BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **19**, No 3
Sofia • 2019 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2019-0026

Simplifying the Structural Complexity of Software Systems

Thamer Al-Rousan[1], *Hasan Abualese*[2]

[1]*Faculty of Information Technology, Isra University, Jordan*
[2]*Faculty of Information Technology, Ajloun National University, Jordan*
E-mails: Thamer.rousan@iu.edu.jo homoush@yahoo.com

***Abstract*:** *Simplification of execution traces is peculiarly important in the case of software comprehension. The objective is to make execution traces in ways that are more tractable and less difficult. However, the simplification process is a difficult task, particularly, in object-oriented contexts. Due to coupling, execution traces of object-oriented systems involve the Spaghetti Architectures phenomenon, which is a very complicated structure of dependencies. Therefore, the simplification process needs a well-established approach to be helpful for software comprehension. Otherwise, the simplified execution traces will be informative as their structures will involve several gaps that lead to a misunderstanding process. This research uses decoupling to guide the simplification of object-oriented execution traces. Specifically, decoupling truthfully can decrease the complexity of execution traces without eliminating the trace components and making numerous gaps in the trace structure. Then, decoupling can solve the problem of the Spaghetti Architectures phenomenon. A controlled experiment was conducted to empirically validate the usefulness and effectivity of the suggested work. There was a significant statistical added value demonstrated in the time required and the accurate solutions of the tasks being solved. More precisely, 25% less time required with a 62% more correct solutions were achieved solving the experiment's comprehension tasks*.

***Keywords*:** *Decoupling, execution trace analysis, object-oriented, software comprehension, software maintenance.*

1. Introduction

Object-oriented software systems are popular platforms in many organizations in the world. Therefore, the maintenance of these software systems is indeed becoming an essential activity.However, the maintenance activities require a long time and are tedious, and expensive due to the problems of modern documentation. Thus, the activities of maintaining software consume more than 66% of the funds of the software systems[1].

The main activity of software maintenance that leads to higher maintenance costs is software comprehension. In particular, the comprehension activity consumes

about half of the maintenance costs [2]. For example, the maintenance time is mostly assigned to understand the system of the software and to investigate the influence of the suggested modifications [3]. Furthermore, the comprehension cost and time raise more if the software system has been updated by several maintenance cycles [4]. Hence, software comprehension tools and techniques are required to reduce maintenance cost and time. However, the mentioned techniques and tools must be based on complete and fresh resources, which might be limited to just the source and object codes related to the software under study [5, 6].

Execution traces of software systems are ones of the up-to-date and reliable resources of study. Unfortunately, the execution traces for current object-oriented systems involve a huge and complicated structures of information which is known as the "Spaghetti Architectures" phenomenon [7]. Hence, analyzing the execution traces needs tools and techniques to simplify them [8]. The objective is to help the comprehension of the content of execution traces and yet minimize the maintenance time and cost. Nevertheless, it should be noted that the problem is not confined to the complexity of trace size only, but also to the complexity of trace structure. Also, the importance of the trace components to program comprehension is usually different from one component to another [4]. For example, there are system components existing which obscure the connections among several other components without impacting any importance to the software comprehension such as utilities.

The study in this research establishes a trace simplification approach that depends on decoupling techniques. The main concept of using decoupling is the reality that coupling property has a tougher correlation to the complexity of the trace structure more than its size.

2. Complexity, coupling, and decoupling

Complexity means the level of difficulty comprehending and verifying a software program or a certain software component [9]. Complexity, in the literature context, is characterized by different aspects like size and coupling. Yet, the execution trace size has inability to characterize the complexity of the execution trace structure. For instance, the structures of any two different execution traces are almost different even though they have a similar size. Instead, coupling is a respectable representative of structural complexity as it is able to illustrate the system hierarchy, and the structural dependencies among the system modules.

A Coupling is a controlling mechanism which assesses relations between the components of a software system [10]. The objective is to enable users to understand how a system component relates to other components prior to any modification is being considered. In coupling, two different components are coupled when they are connected to each other by any type of connection or relationship [11]. As a "metric", coupling was first presented as a measure of the linking intensity which is formed between 2 modules [12]. The coupling notion was adapted to object-oriented systems [13].

Nonetheless, coupling is straightforwardly linked positively to the structural complexity. For instance, strong coupling raises the structural complexity because of

the higher interrelation of the system components, while weak-fitting coupling causes little structural complexity because of the lower interrelation of the system components. In return, coupling is inevitable in software systems. This is because there is intentional programming of tight coupling when top-quality performance and rapid completion time are critical for a system. However, an extremely solid coupling means a difficult system structure; and thus, the structural complexity of the system is anticipated to be elevated.

Both complexity and coupling have a negative impact on software maintenance and comprehension. This is because when coupling a component with further components, more components and links are needed to be reviewed, which makes it more difficult to understand that particular component. Additionally, assembling coupled components may need more effort and/or time because of the excessive intra component dependence. The reason is as well identical in maintenance when a system component requires modification by a maintainer but is connected to several other components, additional constraints may be applied for making the modifications. Consequently, it is better to couple as loose as possible to make sure that the modifications to a certain system component have a limited effect on the remaining components.

Decoupling is one solution to alleviate coupling problems which means reducing correlation among system components. The objective is reducing the complexity of the software system structure and to facilitate comprehending the program. This study utilizes the modularity patterns to carry out decoupling. Particularly, a customized module facade is utilized to produce a subsystem and also decouple it. More details are explained in the next section.

