J. Basic. Appl. Sci. Res., 3(1)918-926, 2013
© 2013, TextRoad Publication

ISSN 2090-4304
Journal of Basic and Applied Scientific Research
www.textroad.com

Understandability Assessment of Concrete and Parametric Test Cases with Respect to Time and Correctness Ratio Measures

Touseef Tahir[1], Ali Jafar[2], Sobia Zaheer[1], Muhammad Khurram Afzal[1], Munib Ahmad[1] and Jawad Shafi[1]

[1]COMSATS Institute of Information Technology, Department of Computer Sciences, Lahore, Pakistan
[2] Blekinge Institute of Technology, SE 371 79, Karlskrona, Sweden

ABSTRACT

The Parametric (Automated) testing is considered as potential replacement for most of the concrete (manual) testing. Software Understandability is one of the major factors that influence the use of concrete and parametric test cases. It affects the duration (time) and the correctness ratio of software testing process. It is harder to measure understandability as it depends on various factors like human cognitive behavior, coding conventions and code complexity.
This paper presents the extension of "Understandability of Concrete versus Parametric Test Cases". We have analyzed the hypothesis i.e. "parameterized test cases are harder to understand than concrete test cases and thus would lead to overall tests that are harder to understand" with modifications in experimental design and variables. The comparative analysis of the experiments is used to reach a conclusion.

KEYWORDS: Understandability, Automated testing, manual testing, Unit testing; Concrete Testing, Parameterized Testing

I. INTRODUCTION

In [1], we have discussed concrete and parametric test cases in context of understandability. We conducted an experiment to analyze the understandability of concrete and parametric test cases [1].

Software understandability can be defined as the effort required reading and correctly perceiving some pre-define models, documents and source codes [1][2]. It can be defined as the system that is written by one person that is easy to read and understand by another person easily without any resistance. Boehm defined software understandability as a characteristic of software quality which means ease of understanding software systems [4].

Software understandability is the important aspects of software quality because it influences reuse, cost, maintenance and evolution of software [2]. However, software understandability is hard to measure because it depends on the cognitive abilities of human [3]. In order to measure understandability, we need to observe external process which externalizes understandability [3].

During the system maintenance, the need of change may be due to enhancing functions, correcting faults, and adapting to the new environment [2]. If the developers of the original system were absent, then to reuse the system the new developers should understand it before using it [38]. When developers try to read or understand the document developed by another developer, the difficulty of understanding limits the reuse and maintenance of the software documents or programs. If the program under development is difficult to read or understand, the program may cause serious faults [6] [7].

In context of software understandability, programmer should be able to comprehend both code and data [5]. Comprehending the processing logic, data variables and constants helps in recognizing input, output and functionality of the software component [5]. Understandability also depends on the code spatial complexity (if there is greater distance between the definition and use of module, then high effort is required to understand the connectivity), and data spatial complexity (it is easy to understand the use of a function, if the programmer is capable of recognizing all the functions input values, intermediate values, and output values) [5].

Software should be comprehensible for all that includes creator as well as all readers. There are many aspects of understandability some of them are code documentation, application structure and complexity [8][5].

Software is maintained with the combination of source code and documentations. Documentation quality and readability of code should be considered while measuring the understandability of the code [8]. Understandability depends on how the organization of software documents is done. Understandability is the prerequisite for reading documents (comprehension) and for creating documents (specification) [5]. If the source code is easy to read, and understand then a programmer can quickly and accurately obtain the critical information about the software artifacts [5]. Thus, understandability leads to better management of source code and the whole software projects.

***Corresponding Author:** Touseef Tahir, COMSATS Institute of Information Technology, Department of Computer Sciences, Lahore, Pakistan. Email: touseeftahir@ciitlahore.edu.pk

Readability of source code can also be measured by finding percentage of comment lines in total code. A Comment Ratio (CR) is defined as,
CR=LOC/LOM,
Where, LOC represents total lines of code including comments and LOM refers total lines of comments in the code [9].

