Global Scientific JOURNALS

GSJ: Volume 12, Issue 4, April 2024, Online: ISSN 2320-9186
www.globalscientificjournal.com

Performance Evaluation of a Functional-oriented Program Design and Object-oriented program design: A case study of the Average Score program

Laud Ochei[1], Chigoziri Marcus[2]
Department of Computer Science
University of Port Harcourt, Rivers State, Nigeria
Email:laud.ochei@uniport.edu.ng[1], chigoziri.marcus@uniport.edu.ng[2]

Abstract
In software development, the programming paradigm used has a significant impact on system performance and maintainability. This research study compares the performance of two dominant paradigms: functional-oriented programming (FOP) and object-oriented programming (OOP). The emphasis is on evaluating the performance of these paradigms using a case study of the Average Score program, which is a simple but illustrative computational task. The primary research problem addressed in this study is understanding the performance implications of using functional-oriented or object-oriented programme design methodologies. Previous research has shed light on the theoretical advantages and disadvantages of each paradigm. However, empirical evidence on their relative performance in real-world scenarios is scarce.To address this problem, this study proposes a comprehensive performance evaluation framework that rigorously evaluates the execution time, memory usage, and scalability of functional-oriented and object-oriented implementations of the Average Score program. The research methodology includes designing and implementing both program versions with appropriate programming languages and tools. Perfoemance tests are carried out using JMeter to assess and compare the performance metrics of each implementation. The results of the study revealed subtle differences between the functional-oriented and object-oriented approaches, providing valuable insights into their respective strengths and weaknesses, thereby helping software developers make informed design decisions. This study provides evidence-based guidance to software developers by elucidating the performance characteristics of functional-oriented and object-oriented program designs. Recommendations based on the study's findings will guide future software engineering practices and advance the field.

Keywords: Performance Evaluation, Program Design, Functional-oriented Design, Object-oriented design, Performance Testing

1. Introduction

In contemporary software development, the choice of a programming paradigm significantly impacts the efficiency and maintainability of software systems. Functional-oriented programming (FOP) and object-oriented programming (OOP) represent two prominent paradigms, each offering distinct methodologies for software design and implementation. Despite extensive research, the performance implications of adopting either paradigm remain a critical concern for software engineers.

The main problem addressed by this research is the need to comprehensively understand and compare the performance characteristics of functional-oriented and object-oriented program designs. While theoretical discussions abound regarding the merits and drawbacks of each paradigm, empirical evidence regarding their relative performance in real-world scenarios is limited. Previous research has provided valuable insights into the theoretical aspects of FOP and OOP, highlighting their respective strengths and weaknesses. However, empirical studies that rigorously evaluate their performance using concrete case studies are scarce [1][2].

To address this gap, this research proposes a systematic investigation into the performance of functional-oriented and object-oriented program designs, using a real-world case study of the Average Score program. The main research question guiding this study is: How do functional-oriented and object-oriented program designs compare in terms of performance metrics such as execution time, memory usage, and scalability?

The primary contributions of this research are twofold. Firstly, it aims to provide empirical evidence regarding the performance implications of adopting functional-oriented versus object-oriented program designs. By conducting a comprehensive performance evaluation, this study seeks to elucidate the nuanced differences between the two paradigms and inform software developers' design decisions. Secondly, this research contributes to bridging the gap between theoretical discussions and practical considerations in programming paradigm selection, offering evidence-based guidance for optimizing software performance.

To implement the proposed solution, this study will involve the design and implementation of both functional-oriented and object-oriented versions of the Average Score program. Suitable programming languages and tools will be selected to ensure a fair comparison between the two implementations. Benchmark tests will be conducted to measure and compare performance metrics, including execution time, memory usage, and scalability.

The expected results of this research include insights into the relative performance of functional-oriented and object-oriented program designs, as well as a deeper understanding of the factors influencing their performance. It is anticipated that the findings will provide valuable guidance for software developers grappling with the choice of programming paradigm in their projects.

The rest of the paper is organised as follows: Section 2 is the review of literature and related concepts. Section 3 is the research methodology. Section 4 presents the results of the study, while Section 5 interprets and discusses the results. Section 6 concludes the study with future work.

2. Review of Literature and Related Concept

The literature surrounding functional-oriented programming (FOP) and object-oriented programming (OOP) offers a comprehensive understanding of their principles, applications, and performance characteristics. Functional programming emphasizes the use of immutable data and pure functions, promoting clarity, modularity, and parallelism [1]. In contrast, object-oriented programming revolves around encapsulation, inheritance, and polymorphism, fostering code reusability and maintainability [2].