3. Decoupling for execution trace simplification

A new approach for execution traces simplification is proposed in this research, which breaks the complexity of execution traces and makes them analyzable. In general, execution traces simplification tools and techniques aim to provide the capacity to effectively retrieve the behavioral design paradigms from the execution traces of a certain software system [4]. In addition, executing trace simplification tools and techniques can be useful in several cases such as enabling top-down analysis and extracting various outlooks at different scales of abstraction for the contents of a particular execution trace. Additionally, these techniques and tools can also handle the documentation issues of software systems dynamics.

The main challenge to any execution trace approach is to determine the trace simplifying technique which is most suitable for the needs of the context. Unfortunately, most execution trace simplification techniques are based on relating trace complexity to its size only. Therefore, they are designed to remove some of the trace components such as utilities. The problem in the removal process is that it causes in making gaps in the structure of that particular trace, which might cause a misunderstanding problem.

Taking into consideration that coupling is linked to the complexity greater than the size, the argument of this research is that applying the decoupling process will

alleviate the complexity of execution traces, and truthfully decrease the issue of Spaghetti Architectures phenomenon. Decoupling could be performed using modularity patterns. Modularity indicates the possible extent of separating and recombining the components of a system; to make its maintaining and implementing goals more attainable. One of the well-known modularity patterns is module facade.

3.1. Module facade

It is a customized version of pattern Facade. Its target is composing and encapsulatinga whole subsystem to guarantee separating the software components so as to decrease their complexity and reduce the issue of "Spaghetti Architectures" phenomenon. However, the main challenges are composing a suitable subsystem which is able to tackle the aforementioned problems, and the way to properly execute decoupling without dismissing "links" that, if taken away, may neglect a vital relationship; which eventually would lead to a misinterpretation affair. Therefore, decoupling every type of class is not possible. For example, we cannot decouple classes that act vital parts in the software. Moreover, classes which have no vital role could be decoupled such as utilities.

Utilities have high coupling because they are reused frequently. Therefore, decoupling utilities will actually reduce the structural complexity and yet facilitate the software comprehension. Utilities could be detected by using several techniques [14, 15]. Afterward, decoupling is performed by using Facade processing. In this case, utilities are grouped to represent the Facade subsystem.

The subsystem volume shall rely on simplification scale which varies from a situation to another. For instance, users commonly demand further particularities when they are advancing to further comprehension cycles. Additionally, the number of components which are engaged in the trace shall as well influence the resolve of choosing the simplification scale. Consequently, the simplification scale ought to be flexible and confirmed by the user. However, confirming the simplification scale necessitates assignation of some parameters to involve the precise simplification scale threshold.

3.2. The algorithm

Different data forms of classes are founded to facilitate the execution trace, to be more specific, the next class structures were formed:

TotalClasses (TC): A structure of the entire group of classes of an execution trace ET.

LibraryClasses (LC): A structure of the un-application and library classes of an execution trace ET.

UndesiredClasses (UC): A structure of the unwanted classes of an execution trace ET.

AnalysisClasses (AC): A structure of the final desired classes as follows:

$$\mathbf{AC = TC – LC – UC.}$$

Algorithm

Step 1. Assign values to some parameters in order to proceed in the procedure of decoupling. The Simplified Ratio (SR) is the key parameter. SR parameter is

specified by a certain threshold. In particular, user's input decides the parametric value. SR parameter is outlined to determine how many classes should be DeCoupled (DC). In this case, "classes that have highest values for utility metric are decoupled". Then, the DC value is determined as

$$\mathbf{DC = SR \times AC.}$$

Step 2. Create class structures: TC, LC, UC, and AC.

Step 3. Use AC structure to extract the requested coupling information.

Step 4. Use the results of Step 3 to calculate utility measures for each class.

Step 5. Create a subsystem which is composed of the classes which would most probably would use utilities according to values of utility metric and the value of SR parameter.

Step 6. Generate the concluding simplified execution trace as a result of Facade processing. Also, generate further outputs such as statistics view.

Step 7. If the concluding simplified execution trace is unsatisfactory, in that case, the SR parameter can be modified.

Step 8. The concluding simplified execution trace might be manipulated manually.

However, the final simplified execution trace might be still too detailed or too abstract. Also, the users may need to collapse or expand it for any reason. In such situations, we can easily adjust the SR parameter and rerun the algorithm to generate a new final simplified execution trace that satisfies users' needs.

4. Related works

Trace simplification instruments refer to the computerized utility that is utilized to automatically analyze the dynamic data that is involved in execution traces in such a manner it is readily utilized, fast, and entailing the least person's exertion. Even though those dynamic tools might aide comprehending the software systems, they are still suffering from some weaknesses.

Many research studies target at providing automatic utility detection [16, 17]. Nevertheless, most of them are based on using "fan-in analysis techniques." Therefore, they are depending on either dependency graphs or call graphs, which suffer from precision issues [18]. Notwithstanding, there is not much research depending on the original source of data utilizing a specific technique, for instance, dynamic coupling measurements [14]. Another drawback is that removing utilities leads to create gaps in the structure of the execution trace, and yet might lead to encounter misunderstanding situations.

Pattern-matching techniques are used to categorize identical patterns of events with a view to represent them as a single pattern [19]. They are able to decrease the size and complexity of execution traces effectively when they are generalized [20]. However, a matching criterion entails regulating certain parameters, for example, depth-of-sequence to examine if two "distinct sequences" of events are comparable. Sampling methods are used to efficiently decrease the execution traces size [21]. They are based on selecting and analyzing representative samples instead of considering the whole trace. The sampling process is either chosen according to some

parameters, or it is chosen randomly. However, in the former case, selecting the proper sampling parameters considered a big task, while in the latter one, the sampling execution trace will not be representative.