The symbolic method can be used for estimating the understandability of programs source code. Different programming language and modeling languages uses different symbols (technical or textual symbols). The symbols can be graphical, technical and textual in a unique language [10]. The technical symbol is composed of character-based and graphical based symbols. Graphical symbols are used during modeling methods. Character-based symbols like <, >. ==, <=, >=, //, /* */, if, else, while, int, char, const, struct, #define, etc are used during modeling and programming language [10]. Textual symbols are natural words such as 1, 2, s, d, file, result, ref, bit, to, this, etc. The symbolic categorization helps in reading the textual complexity of the software [10]. Textual symbols are very hard to understand and these symbols are written by developers whereas technical symbols are generated by the software methods and tools [10].

Understandability depends on the perception of the software system as well as comprehension of human resource [3]. The average number of attempts needed for correct reconstruction of single software artifact by many human resources means understandability of the software artifact [3]. Understandability is not only based on the comprehension of software components but also on the methods or frameworks that are used by the programmer.

Understandability is inversely proportional to complexity, if the understandability increases then the complexity of source code decreases and vice-versa [10]. The methods such as cyclomatic complexity, Halsted method, etc are used to measure the complexity of the code and structure of program.

This paper evaluates a hypothesis i.e. “Parameterized test cases are difficult to understand than concrete test cases and thus make software testing difficult”. Section II represents experimental planning. Section III represents experiment operation. Section IV represents analysis of collected data during experiment. Section V represents the comparative analysis of Experiment-1[1] and Experiment-2 Section VI represents conclusions.

II. Experimental Planning

This section provides different dimensions of experimental planning.
2.1 Context:

The purpose for conducting this experiment is to measure the understandability of developers and testers on two different approaches of unit testing [12][13] as shown in figure 1.0.

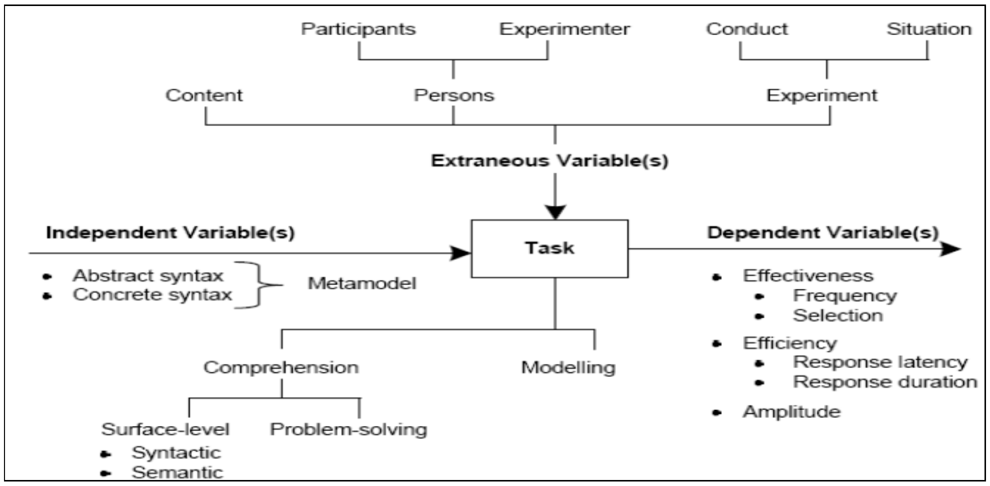


Fig. 1.0 Theoretical Framework for Investigations on Understandability [14]

2.2 Goal:
To evaluate difference of understanding between concrete and parametric test cases with the help of Quantitative analysis of collected data.
2.3 Hypothesis Formulation:
Proposed Hypothesis H0: “Parameterized test cases are difficult to understand than concrete test cases and thus make overall software testing difficult.”
Alternative Hypothesis H1: “Concrete test cases are difficult to understand than Parameterized test cases and thus make overall software testing difficult.”
Measures: Time taken and correctness ratio of given answers by subjects.
2.4 Selection of Variables:
Variable identification is very important for the experiment. Potential variables were identified before the experiment. This helped us to overcome the validity threats for the study [15].