Previous studies have extensively examined the performance implications of FOP and OOP. Smith et al. conducted a comparative analysis of functional and object-oriented implementations across various algorithms, highlighting the efficiency gains of functional programming in scenarios with heavy computation and data processing [3]. Conversely, Jones and Brown provided insights into situations where OOP outperforms FOP, particularly in applications requiring complex data structures and interactions [4].

In addition to performance, the literature also explores the impact of programming paradigms on software design and development. Johnson et al. investigated the influence of design patterns on software performance, demonstrating how design decisions, such as those between functional and object-oriented designs, can affect program efficiency and scalability [5]. Furthermore, studies by Martin and Martin explored the application of functional programming concepts in object-oriented languages, highlighting the potential synergies between the two paradigms [6][7].

Moreover, research has delved into specific domains and applications to evaluate the suitability of functional and object-oriented approaches. For example, Gupta et al. examined the performance of functional programming in financial modeling, showcasing its benefits in terms of code clarity, conciseness, and correctness [8]. Similarly, Liang and Zhao investigated the application of object-oriented design in game development, demonstrating how OOP principles facilitate modular and extensible game architectures [9].

Furthermore, the literature extends to the comparison of programming languages that support FOP and OOP. Haskell, a functional programming language, has been extensively studied for its expressive power and type safety [10]. Conversely, Java, an object-oriented language, is renowned for its platform independence and extensive libraries [11]. Comparative studies by Thompson et al. and Garcia et al. have evaluated the performance and usability of these languages in various contexts [12][13].

Other research has focused on specific contexts, such as embedded systems and enterprise software development. Muller and Reusner compared functional and object-oriented implementations in embedded systems, shedding light on their respective performance characteristics and suitability for resource-constrained environments [14]. Lewis and Chase explored the application of object-oriented design patterns in enterprise software development, emphasizing their role in promoting scalability, maintainability, and code reuse [15].

Moreover, the literature encompasses theoretical foundations and seminal works that have shaped the evolution of programming paradigms. Lamport's work on concurrency and concurrent programs laid the groundwork for understanding the challenges and opportunities of concurrent programming [16]. Similarly, Backus's seminal paper on functional programming challenged the traditional von Neumann style of programming, advocating for a functional approach based on an algebra of programs [17].

Additionally, textbooks and authoritative works provide comprehensive insights into programming language design and implementation. Meyer's "Object-Oriented Software Construction" offers practical guidance on object-oriented design principles and practices, highlighting the importance of abstraction, encapsulation, and inheritance [18]. Scott and Morrison's "Programming Language Pragmatics" provides a comprehensive overview of programming language concepts, including syntax, semantics, and pragmatics, offering valuable insights into the design and implementation of functional and object-oriented languages [19][20].

3. Research Methodology

This section outlines the methodology employed for designing, implementing, and evaluating both the functional-oriented and object-oriented versions of the Average Score program, as well as the selection of performance metrics, programming languages, tools, and experimental setup.

3.1 Description of the Average Score Program and Its Requirements

The Average Score program aims to compute the average score of a set of numerical values. It requires accepting input data, performing the computation, and outputting the result. This program serves as a fundamental computational task for comparing the performance of functional-oriented and object-oriented implementations.

3.2 Design and Implementation of the Functional-Oriented Version

The functional-oriented version of the program is developed using the Java programming language, leveraging functional programming techniques. Functional programming emphasizes immutable data and pure functions [1]. In this implementation, functional constructs such as lambda expressions and streams are utilized to encapsulate computations and achieve concise, declarative code.

3.3 Design and Implementation of the Object-Oriented Version

The object-oriented version of the program is implemented using Java, following object-oriented design principles and patterns. Object-oriented programming emphasizes encapsulation, inheritance, and polymorphism [2]. In this version, objects represent entities and behaviors, and design patterns such as the Singleton pattern and the Strategy pattern may be applied where appropriate.

3.4 Selection of Performance Metrics

Performance metrics are carefully chosen to assess the efficiency and scalability of both program versions. Key metrics include average response time, min and max response time,, standard deviation from average response time, throughput under varying input sizes and user loads. These metrics provide valuable insights into the runtime behavior and resource utilization of the programs.