Clustering techniques are used to group the elements of the trace to correspond to several criteria [22, 23]. Consequently, they just display the relationships between the main elements, like the architectural level. Consequently, clustering techniques are able to effectively lessen the size of traces. In return, the clustering techniques entail identifying a grouping criterion, and existence of the system architecture.

The common weakness for the above mentioned techniques is that each one of them relies on the removal of several elements or events from the "execution traces." Even though the removing process could handle the issue of traces size, it yields in making gaps in the structure of execution trace, which might lead to encounter misunderstanding situations. However, the coupling is proposed to handle this problem [24].

5. Controlled experiment

The aforementioned algorithm was implemented in a tool called eXecution TRAce DECoupling (XTRADEC) to quantitatively evaluated for the purpose of program comprehension, by following an empirical design. XTRADEC is particularly validated experimentally using controlled experimentation; in order to achieve a quantitative measure of the effectivity and efficiency of the projected research. In fact, controlled experimentation is particularly essential to demonstrate the usefulness in practice for trace analysis techniques and tools. However, not that much work has been conducted for the purpose of controlled experimentation of trace analysis techniques and tools. Two groups are used as a sample, one is considered as a control group, while the other is used as an experimental group. Eclipse IDE was used by both groups to discover answers to certain comprehension tasks, while XTRADEC was only used by the experimental group.

The experimental model used for this research is based on the Goals/Questions/Metrics (GQM) model [25]. The goals, questions, and metrics are represented in this experiment as the following:

1. Goals. The most important goal is statistical analysis of the improvement of the degree of program comprehension using XTRADEC, and to estimate the speed and accuracy of answering certain comprehension tasks by using XTRADEC. Two sub-goals are traced from the aforementioned goal, the first one related to the efficiency (the time required for the completion of the presented comprehension tasks), while the other goal is concerned with the accuracy (correctness) of the resulting solutions.

2. Questions. The advantage provided by using XTRADEC to perform a number of comprehension tasks, is measured by using a questionnaire. It is divided into three parts. The first part is outlined for the purpose of collecting the subjects' personal and professional backgrounds. The next part is used for listing the comprehension tasks that the subjects must accomplish for the purpose of evaluating the understanding process. While the final section is regarded as the "usability

section"; as it is mainly concerned with subjects' point of view regarding the usability and the advantages of the XTRADEC tool.

3. Metrics. A pair of metrics representing the dependent variables were used to assess the added value. Based on the time calculated finding answers for certain tasks; the first metric is considered as efficiency related. The second metric is used to be concerned with the effectiveness; which is measured depending on the accuracy of the given solutions. The variables selected in this experiment are similar to what was used in [8].

5.1. Experiment questions and hypotheses

A certain question is mostly asked while using the tools and techniques of trace analysis. It concerns the amount of improvement that they can bring to program comprehension. The very same question is used in this trial regarding the improvement that XTRADEC would add to program comprehension. Three independent variables are used in this experiment; namely XTRADEC tool and Eclipse IDE, in addition to the tasks that should be resolved. On the other hand, the time necessary to answer the aforementioned tasks, and the degree of accuracy of their solutions are the two main dependent variables in our experiment. Therefore, two sub-questions arise from this experiment's main question:

1. How much time reduction can we achieve by using XTRADEC in order to get solutions to certain comprehension tasks?
2. How much the accuracy of the solutions of certain comprehension tasks can be increased, by using XTRADEC tool?

This experiment's chief question is associated with a null hypothesis; which states the following:

Hypothesis H_0: Using XTRADEC will not help in understanding targeted application to achieve answers to certain comprehension tasks.

Consequently, there are two sub-hypotheses which are associated with our experiment's two sub-questions:

Hypothesis $H1_0$: The time will not be decreased to achieve answers to certain comprehension tasks by using XTRADEC.

Hypothesis $H2_0$: The accuracy will not be increased to achieve answers to certain comprehension tasks by using XTRADEC.

5.2. System subject and subjects

Thirty participants from industry and academia were involved as subjects in this trial; all with experience in the field of software industries. The selected group consisted of ten programmers, a couple of system analysts, ten programmers, in addition to eighteen postgraduate students in software fields. The experiment assured an adequate targeting of the meant population by having both academia and industry as the two main subjects; which in turn would assure the reliability of the experiment's results [26].The concept of voluntary participation was the corner stone of proper motivation for all of the subjects. Furthermore, none of the targeted subjects had any previous experience in the XTRADEC tool.

The subjects are fairly divided into two groups by identifying a list of controlled factors. They were divided into pairs witha similar level of experience as far as these factors are concerned, with each member of the pair assigned either to the control group or to the experimental group. This method of dividing the subjects assured an equal level of experience in each one of the two groups. Particularly, the two groups have an overall average of expertise of 1.71 (Stddev= 0.42) for the control group, and 1.69 (Stddev= 0.45) for theexperimental group.

CHECKSTYLE tool, written in Java, is the the subject system. It checks whether application's codes follow the standards of coding or not. CHECKSTYLE is composed of 21 packages, 310 classes and about 57K lines of statements. Hence, CHECKSTYLE is believed to be aadequate representative of systems applications in real life. Additionally, reproducing this experiment's results is made possible due to using CHECKSTYLE; as this tool is an open source tool. The subjects' lack of experience in the subject system (CHECKSTYLE) nullifies any possibility of any bias in favor of any of the two study groups.