Table 1.0: Variables Declaration

Dependent variables	Understandability, time, accuracy
Independent variables	Number of subjects, programming languages (java), testing frameworks/tools
Treatment variables	Development of concrete test cases, Development of parametric test cases
Extraneous variables	Gender of subjects, Environment, Equipment, overall experiment time, conduct of experiment

2.5 Selection of Subjects:

Total numbers of subjects selected for the experiment were 12. All the subjects were having industrial software engineering experience. The selection criterion for subjects was minimum 1 year of testing/development experience.

2.6 Experimental Design:

Experiment was based on more than one independent variable. Developing codes and test codes are the only factor for experiment. Two treatments concrete test cases and parameterized test cases were applied on the set of codes. Factorial design [11] is used for experiment. Subjects were kept in one group for java on the basis of their experience in a programming language and this is how this experiment is different from one conducted in [1] as we can see in table 1.0 and table 2.0.

Table 2.0: Experiment Information

Subjects	Professional software engineer
Environment	Meeting Room
Instrument	Source code in java and c#, parametric and concrete test cases, Questionnaires based on parametric and concrete unit testing,
Experimental design	Factorial Design Method
Programming languages	Java
Testing tools/ Frameworks	JUnit, JUnit Pex
Groups for experiment	Language groups: Java (12 subjects) Note: All the subjects were provided with both treatments i.e. parametric and concrete test cases.

2.7 Environment and Group Division:

The experiment was conducted in meeting room to avoid any noise disturbing the participants and smooth execution of the experiment.

The subjects for the experiment were arranged in a single group as shown in table 2.0. Each subject in a group was given the source codes (java) and test cases. Experimenters were requested to answer the question carefully for the better result.

2.8 Instrument:

Due to lack of space the source code (java) and test cases (Parametric, Concrete) for only Euclidean algorithm are given in table 3.0 and table 3.1.

Table 3.0: Java Concrete Test Cases for Euclidean Algorithm

Test *methods*

```
public class MainTest {
@Test
public void testEuclid() {
int p = 0; int q = 0; int expResult = 0;
int result = Main.Euclid(p, q);
assertEquals(expResult, result);
                    }
@Test
public void testEuclid1() {
int p = 0; int q = 1; int expResult = 1;
int result = Main.Euclid(p, q);
assertEquals(expResult, result);
                     }
@Test
public void testEuclid2() {
int p = 0; int q = -512; int expResult = -1;
int result = Main.Euclid(p, q);
assertEquals(expResult, result);
                      }
@Test
public void testEuclid3() {
int p = 1039663536; int q = 512; int expResult = 16;
int result = Main.Euclid(p, q);
assertEquals(expResult, result);
                    }
```

Source Code

```
public class Main {
public static int Euclid(int p, int q) {
if (q < 0 || p < 0) return -1;
if (q == 0) return p;
else return Euclid(q, p % q);}
```

Table 3.1: Java Parametric Test Cases for Euclidean Algorithm

Test methods

[TestFixture]
public partial class Class1Test
{
@parameterizedmethod
public int Euclid(int p, int q)
{
int result = Class1.Euclid(p, q);
return result;
}
}
public partial class Class1Test
{
@test
public void testEuclid01()
{
int i; i = this.Euclid(0, 0);
assertEquals(0, i);
}
@test
public void testEuclid02()
{
int i; i = this.Euclid(0, 1);
assertEquals(1, i);
}
@test
public void testEuclid03()
{
int i;
i = this.Euclid(0, int.MinValue);
assertEquals(-1, i);
}
@test
public void testEuclid04()
{
int i;
i = this.Euclid(1039663536, 512);
assertEquals(16, i);
}
}

Source code

public class Class1
{
public static int Euclid(int p, int q)
{
if (q < 0 || p < 0) return -1;
if (q == 0) return p;
else return Euclid(q, p % q);
}

III. EXPERIMENT OPERATION

3.1 Experiment Initialization:

A tutorial was given to the audience before initialization of the experiment. It included guidelines to understand the concepts of concrete and parametric testing and design of experiment. All the information was provided to subjects about the testing frameworks used in the development of test cases both concrete and parameterized. So the general instructions were given to both groups at a same time. The subjects were also informed about special techniques and terms performed in testing frameworks.