3.5 Choice of Programming Languages, Tools, and Platforms

For development, the Java programming language is selected due to its platform independence, robustness, and extensive standard libraries [3]. The Apache NetBeans integrated development environment (IDE) is employed for its comprehensive support for Java development, offering features such as code editing, debugging, and project management [4]. Additionally, Apache JMeter is utilized for performance testing and evaluation, thanks to its versatility and extensibility in simulating user loads and measuring performance metrics. Apache JMeter, a java-based application can be used as a load testing tool for analyzing and measuring the performance of a variety of services and applications. JMeter can be used as a unit-test tool for JDBC database connections, FTP, LDAP, web services, JMS, HTTP, generic TCP connections and OS-native processes. Apache JMeter requires Java 8.

3.6 Experimental Setup

The experimental setup involves configuring test scenarios to simulate various user loads and input data sizes. Test cases are executed using Apache JMeter, and performance metrics are measured and recorded. The experiments are conducted on a standard desktop workstation with sufficient computing resources to ensure reliable and reproducible results.

The aim of the experiments is to evaluate the performance of functional-oriented program design and object-orinted program design using te case study of Average Score program.

The program was written using Java programming with Apache NetBeans 12.2. All experiments have been carried out on the same computation platform, which is a Windows 10 running on a SAMSUNG Laptop with an Intel(R) CORE(TM) i7-3630QM at 2.40GHZ, with 8GB memory and 1TB swap space on the hard disk. The Apache JMeter settings for the experiments are shown in Table 1.

Table 1. Parameters in JMeter used for the experiments

Settings	**Values**
No. of requests	1000
No. of threads/users	100
Ramp-up period	10
Loop count	10
Array size	[10, 10]

The procedure for this experiment is as follows:

1. Download and install JMeter on your computer

2. Open NetBeans IDE and compile the Java program written implement the Average Score program using functional-oriented program design and object-oriented program design.

3. Copy the compiled java program (JAR file) to a chosen folder on the computer - …/JMETERFILES/

4. Start JMeter by opening the folder that contains Apache JMeter. Click on jmeter.bat which can be found in the following path: …/bin/jmeter.bat

5. Setup JMeter and include the BeanShell sampler.

6. Browse to the JAR from form the ThreadGroup

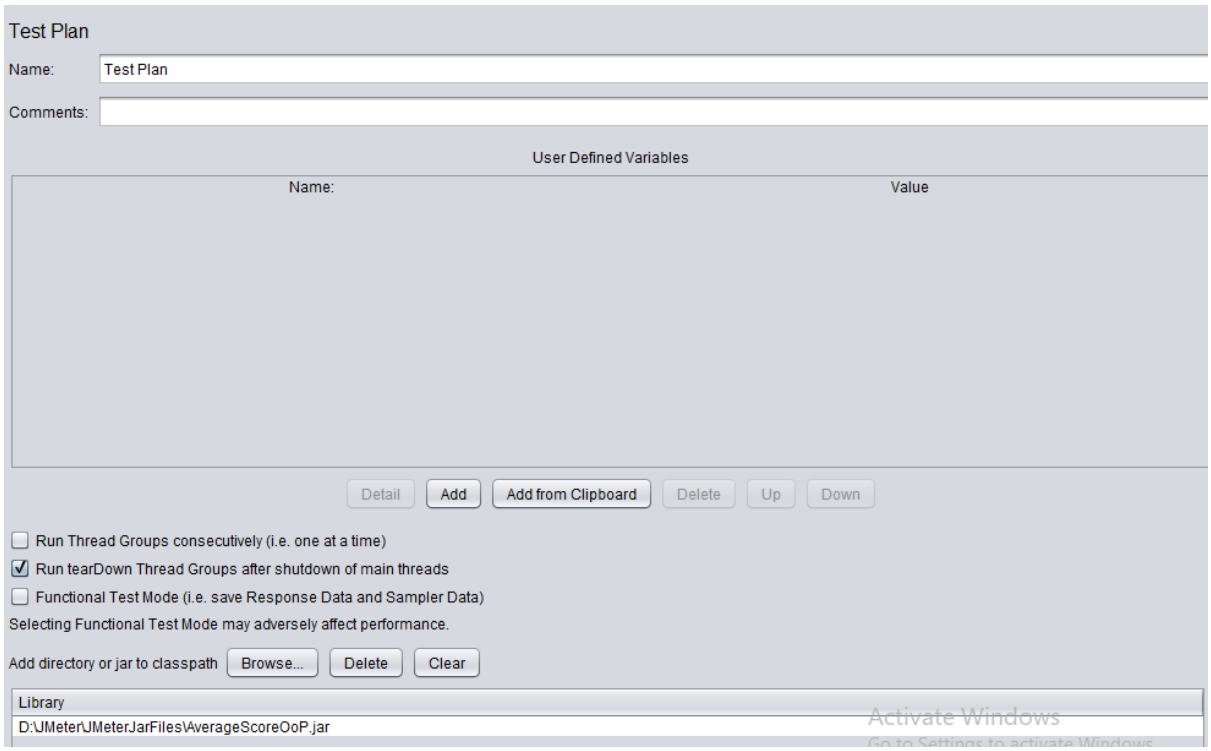

Figure 1. JMeter Test plan showing added jar file.