The execution of CHECKSTYLE shall be conducted depending on different scenarios to end up with the required dynamic information; as the required trace data must be achieved for XTRADEC tool in addition to CHECKSTYLE. Two trace files are used in this trial. The former was generated by running CHECKSTYLE with 64 kinds of checks and result in 31,260 calls. The latter was generated by running CHECKSTYLE with 6 checks only and result in 17,126 calls. Nevertheless, it is not possible to manually analyze either one of the two traces in limited time.

5.3. Comprehension tasks

The selected tasks should represent true situations where software maintenance is required, without any bias for neither the subject groups nor the subject tools.

Table 1. Depiction of comprehension tasks

Tasks	Activities	Depiction
Task 1	A1, A7, A9	Globally understanding main stages in a typical CHECKSTYLE scenario
Task 2.1	A4, A8	Identifying three classes with a high fanin and a low fanout
Task 2.2	A4, A8	Identifying a class in package *X* with a strong coupling to package *Y*
Task 3.1	A1, A2, A5, A6	Describing the life cycle of check *X* during execution
Task 3.2	A3, A4, A5	Listing of all interactions between check *X* and class *Y*
Task 3.3	A3, A4, A5, A9	Listing of additional interactions in case of check *Z*
Task 4.1	A1, A3	Providing a detailed description of the violation handling process
Task 4.2	A1, A5	Determining whether check *X* reports violations

Different frameworks of comprehension are suggested to take on these issues, like the ones chosen by [27]. The comprehension tasks are categorized in [27] depending on nine main activities. The mentioned activities are carefully classified to include different aspects of abstractions ranging from getting a complete concept of understanding the software system, to conducting a certain task, in addition to representing the information from a static and dynamic point of views. Table 1 shows eight comprehensive taks for CHECKSTYLE based on the nine activities [8]. The eight tasks follow an "open-question" format rather than "multiple-choice"; which eliminated the possibility of guessing by the subjects and made the tasks more

representative of real maintenance situations. Then, two evaluators provided points for each answer as the points were in the range from 0-4, by referring to a reviewed solution model.

5.4. Pilot studies

Two qualified programmers conducted two pilot studies to ensure the clarity and doability of the "comprehension tasks" during the specified time. Both of them were not participants in the principal trial; as the results they achieved were not considered for the analysis. They accomplished the comprehension tasks during the 90-minute specified time, but the allocated time was too tight for the control programmer. Accordingly, regarding excluding the last two comprehension tasks from the experiment's analysis, we opted to follow the original pilot studies [8].

5.5. Experimental procedure

We conducted our experiment in the Jordanian Institute, Al alBayt University. The participants were informed to report to the University's computer lab as soon as we got permission. The specifications of the software and hardware of this lab's workstations were all similar; 17-inch screens, 2GB RAMs, and Pentium 4 processors. The control group and experimental group each consisted of 15 participants, as they were randomly allocated to the groups. The subjects' experience was carefully considered while allocating the subjects to the two groups. Eclipse IDE was used to figure out answers to 8 different tasks in the control group, while XTRADEC tool was used in for a similar purpose in the experimental group.

Subjects from both groups were informed about the way to use Eclipse IDE by being briefed for 15 min, in addition to 10 more minutes of briefing the experimental group about the way to use XTRADEC. The experiment's researchers, along with two faculty members of Isra University supervised the experiment; as they answered any questions or inquiries by the subjects, and ensured that the subjects wouldn't get any other mean of help; such as seeking any help from outside or by getting access to information online. Additionally, the subjects were unfamiliar with the goals of the experiment. A large digital watch was provided to the lab to enable the subjects of recording each comprehension task's starting recording time. The time limit was determined at 90 min while affording the subjects the flexibility and freedom when it comes to time; in order to eliminate any time pressure restrains. The subjects were made aware of the time expiry by passing a note to them while being allowed to carry on finding solutions to what is remaining of comprehension tasks.

5.6. Analysis and results

This experiment's chief goal is a statistical analysis of the degree of improvement (in terms of speed and accuracy) of program comprehension by using XTRADEC while solving comprehension tasks. The improvement was measured depending on the degree of accuracy of the comprehension tasks achieved, and the time consumed solving them. So the two mentioned variables measured the efficiency when it comes to the time consumed, and the effectiveness with regard to the accuracy of the

solutions. These two variables are demonstrated in Fig. 1, which illustrates the time required by both two groups, while Fig. 2 demonstrate the accuracy of each comprehension task solved. Descriptive statistics of both the solutions' accuracy and the time required to solve the tasks is illustrated in Table 2.