3.2 Validity Threats:

The validity threats [15] are given in table 4.0.

Table 4.0: Validity Threats

Validity Threats		
Threat Category	***Threat***	***Precautions***
Internal validity threats	Experience of participants was a major validity threat.	Minimum 1 year development/testing experience.
External Validity Threats	Selection of subjects can affect the overall result.	Only professional software engineers were chosen for the experiment to reduce the carryover effect.
Construct Validity Threat	Chances of Documentation error and Participants can start guessing hypothesis.	Two pilot tests were conducted to avoid documentation error. Hypothesis guessing depends on the human behaviour that how they will appear in different situations. But the affect is minimum or negligible.
Conclusion Validity	Selecting a hypothesis testing method was an issue to come at conclusion.	To overcome this problem the data was initially arranged by the experimenters. The hypothesis testing method was selected according to the nature of experiment by keeping factors, treatments and variables in notice. Hence conclusion validity threats can be minimized

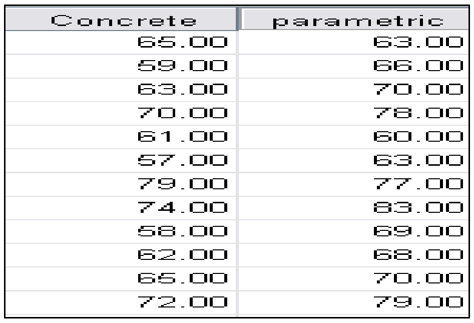
3.3 Experiment Execution:

Experiment was started with the required early instructions. Experiment started according to the information available in table 1.0 and table 2.0. Every participant was asked to write total time against every set of questions on a page particularly.

3.4 Data Arrangements:

The collected data was organized according to section 2.7. To develop test cases for java testing JUnit and Pex tools were used. JUnit was used to develop concrete test cases and JUnit Pex was used to develop parametric test cases. Source code and test cases were for Euclidean, Capitalization and StringTokenizer algorithms. Time was observed in terms of number of minutes. The correctness ratio is measures by using the formula i.e.
Correctness ratio= (Correct answers given*100)/Total number of questions [1].

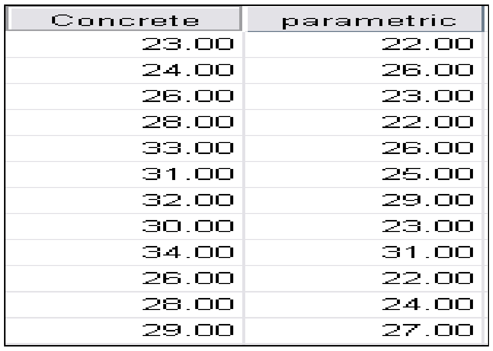
Time: Figure 2.0 shows overall time taken by each individual to solve questionnaires for both concrete and parameterized test cases. The maximum available time for subjects was 90 minutes. The time was average of the time taken by five highly skilled programmers/testers and five inexperienced professionals of industry in Pakistan. During experiment subjects were instructed to write starting time and ending time of every example given to them. Figure 2.0 shows the total time taken by each of 12 subjects in case of parametric and concrete test cases for programs in java.



Concrete	parametric
65.00	63.00
59.00	66.00
63.00	70.00
70.00	78.00
61.00	60.00
57.00	63.00
79.00	77.00
74.00	83.00
58.00	69.00
62.00	68.00
65.00	70.00
72.00	79.00

Fig. 2.0 Time Taken by Subjects

Correctness Ratio: The second selected measure was correctness ratio of the answers given by the participants. There were 37 questions provided to each subject for each type of testing i.e. parametric and concrete. Figure 3.0 shows total correct answers given by every participant.