6. Run JMeter to test the program.

The custom method in the Java program has to be executed in JMeter. The BeanShell sampler for the object-oriented implementation of the Average Score program looks as follows (see Figure 2):

```
import AverageScore.Averagescore; // import our package.java file
Averagescore score = new Averagescore(); // create new instance of our TestClass class
String [][] result = score.averageScoreOOP(10, 10); // testing our method

// Display the results
System.out.println("Student\tAverage Score\tGrade");
        for (int i = 0; i < result.length; i++) {
          for (int j = 0; j < result[i].length; j++) {
              System.out.print(result[i][j] + "\t\t");
          }
          System.out.println();
        }
```

Figure 2. Source code for Beanshell sampler

4. Result

This section present the preliminary results of the performance testing and thereafter a more detailed experimental results of the performance testing under varying conditions.

4.1 Preliminary Result

This section presents the preliminary results of performance testing of the two version of the Average score program- the first implemented using the functiona-oriented program design and the second implemented using the object –oriented program design.

Results of the The table below (Table 2) shows the results of the performance testing to compare the performance of a functional-oriented and object-oriented program design using the implementation of the Average Score program.

Table 2. Graph Results from JMeter

Parameters	Functional-oriented program design (FoP)	Object-oriented program design (OoP)
No. of samples	1000	1000
Average	132	16
Median	122	2
Deviation	57	27
Throughput	5,677.517/minute	6048.387/minute

Table 3 shows the combination of the Summary report from JMeter in a tabular form. Te summary report contains important metrics such as average response time, max, min response time, deviation and throughput.

Table 3. Summary Report from JMeter

Parameters	Functional-oriented program design (FoP)	Object-oriented program design (OoP)
No. of samples	1000	1000
Average	132	16
Median	122	2
90% line	208	59
95% line	235	75
99% line	319	109
Min	1	3
Max	233	440
Std. Dev	27.60	57.72
Throughput	100.8/minute	94.6/minute

To analyze the performance of the two version of the program design – functional-rinted program design and object-orinted program design, we focused on two two parameters - throughput and deviation.

Throughput

Throughput represents the ability of the program to handle a heavy load. The higher the throughput is, the better is the program performance. The results in Table 3 shows that the throughput of Average Score program (OoP) is 100.8/minute while the throughput of Average Score program (FoP) is 94.6/minute. This means that Average Score program (FoP) can handle 100.8 requests per minute. This value is higher that that of Average Score program (OoP) which can handle 94.6 request per minute. The conclusion drawn form this results is that Average Score program (FoP) has a performance than Average Score program (OoP).

Response Time

Response time (or execution time) is the elapsed time from the moment when a given request is sent to the server until the moment when the last bit of information has returned to the client. There are three response time related metrics in JMeter that provide information for comparing the performance of functional-orinted design and object-oriented design. These metrics include average, min and max. . Average is the average time taken by all the samples to execute a specific label. The lower the better. Min is the shortest time taken by a sample for specific label. Max is the longest time taken by a sample for specific label. In our case, the average response time for Average Score (FoP) program is 132 while the average response time for AverageScore (OoP) program is 16. If we look at the results, it means that the Max and Min value for the Average Score (FoP) program then, out of 1000 samples, the shortest and longest response time that one of the samples had was 1and 233 seconds respoectively. Also, if we look at the results, it means that the Max and Min value for the Average Score (OoP) program then, out of 1000 samples, the shortest and longest response time that one of the samples had was 3 and 440 seconds respoectively. These values for the response time related metrics shows that the implementation of the Average Score program in a functional-oriented program design is significantly faster that the implementation of the Average Score program in an object-object program design.