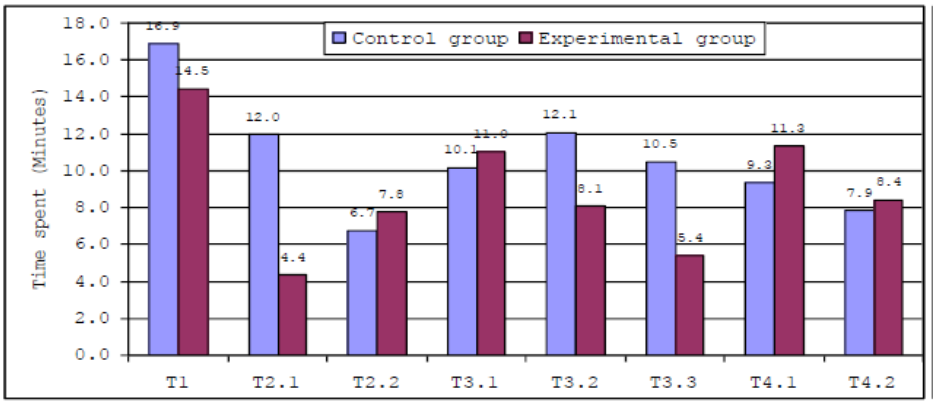

Fig. 1. Time average per task

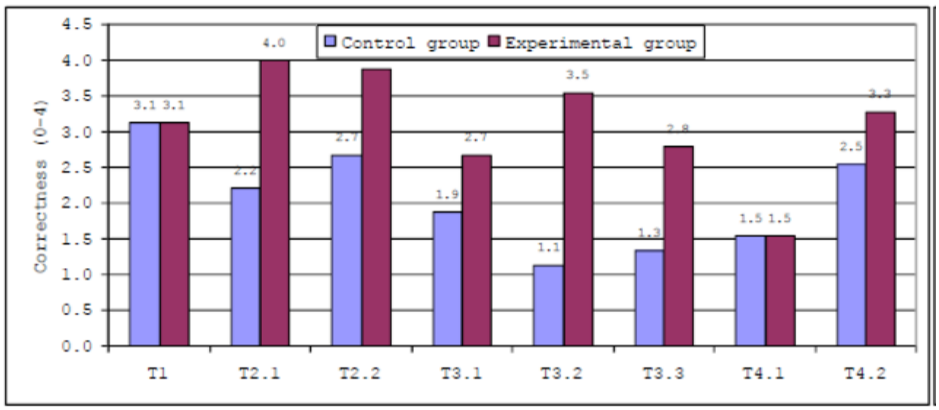

Fig. 2. Answers_Correctness average per task

Table 2. Time_Spent and Answers_Correctness descriptive statistics

	Time_spent		Answers_Correctness	
	Control Group	Experimental Group	Control Group	Experimental Group
Mean	68.27	51.13	12.33	20.00
Difference		–25.10%		62.16%
Standard Dev.	10.93	14.63	3.37	2.39
Min.	55.00	29.00	4.00	14.00
Max.	89.00	80.00	17.00	22.00
Median	67.00	50.00	13.00	21.00
Variance	119.50	213.99	11.38	5.71

The parametric “Student’s *t*-test” and the nonparametric “Mann-Whitney test” are used in this trial.Depending on the experiment’s alternative hypotheses, One-tailed variant is used. A typical confidence level of 95% is maintained by choosing a 0.05 significance level. The statistical package of SPSS was used to conduct statistical tests in regard to the time required, and the accuracy of the solutions of comprehension tasks.

5.6.1. Time_Spent results

By taking Table 2 into consideration, an average time of 68.27 (stddev=10.93) was required by members of the control group, while members of the experimental group needed an average time of 51.13 (stddev=14.63). Therefore, the experimental group’s subjects required a shorter time solving the comprehension tasks, than their counterparts from the control group. The results showed that they needed 25.10% less time than what subjects in the control group needed. The box plots and mean for the time_spent by subjects from the two groups, are shown in Figure 3; which in turn confirms our results.

The requirements are validated for using a parametric test; as the results revealed the significant role played by the XTRADEC tool in reducing the time required solving the comprehension tasks. The results show a *t*-value of (3.634) and a *p*-value of (<0.001; less than 0.05); which is considered a remarkably lower time required by members of the experimental group, than the others from the control group. Therefore, the alternative hypotheses H1 is taken into consideration instead of the null hypotheses $H1_0$. The H1 alternative hypotheses indicate that the use of XTRADEC will significantly decrease the time required to get solutions for certain comprehension tasks.

5.6.2. Answers_Correctness results

By taking Table 2 into consideration, we notice a variation of the correct answers between the two groups; as the average of the answers_correctness of the control group’s subjects is 12.33 from a total of “24” points, with a (stdev=”3.37”). On the other hand, the average of the experimental group’s subjects is 20.00 out of the total of “24” points, with a (stdev=”2.39”). Accordingly, members of the experimental group achieved better results (more correct solutions to the tasks), than their counterparts from the control group.An average of 62.16% more accurate solutions was shown by the experimental group, than the results achieved by the control group.These results are shown in Fig. 4, which shows the box plots and mean of the correct solutions of the subjects from the two groups.

The Mann-Whitney nonparametric test is used due to the lack of success of the normality test. The remarkable influence of the XTRADEC tools was evident in the results and the correct solutions. The accuracy of the solutions achieved by the experimental group is indicated by au-value of (7.500), and a *p*-value of (0.000; less than 0.05). Therefore, the alternative hypotheses H2 is taken into consideration instead of the null hypotheses $H2_0$. As the alternative hypothesis indicate that the use of XTRADEC will significantly raise the answers accuracy to certain comprehension.