Concrete	parametric
23.00	22.00
24.00	26.00
26.00	23.00
28.00	22.00
33.00	26.00
31.00	25.00
32.00	29.00
30.00	23.00
34.00	31.00
26.00	22.00
28.00	24.00
29.00	27.00

Fig. 3.0 Correctness Ratio

IV. DATA ANALYSIS FOR EXPERIMENT 2

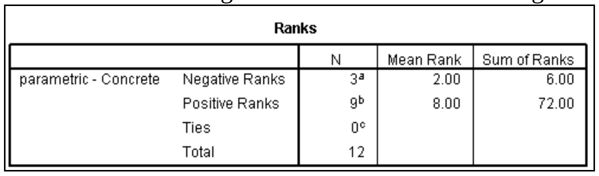
This phase identifies how well the data has been collected and how well it can help us to reach a conclusion. There are various ways of analysing the results [11], as in this case experiment is conducted to accept or refute a hypothesis that is why we have used hypothesis testing method.

4.1 Hypothesis testing

Different techniques are used for testing hypothesis of experiment. This process is done in order to calculate the result for the experiment. There are several hypothesis testing models available to calculate the data. After the early analysis of data it was seen that subjects have taken different amount of time to solve a specific example. In some cases the time difference was huge so it was not fitted for parametric hypothesis testing methods. The data was not in intervals as well so it was decided to use non parametric statistics for the collected data. The only factor involved in experiment was unit testing with two different treatments which were Parameterized test cases and Concrete test cases. The best suitable method for hypothesis testing was Wilcoxon signed test [11][1]. Wilcoxon signed method can be used as a non parametric alternative method to the t-test. The method is used to analyse differences between data taken from the samples. Experimenters have performed Wilcoxon signed both measures i.e. time and correctness ratio of the answers given by participants [11].

Time: Authors did calculations using Wilcoxon singed (WS) test in SPPS (software used for statistical calculations) for the data. Table 6.0 shows the application of WS using time measures provided in figure 2.0.

Table 6.0: Wilcoxon Signed Method for Time Showing Ranks



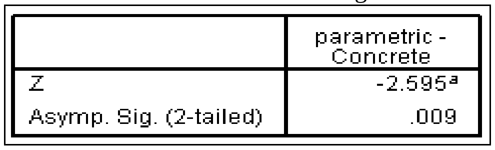
Ranks

		N	Mean Rank	Sum of Ranks
parametric - Concrete	Negative Ranks	3a	2.00	6.00
	Positive Ranks	9b	8.00	72.00
	Ties	0c		
	Total	12		

Where, *a: Parameterize < Concrete,* b: *Parameterize > Concrete, c*: *Concrete = Parameterize.*

The table 6.0 contains 3 values of negative ranks i.e. by subtracting concrete group values from parameterize group values. Sum and mean of negative rank are 6.00 and 2.00 respectively. There are 9 values of positive rank in the data sets. The mean for positive rank values is 8.00 and there sum is 72. There is no tie between the values of both data sets. The table value for the data with “N=12” is 13.

Table 7.0: Test Statistics Wilcoxon Signed Ranks Test



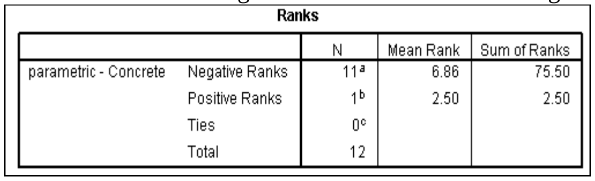
	parametric - Concrete
Z	-2.595a
Asymp. Sig. (2-tailed)	.009

Where, a: *based on Negative ranks.*

The level of significance for calculation is 5% that is 0.05. The obtained P-value is 0.009 after analysis and the calculated Z value obtained from data calculation is -2.595. From the comparison of P+ and P- (sum of Positive ranks and sum of negative ranks) with the table value, we found that positive values were exceeding table value that is 13. Averages of positive ranks were far above than the average of negative ranks,

This shows that participants who were solving parameterized test cases took more time than the participants solving concrete test cases to understand. There was significance difference from the obtained p value analysis with the level of significance because obtained p value was less than 0.05 [16][17]. Thus, it assures us to conclude that “Parameterized test cases are difficult to understand than concrete test cases and thus make overall software testing difficult.”
Correctness ratio: Table 8.0 shows application of WS method using correctness measures provided in figure 3.0.