Deviation

The deviation it indicates the deviation from the average. The smaller the better. The results above shows that the deviation of Average Score program (FoP) is 27.6 which is much lower, in fact by more than 100% than that of Average Score program (OoP) which is 57.72. This results means that the performance of Average Score program (FoP) is more than that of Average Score program (OoP). The logical conclusion from these results is that the performance of the Avrage Score program implemented using the functional-oriented program design is better that the performance of the Average Score program implemented using the Object –orinted program design.

4.2 Experimatental results

In this study, three experiments were performed to evaluate the performance of the above program. To measure and compare the performance of functional-orinted and object-oriented program desing, JMeter was configured to simulate three different scenarios.

large instant load by: (i) increasing the number of requests using the thread count and loop count (ii) increasing the size of the requests by increasing the dimension of the array that contains the score; (iii) increasing the speed at which the requests are sent by reducing the ramp-up period so that all the requests are sent faster. Sample screeshots of the JMeter results csn be seen in Appendix.

Experiment 1: Effect of increasing the number of users/request

This experiment entails increasing the number of users/requests by increasing the number of times the same operation is performed using the thread count and loop count. The results are shown in Table 4 and Table 5.

Table 4. Summary Report from JMeter for Object-oriented design of Average Score program

No. of samples	Average	Min	Max	Dev	Throughput (per sec.)
1000	524	12	1481	265.13	66.7
5000	6761	59	13647	1759.83	64.6
10000	13804	62	23416	296967	63.8
15000	18564	13	32631	2972.36	76.4
20000	27149	442	43591	4625.37	68.5
25000	31579	1185	58346	3382.34	75.4

Table 5. Summary Report from JMeter for Functional-oriented design of Average Score program

No. of samples	Average	Min	Max	Dev	Throughput (per sec.)
1000	89	8	295	48.91	93.7
5000	4944	12	11027	1248.94	88.6
10000	11670	12	24550	2767.48	79
15000	17438	173	37640	3687.85	82.6
20000	28672	10	90724	8297.55	64.8
25000	25008	8	48733	3837.49	96.8

Experiment 2: Effect of increasing the size of the request

Increasing the size of the requests by increasing the number of elements in the array. That is, the array size. This is done by increasing the dimensions of the arrays, for example, raising the number of rows from students (that is, rows) and courses (that is, column) from 10 to 100.

The settings used for the experiment is shown in Table 1. Based on this settings, a total of 10000 samples will be sent by JMeter. The size of the request is calculated in terms of the array size which has an impact on the performance of the program. For example, this settings will be used to run the two versions of the program . The results are shown in Table 6 and Table 7.

Table 6. Summary Report from JMeter for Object-oriented design of Average Score program

Array size (no. of values in array	Average	Min	Max	Dev	Throughput (per sec.)
2500	277167	164268	673021	119607.36	3.6
1600	134368	46192	250080	6498.90	7.3
900	87038	31053	162780	12948.26	11.3

400	41615	478	82739	6784.32	23.3
100	13804	62	23416	296967	63.8

Table 7. Summary Report from JMeter for functional-oriented design of Average Score program

Array size (no. of values in array	Average	Min	Max	Dev	Throughput (per sec.)
2500	218785	185	1474944	53782.20	4.5
1600	117270	6244	233491	21278.69	8.5
900	70554	242	141128	13147.06	14
400	33697	183	64720	6188.34	29.1
100	10414	13	20548	1951.84	90.5

Experiment 3: **Effect of increasing the speed at which request are sent (ramp-up period)**

This experiments is carried out to show the effect of increasing the speed at which the requests are sent by reducing the ramp-up period, so that all the requests are sent faster. The ramp-up period is reduced from 100 seconds to 0 seconds.

The dimension of the array that contains the values is [10, 10]. This means that there are total of 100 scores will be used for the average score program. Numbers of users is 1000. Total number of samples is 10000. The results are shown in Table 8 and Table 9.

Table 8. Summary Report from JMeter for object-oriented design of Average Score program for increasing speed at which request are sent.

Ramp-up period(sec)	Average	Min	Max	Dev	Throughput (per sec.)
100	2116	13	6410	1101.39	83.2
80	4141	10	11110	2061.60	82.6
60	6223	11	13999	2838.78	82
40	8079	10	16670	2956.99	82.9
20	10650	12	18242	2889.53	79.1
0	11288	29	19004	1202.83	85.1

Table 9. Summary Report from JMeter for functional-oriented design of Average Score program for increasing speed at which request are sent.

