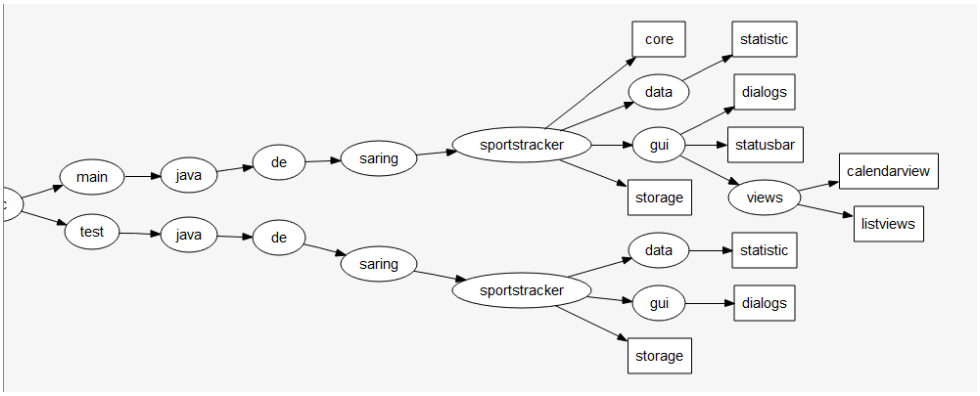

Figure 6: understand tool that measure the dependency of software architecture

https://sites.google.com/site/ijcsis/
ISSN 1947-5500