Table 8.0: WILCOXON Signed Method for Time Showing Ranks



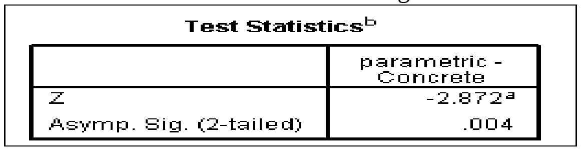
Ranks

		N	Mean Rank	Sum of Ranks
parametric - Concrete	Negative Ranks	11a	6.86	75.50
	Positive Ranks	1b	2.50	2.50
	Ties	0c		
	Total	12		

Where, *a: Parameterize < Concrete,* b: *Parameterize > Concrete, c*: *Concrete = Parameterize.*

There were only 11 values for P- (negative rank) with sum of 75.50 and average of 6.86. The numbers of P+ (positive rank) values were 1 with 2.50 sum and 2.50 mean. There were 12 values in both data sets and for correctness ratio all of them will be considered since there are no ties in the data. Table value for the 2 tailed Wilcoxon will be 13.

Table 9.0: Test Statistics Wilcoxon Signed Ranks Test



Test Statisticsb

	parametric - Concrete
Z	-2.872a
Asymp. Sig. (2-tailed)	.004

Where, a: *based on negative ranks.*

The level of significance for calculation is 5% that is 0.05. The obtained P-value is 0.004 after analysis and the calculated Z value obtained from data calculation is -2.872. From the comparison of P+ and P- (sum of Positive ranks and sum of negative ranks) with the table value, we found that negative rank obtained values are exceeding the table value that is 13. Averages of negative ranks were above than the average of positive ranks, *which shows that participants who were solving parameterized test cases gave less correct answers* than the participants solving concrete test cases. The obtained p value is less than level of significance this means it shows a significance difference between two treatments and we can accept the null hypothesis [18][19].

The analysis of time and correctness ratio enables us conclude that we have to accept null hypothesis.

V. ANALYSIS OF EXPERIMENT (1 AND 2)

We conducted two experiments in order to evaluate the understandability of subjects in concrete and parametric testing. The parameters for calculating understandability were time and correctness ratio. In experiment-1[1] the subjects were divided into two groups [1]. In Experiment-2 subjects were kept into single groups. Only java language was selected to reduce the effect of programming language. The time taken and correctness ration of subjects in both experiments is show in figure 4.0 and figure 5.0 respectively with the help of graphs.