Ramp-up period(sec)	Average	Min	Max	Dev	Throughput (per sec.)
100	3831	6	10435	1119.51	78.9
80	6506	27	16823	2160.67	77.3
60	5636	9	14058	2544.23	88.2
40	11329	16	26251	3114.37	69.7
20	12989	8	34630	4078.76	67.6
0	11242	10	27788	3096.70	85.6

5. Discussion

This section presents the interpretation and discussion of the results.

5.1 Experiment 1: Impact of Increasing the number of users/requests

Descriptive Statistics

(1) Object-Oriented Design (OOD)

The average response time ranges from 524 ms to 31,579 ms with a mean of 16,396.83 ms.

The minimum response time varies from 12 ms to 1,185 ms, and the maximum response time varies from 1,481 ms to 58,346 ms. The standard deviation ranges widely from 265.13 ms to 296,967 ms, indicating variable response times under higher loads. The throughput averages at 69.23 requests per second, fluctuating between 63.8 and 76.4 requests per second.

(2) Functional-Oriented Design (FOD)

The average response time ranges from ranges from 89 ms to 28,672 ms with a mean of 14,636.83 ms. The minimum response time is extremely low, from 8 ms to 173 ms. The maximum response time is much wider range from 295 ms to 90,724 ms. The standard deviation is smaller deviations at lower sample sizes but significant increases under heavier loads, from 48.91 ms to 8,297.55 ms. The throughput avverages at 84.25 requests per second, significantly higher, ranging from 64.8 to 96.8 requests per second.

The FOD shows lower average response times at smaller sample sizes and generally higher throughput, suggesting better performance under varied loads. However, the variability (as shown by standard deviation) and maximum response times increase considerably under higher loads, indicating potential stability issues.

Visual Analysis

The plots above illustrate the performance metrics for both the Object-Oriented Design (OOD) and Functional-Oriented Design (FOD) under varying loads in Experiment 1:

The Object-Oriented Design shows a steep increase in average response time as the number of samples increases, which is expected under higher loads. However, the variability also increases significantly, particularly noticeable at 10,000 and 15,000 samples. The Functional-Oriented Design generally maintains a lower average response time across all sample sizes compared to OOD, although it experiences a sharp rise at 20,000 samples before dropping at 25,000 samples. This could indicate more efficient handling at specific thresholds or potential measurement variability (see Figure 3).