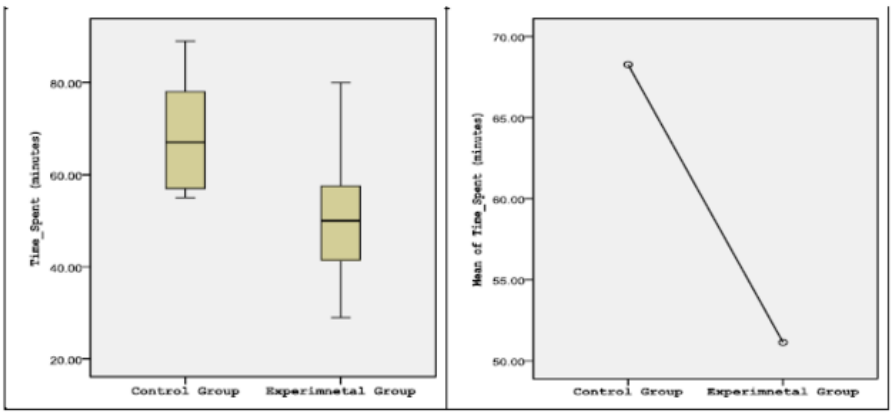

Fig. 3. Box Plots and Mean for Time_spent

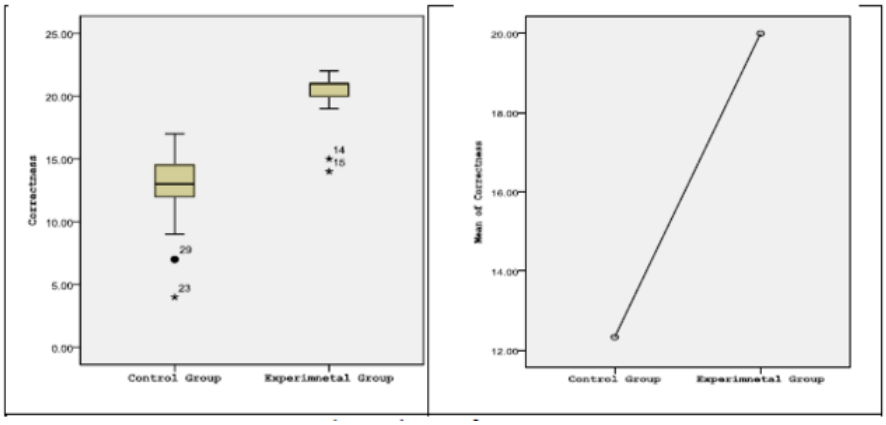

Fig. 4. Box Plots and Mean for Answers_Correctness

5.7. Individual tasks analysis

Figuring out the type of comprehension tasks most evidently benefiting from using the XTRADEC tool is the chief purpose of this experiment's analysis. The subjects' performance in terms of the time required to solve each comprehension task and their accuracy of achieving correct solutions, is compared.

Fig. 1 indicates that both groups were similar in the time preference aspect. The experimental group consumed a lesser amount of time to solve the tasks Task1, Task 2.1, Task 3.2, and Task 3.3, while the tasks Task 2.2, Task 3.1, Task 4.1, and Task 4.2 required less time by the control group to be solved. Normality and homogeneity variances tests were conducted to come to conclude that using or not using a parametric test for the time required for each task. These tests were successful for three tasks only; Task 2.2, Task 4.1, and Task 4.2. Therefore, their significances were put to the test by using the parametric Student's *t*-test (*t*). On the other hand, the significances of Task 1, Task 2.1, Task 3.1, Task 3.2, and Task 3.3 were tested using the Mann-Whitney (u) nonparametric test.

The parametric Student *t*-test showed no time significance in any of the tasks that the test was conducted on, while time significance was evident in four of the five

tasks which were tested by the nonparametric Mann-Whitney test. Tasks Task 1, Task 2.1, Task 3.2, and Task 3.3, in particular, registered high significance in time required as they registered 0.02, 0.00, 0.002 and 0.001-time values respectively. The aforementioned time significance was registered in the experimental group. So we conclude that when it comes to the time variable, the control group did not show any time advantage over the experimental group in any of the tasks being solved.

While as far as the accuracy variable is concerned, Fig. 2 indicates that the experimental group was not outperformed by the control group in any of the tasks solved. Tasks Task 1 and Task 4.1 showed similar scores for both groups. The normality test was non-successful in testing the accuracy and correctness of each task; so the Mann-Whitney (u) nonparametric test was conducted on all the tasks. The results indicated that five tasks, Task 2.1, Task 2.2, Task 3.1, Task 3.2 and Task 3.3, all showed significance in terms of the correctness; as they registered values of 0.000, 0.013, 0.002, 0.000 and 0.000, respectively. These mentioned significant results were registered by the experimental group, as it outperformed the control group.

5.8. Discussion

This section is concerned with interpreting the differences in the results between the two groups. The variations in the results of "time_spent" and "answers_correctness" are discussed in the following subsections.

5.8.1. Time_spent variations

The analysis indicates that the subjects who used the XTRADEC tool, required shorter time solving the comprehension tasks. This result can be attributed to several factors.

Generally speaking, dynamic information indicates the real runtime activity which alleviate the cognitive load which necessitates to figure out how software elements operate at execution time. Second, a top-down comprehension is enhanced by the XTRADEC tool; which eliminates any noise and utility elements that are expected to interfere with the comprehension process. Third, the subjects are directly guided to a specific target by the XTRADEC tool; which is achieved by supporting a strategy of scope filtering; which in turn cancels any need of searching all the system's modules. Finally, that could be attributed to the XTRADEC tool's high speed and response.

On the other hand, a longer time required by subjects in the experimental group to solve the required comprehension tasks could be attributed to certain factors. First, visualization tools, such as UML tools, are not supported by the XTRADEC tool. Actually recovering the system's behavioral design models is one of the XTRADEC tool's direct applications. Second, the user could be confused when using the XTRADEC tool; since it is a standalone tool that requires to switch among many available tools. Integrating XTRADEC into IDE environment, as an extra plug-in, could be considered as a solution to this issue. Finally, the experimental group subjects' unfamiliarity with using the XTRADEC tool arose from the very short tutorial.

5.8.2. Answers_Correctness variations

The analysis indicated a significant effect by the XTRADEC tool when used by subjects from the experimental group to solve the required comprehension tasks; as it yielded more correct solutions. Dynamic analysis is the main factor to achieve such results; as it is able to accurately tackle dynamic binding and polymorphism features in programmin. So more correct answers can be achieved when using XTRADEC by showing the real elements participating in each call; which makes the interactions among the elements easier to be understood.