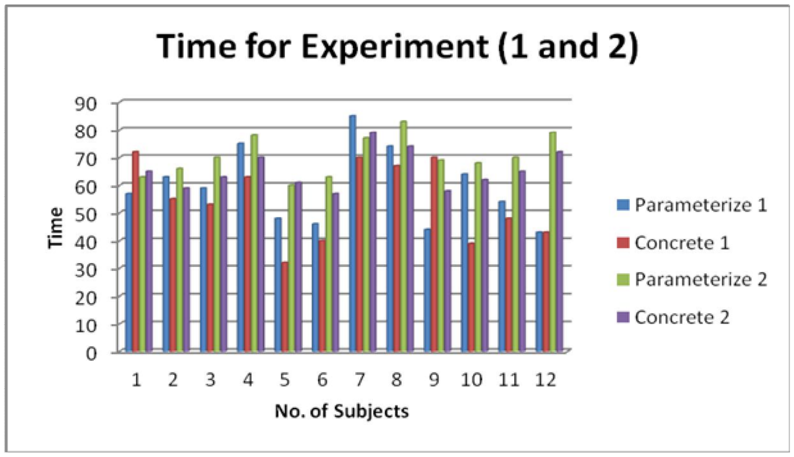


Fig. 4.0 Time taken in both experiments

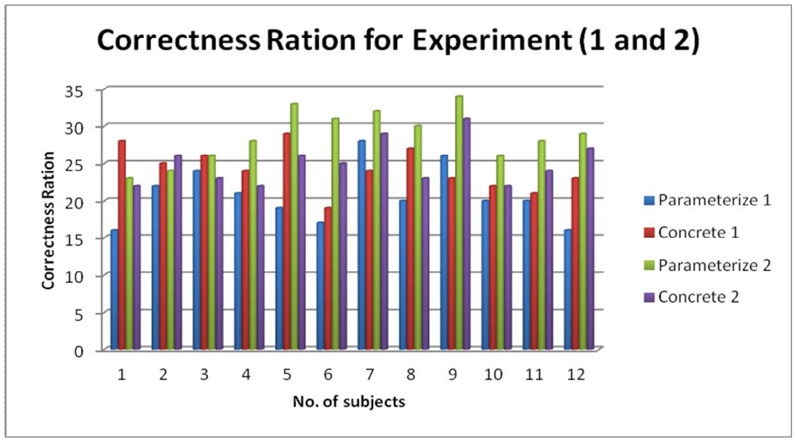


Fig. 5.0 Correctness Ratio in Both Experiments

In order to come at the conclusion experimenters perform hypothesis testing on the experiment data. For both experiments they performed Wilcoxon signed test. Table 10.0 and Table 11.0 show the comparison between the calculated values from both experiments for time taken and correctness ratio respectively.

Table 10.0: Wilcoxon Signed Test for Time Taken in Both Experiments

Attributes	*Experiment 1 (Concrete –parametric)*	*Experiment 2 (Parametric – Concrete)*
Positive Ranks	2^b	9^b
Negative Ranks	9^a	3^a
Tie	1^c	0
Mean Ranks (Positive)	9.25	8.00
Mean Ranks (Negative)	5.28	2.00
SUM of Ranks (Positive)	18.50	72.00
SUM of Ranks (Negative)	47.50	6.00
N (No. of subjects in group)	12	12
Level of significance	5%	5%
Z. value	-1.292^a	-2.595^a
Asymp. Sig. (2 – tailed)	0.196	0.009

Table 10.0: Wilcoxon Signed Test for Correctness Ratio in Both Experiments

Attributes	**Experiment 1 (Concrete – parametric)**	**Experiment 2 (Parametric – Concrete)**
Positive Ranks	10^b	1^b
Negative Ranks	2^a	11^a
Tie	0^c	0^c
Mean Ranks (Positive)	6.40	2.50
Mean Ranks (Negative)	7.00	6.86
SUM of Ranks (Positive)	64.00	2.50
SUM of Ranks (Negative)	14.00	75.50
N (No. of subjects in group)	12	12
Level of Significance	5%	5%
Z. value	-1.968^a	-2.872^a
Asymp. Sig. (2 – tailed)	0.049	0.004

For both experiments the subjects gave more correct answers to the questionnaire based on concrete test cases than parametric test cases. It was clearly shown by the hypothesis test that the calculated value 0.049 was less than 0.05 with z-value -1.968. There was a significant difference between the correctness ratios of answers by the subjects to solve questionnaires when given to same group for Experiment -2. The calculated value 0.004 was clearly less than 0.05 with z-value -2.872.

The both experiments were quite helpful to conclude more understandable types of test cases. In first experiment the time taken by the subjects was approximately same and thus it was not justifiable to conclude on the basis of time taken by subjects to solve the treatment. But for the other factor correctness ratio from the answers given by the subjects, it was clear that subjects found difficulty to answer questionnaire based on parametric test cases than concrete test cases.

In second experiment both treatments (questionnaires based on parametric and concrete test cases) were given to same group. It is clear from the results that subjects have taken more times to solve questionnaire based on parametric test cases with minimum correct answers. Thus, Analysis of both experiments supports us to conclude that *"Parameterized test cases are difficult to understand than concrete test cases and thus make overall software testing difficult".*

VI. CONCLUSION

This paper is an extended evaluation of understandability of concrete and parametric testing. We have used Wilcoxon signed test for both experiments analysis with level of significance as 0.05. The first experiment analysis concluded "*proposed hypothesis H0"* majorly *on the basis of correctness ratio* measure as time measure was not significant.