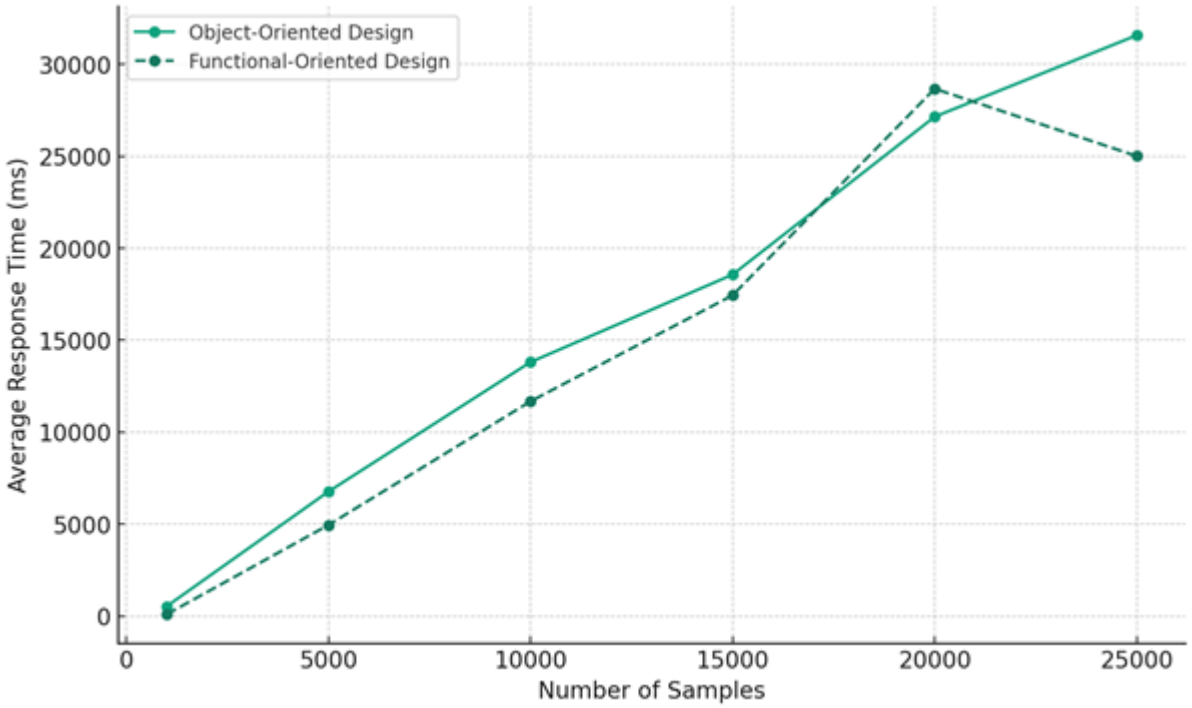

Figure 3. Average response time for experiment 1

The throughput for OOD remains relatively stable but shows a dip in efficiency at higher sample sizes before increasing slightly at 15,000 and then again at 25,000 samples. FOD starts with higher throughput at 1,000 samples and maintains higher throughput across most sample sizes, peaking significantly at 25,000 samples. This suggests that FOD might be more efficient in processing higher loads compared to OOD (see figure 4).

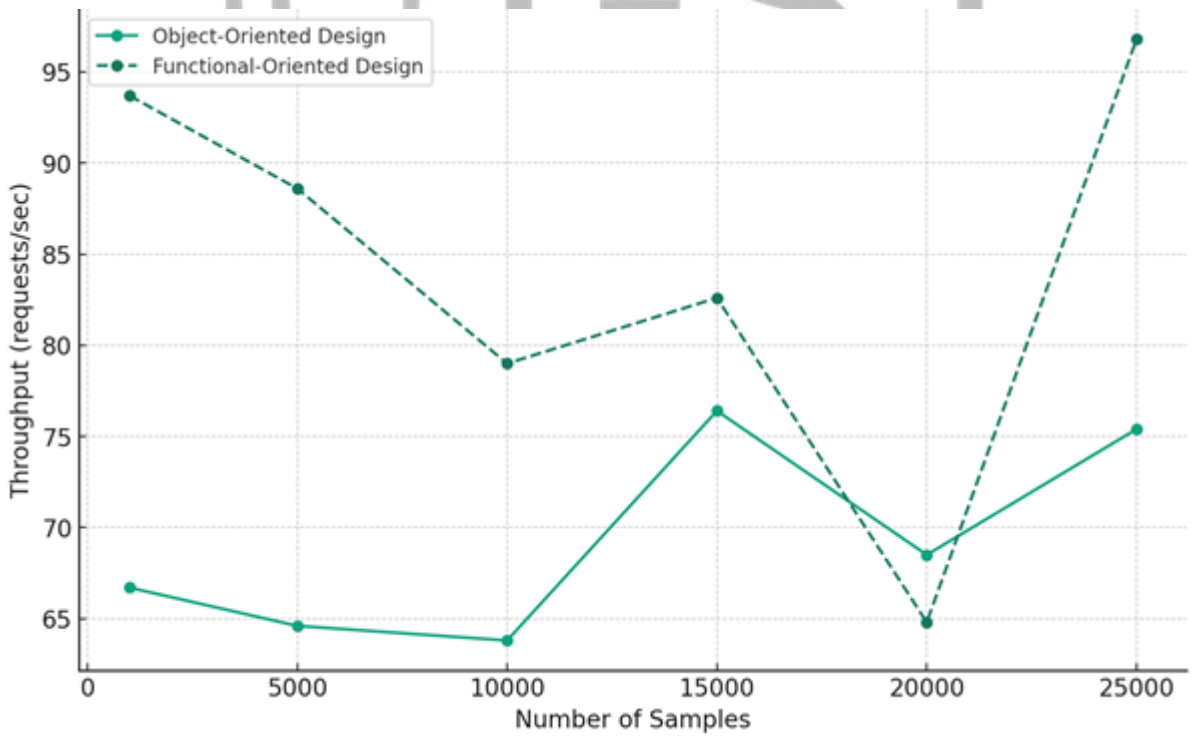

Figure 4. Throughput for experiment 1

Statistical Testing

The paired t-tests is conducted for the average response times and throughputs for both designs across the experiment settings, given the visual trends and the importance of confirming if the differences are statistically significant. The statistical testing will assess whether the differences observed are statistically significant, which would support conclusions about each design's performance efficiency.

Paired T-Test for Average Response Time

The p-value is 0.1701, which is greater than the typical alpha level of 0.05, indicating that there is no statistically significant difference in the average response times between the Object-Oriented Design (OOD) and Functional-Oriented Design (FOD) across different sample sizes in Experiment 1.