In return, the lack of knowledge of dynamic analysis is an important factor that leads to uncorrect answers. Unfortunately, the two groups had less than an average of 1, on a scale of 1-4. Software communities shall be aware of the importance of dynamic analysis to eliminate such a problem.

5.9. Possible threats

The experiment's results are affected by various factors. The subjects and the system of the subject, in addition to the comprehension tasks, could have an effect on the results. The probable threats are outlined in the following subsections.

5.9.1. Threats to subjects

Various factors could have had an influence in regard to the subjects' threats, among which are the subjects' competence, the way they were grouped, motivating the subjects, and their representation. All threats are thoroughly explained in the following paragraph.

First, the subjects' capabilities and competency were taken care of through the questionnaire, particularly in its opening section.; the results of the questionnaire revealed that the subjects were competent enough to participate in the experiment. For instance, the subjects had a medium experience in JAVA language where average JAVA knowledge = 2.87 (Stddev= 0.63) which is considered as "Advanced" level. Additionally, they all had a computer science/software engineering degree. Second, the subject grouping threat was negated by dividing each pair of subjects with identical experience level to the experimental and control groups in a random fashion; the result of such random distribution was two groups with a similar level of experience. In particular, the experience levels were measured for the groups as 1.71 (Stddev=0.42) for the control group, and 1.69 (Stddev=0.45) for the experimental group. Ultimately, the way the subjects were assigned led to having two groups with no bias. Third, all subjects volunteered to participate in the experiment, which nullified the threat of motivation; as they were looking forward to learning new concepts that would eventually have a beneficial influence on their future careers. Moreover, the value of the subject was an encouraging factor for the subjects to prepare for the experiment.

5.9.2. Threats to subject system

The real size and system availableness are the main threats to the subject system. The CHECKSTYLE system software is the chosen subject system in our experimentation.

However, CHECKSTYLE is a famous software system with “open-source” and realistic size. Furthermore, its free availability renders it usable in other experiments. Upon assessing the subjects’ experience levels of our “subject system”, we found out that they had no expertise level on CHECKSTYLE; which eventually nullified any possibility of bias to either group.

In reality, we could have had different results; had our subject system been different, or had one more system been used. However, an extra load of work on the subjects’ part would have been required, had we used one more subject system. Due to the duration of the primary experiment (3 hours), the possibility of adding more workload on the subjects would be unreasonable. Additionally, finding software programmers or researchers would be hard; as they are not willing to spend such a long time for even one experiment. Using undergraduate students as this experiment’s subjects would lead to unreliable results; due to their low skill levels.

5.9.3. Threats to comprehension tasks

There are numerous threats to the comprehension tasks, among which are, having a bias in favor of the XTRADEC tool, having tasks which are hard to answer, representing contexts of real comprehension tasks. All the threats are thoroughly outlined in the next paragraph.

First, the bias towards the XTRADEC tool while designing the comprehension tasks is avoided as the recognized model was used to design the tasks. Second, the threat of having very hard tasks to solve is not existing based on the accuracy of the task solutions as numerous subjects from the two groups scored full marks, like in Task 1, Task 2.1, Task 2.2, Task 3.2, and Task 4.2. However, subjects from both groups scored three out of four in Task 3.1, Task 3.3, Task 4.1. Moreover, the comprehension tasks’ difficulty was rated by the subjects with a rate of 2 on a 0-4 difficulty scale which translates the task difficulty as intermediate. The third threat of not representing real comprehension task by the experiment’s tasks is nullified by using Pacione’s recognized model to design the tasks. Pacione’s model requires using many levels of an abstract, and to get general knowledge and detailed understanding. In addition, the tasks follow an open-question format rather than multiple-choice; which renders the tasks a more realistic model of industrial reality and representing true comprehension conditions.

6. Conclusion

The contribution of this paper was to present the decoupling approach for traces simplification. In addition to tackling the problem of traces size, the approach attempts to overcome the problem of traces complexity issue in a more efficient way. The decoupling strategies are employed to lessen the coupling complexity problem. Decoupling has been made at class level, so can be applied on smaller components such as methods or larger components such as packages. Also, compare it with other trace analysis techniques. Furthermore, one straight implementation of this strategy is recovering the behavioral designing paradigms of the software application.

Consequently, the proposed work implements top-down approach to retrieve the vital trace components that users want to look first before probing further details.

The value added to program comprehension by XTRADEC was quantitatively evaluated in our controlled experiment. Two groups were formed by a set of subjects with an even distribution of knowledge and experience. Subsequently, eight comprehension tasks were solved by the subjects in both groups. Then, the subjects' performances were evaluated in regard to the time needed solving the tasks and the accuracy of the answers correctness. The results of the experiment showed how useful the XTRADEC tool is in regard to program comprehension. That was evident in the statistic evidence in the outcomes of the aforementioned time_spent and answers_correctness variables.

This experiment's outcomes showed an average of 25% time improvement and 62% more correct answers as a result of using the XTRADEC tool. Finally, quantitative comparisons between XTRADEC and other trace analysis tools should be conducted.