The analysis of modified Experiment design supported us to again conclude the Proposed hypothesis H0 i.e. "Parameterized test cases are difficult to understand than concrete test cases and thus make overall software testing difficult", *on the basis of Time and Correctness Ratio measures.*

REFERENCES

[1] Tahir, T.; Jafar, A.; Maharajan, M.; , "Understandablity of Concrete versus Parametric Test Cases," Frontiers of Information Technology (FIT), 2011 , pp.212-217, 19-21 Dec. 2011

[2] Goutam Kumar Saha,"Understanding Software Testing Concepts", ACM Ubiquity Vol.9, Issue6, 2008, February 12, 2008 – February 18, 2008.

[3] K. Shima, Y. Takemura, and K. Matsumoto, "An Approach to Evaluation of Software Understandability", Proceedings of the 2002 International Symposium on Empirical Software Engineering (ISESE'02), 2002.

[4] Boehm, B.W., et al, Characteristics of Software Quality, North-Holland, 1978

[5] Jitendra Kumar Chhabra, K.K. Aggarwal, and Yogesh Singh, "Code and data spatial complexity: two important software understandability measures", Information and Software Technology 45 (2003), pp. 539-546, 2003.

[6] Patrice Godefroid, Nils Klarlund, and Koushik Sen, "DART: Directed Automated Random Testing," PLDI'05, Chicago, Illinois, USA, June 12-15, 2005.

[7] Nikolai Tillmann, and Peli de Halleux "Code Digging with Pex Code Understanding and Automatic Testing At Your Fingertips," Microsoft Research, Preliminary Draft, October 21, 2008.

[8] Jin-Cherng Lin, Kuo-Chiang Wu, "A Model for Measuring Software Understandability," Proceedings of The Sixth IEEE International Conference on Computer and Information Technology (CIT'06), 2006.

[9] Krishan K. Aggarwal, Yogesh Singh, and Jitender Kumar Chhabra, "An Integrated Measure of Software Maintainability", Proceedings Annual RELIABILIITY and MAINTAINABILITY Symposium, 2002.

[10] Kari Laitinen, "Estimating Understandabiity of Software Documents", ACM SIGSOFT Software Engineering Notes, vol. 21, no. 4, pp. 81-92, July 1996.

[11] Claes Wohlin, Per Runeson, Martin Höst, Björn Regnell, and Anders Wesslén"Experimentation in Software Engineering: an Introduction".

[12] S. R. Rakitin. Software Verification and Validation for Practitioners and Managers. Artech House Inc. Boston-London, 2001.

[13] "Getting Started With Unit-Testing", An Oracle White Paper, December 2004.

[14] Susanne Patig, "A Practical Guide to Testing the understandability of Notations", Proc. 5th Asia-Pacific conference on conceptual Modeling (APCCM 2008), Wollongong, Australia, vol. 79, pp. 49-58, 2008.

[15] Hakan Erdogmus, Maurizio morisio, and Marco Torchiano, "On the Effectiveness of the Test-First Approach to Programming", IEEE Transactions on software engineering, vol. 31, no. 3, pp. 226-237, March 2005.

[16] Available at: http://www.ats.ucla.edu/stat/stata/whatstat/whatstat.htm [Accessed: June. 27, 2012]

[17] Available at:http://zone.ni.com/reference/en-XX/help/371361D-01/gmath/wilcoxon_signed_rank_test/ [Accessed: June. 27, 2012]

[18] Available at: http://www.ats.ucla.edu/stat/stata/whatstat/whatstat.htm [Accessed: July. 07, 2012]

[19] Available at: http://zone.ni.com/reference/en-XX/help/371361D-01/gmath/wilcoxon_signed_rank_test/ [Accessed: June 07, 2012].