Acknowledgments**:** *This research was supported by Isra University, Amman, Jordan. We thank our colleagues who provided insight and expertise that greatly assisted the research.*

References

1. Sommerville, I. Software Engineering. 10th Ed. New York, Pearson Education, 2015.
2. Bourque, P., R. Fairley. Guide to the Software Engineering Body of Knowledge. IEEE Computer Society, SWEBOK Version 3.0, 2014.
3. Storey, A. Tools and Research Methods in Program Comprehension: Past, Present and Future. – Software Quality Journal, Vol. **24**, 2016, No 1, pp. 187-208.
4. Hamou-Lhadj, A., T. Lethbridge. Understanding the Complexity Embedded in Large Routine Call Traces with a Focus on Program Comprehension Tasks. – IET Software, Vol. **11**, 2017, No 2, pp. 161-177.
5. Hassine, J., A. Hamou-Lhadj, L. Alawneh. A Framework for the Recovery and Visualization of System Availability Scenarios from Execution Traces. – Information and Software Technology, Vol. **96**, 2018, No 1, pp. 78-93.
6. Abu Al-Ese, H., A. Ghani, R. Mahmod, M. Saman. Java-Based Static Analyzer for Object-Oriented Software Metrics. – In: Proc. of Information Technology Colloquim 99 (INTEC'99), UPM Malaysia,1999.
7. Al-Rousan, T., H. Al Ese. Impact of Cloud Computing on Educational Institutions: A Case Study. – Recent Patents on Computer Science, Vol. **8**, 2015, No 2, pp. 106-111.
8. Cornelissen, B., A. Zaidman, A. Deursen. A Controlled Experiment for Program Comprehension through Trace Visualization. – Transactions on Software Engineering (TSE), Vol. **42**, 2016, No 3, pp. 201-216.
9. IEEE 610.12-90. ANSI/IEEE Standard 610.12-1990. IEEE Standard for Glossary of Software Engineering, IEEE.1990.
10. Fregnan, E., T. Baumb, F. Palomba, A. Bacchelli. A Survey on Software Coupling Relations and Tools. – Information and Software Technology, Vol. **107**, 2018, No 1, pp. 159-178.
11. Abdurazik, A. Coupling-Based Analysis of Object-Oriented Software. PhD Thesis, George Mason University, 2015.
12. Stevens, W., G. Meyers, L. Constantine. Structured Design. – IBM Systems Journal, Vol. **53**, 2016, No 2, pp. 115-139.
13. Arenas, M. Database Theory Column Report on PODS. – ACM SIGACT News, Vol. **49**, 2018, No 4, pp. 55-57.

14. Al-Rousan, T., H. Abualese. A New Technique for Utility-Class Detection in Object-Oriented Software. – Tem Journal – Technology, Education, Management, Informatics, Vol. **49**, 2019, No 4, pp. 157-169.
15. Abualese, H., P. Sumari, T. Al-Rousan, M. Al-Mousa. Utility Classes Detection Metrics for Execution Trace Analysis. – In: Proc. of 8th International Conference on Information Technology (ICIT'17), IEEE, Amman, Jordan, 2017, pp. 147-159.
16. Pirzadeh, H., L. Alawneh, A. Hamou-Lhadj. Quality of the Source Code for Design and Architecture Recovery Techniques: Utilities are the Problem. – In: Proc. of 12th International Conference on Quality Software, IEEE, NY, USA, 2016.
17. Rohatgi, A., A. Hamou-Lhadj, J. Rilling. Approach for Solving the Feature Location Problem by Measuring the Component Modification Impact. – IET Software, Vol. **10**, 2016, No 2, pp. 111-119.
18. Lee, B., K. Resnick, M. Bond, K. McKinley. Correcting the Dynamic Call Graph Using Control-Flow Constraints. Compiler Construction. – In: Lecture Notes in Computer Science, Vol. **4420**, 2015, pp. 80-95.
19. Hamou-Lhadj, A., T. Lethbridge. An Efficient Algorithm for Detecting Patterns in Traces of Procedure Calls. – In: Proc. of 1st ICSE international Workshop on Dynamic Analysis, TX, USA, 2013, pp. 33-39.
20. Kali, P., A. Srinivasan, S. Bishnoi. Oracle International Corp. Pattern Matching across Multiple Input Data Streams. U.S. Patent 9,934,279, Accessed on 4 May 2018.
21. Pirzadeh, H. Trace Abstraction Framework and Techniques. Ph. D Thesis, Concordia University, Montreal, Quebec, Canada, 2016.
22. Patel, C., A. Hamou-Lhadj, J. Rilling. Software Clustering Using Dynamic Analysis and Static Dependencies. – In: Proc. of 13th European Conference on Software Maintenance and Reengineering (CSMR'09), Architecture-Centric Maintenance of Large-Scale Software Systems, 2017.
23. Delias, P., M. Doumpos, E. Grigoroudis., N. Matsatsinis. A Non-Compensatory Approach for Trace Clustering. – International Transactions in Operational Research, Vol. **26**, 2019, No 5, pp. 1828-1846.
24. Abualese, H., P. Sumari, T. Al-Rousan, M. Al-Mousa. A Trace Simplification Framework. – In: Proc. of 8th International Conference on Information Technology (ICIT'17), IEEE, Amman, Jordan, 2017, pp. 63-72.
25. Basili, V. Software Modeling and Measurement: The Goal/Question/Metric Paradigm. University of Maryland, Technical Report, UMIACS-TR-92-96, 2011.
26. Penta, M., R. Stirewalt, E. Kraemer. Designing Your Next Empirical Study on Program Comprehension. – In: Proc. of 15th IEEE international Conference Program Comprehension, Japan, 2015, pp. 281-285.
27. Pacione, M., M. Roper, M. Wood. A Novel Software Visualisation Model to Support Software Comprehension. – In: Proc. of 11th Working Conference on Reverse Engineering (WCRE'11), IEEE, CS Press, USA, 2016, pp. 70-79.

Received: 07.08.2019; Accepted: 22.08.2019 (fast track)