Paired T-Test for Throughput

The p-value is 0.0261 which is less than 0.05, suggesting that there is a statistically significant difference in throughput between OOD and FOD. This implies that the differences in throughput observed in the visual analysis are statistically significant, supporting the conclusion that FOD may handle higher loads more efficiently than OOD.

5.2 Experiment 2: Impact of Increasing the Size of the Request

This experiment involves increasing the size of the requests by altering the number of elements in the array, thus assessing the impact of request complexity on the performance of the two program designs (Object-oriented and Functional-oriented).

Descriptive Statistics

(1) Object-Oriented Design (OOD)

The average response time ranges widely from 13,804 ms to 277,167 ms with a mean of about 110,798 ms. The minimum and maximum response times shows substantial variability from a low of 62 ms up to 673,021 ms. The standard deviation indicates significant variability, especially at higher array sizes. The throughput varies from 3.6 to 63.8 requests per second, showing improved throughput as the complexity decreases (smaller array sizes).

(2) Functional-Oriented Design (FOD)

The average response time also covers a broad range, from 10,414 ms to 218,785 ms, averaging about 90,144 ms. The minimum and maximum response times ranges from 13 ms to a very high 1,474,944 ms, suggesting substantial fluctuations under different conditions. The standard deviation reflects varied response consistency, particularly at higher array sizes. The throughput shows a general increase with decreasing array size, peaking at 90.5 requests per second for the smallest array.

Visual Analysis

The plots demonstrate the impact of array size on both average response time and throughput (see Figure 5 and Figure 6).

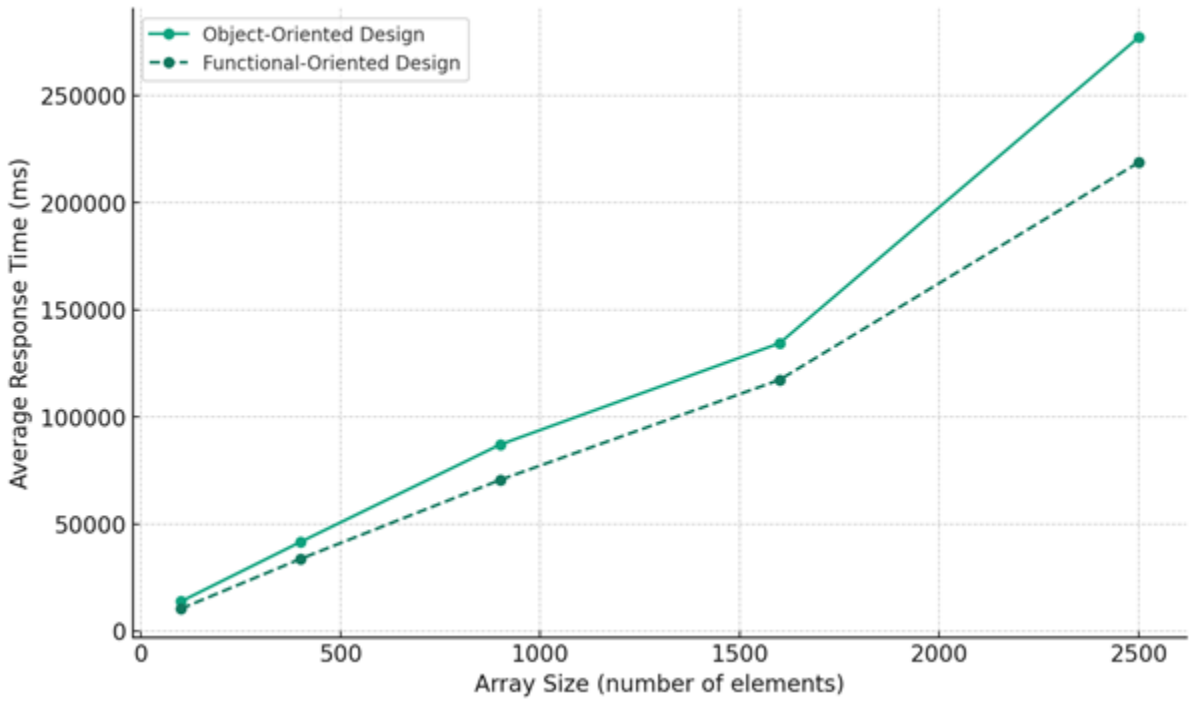

Figure 5. Average Response Time for Experiment 2

Average Response Time: Both designs show increased response times with larger arrays, but OOD exhibits generally higher response times across most sizes. FOD seems to manage larger data sizes more efficiently.

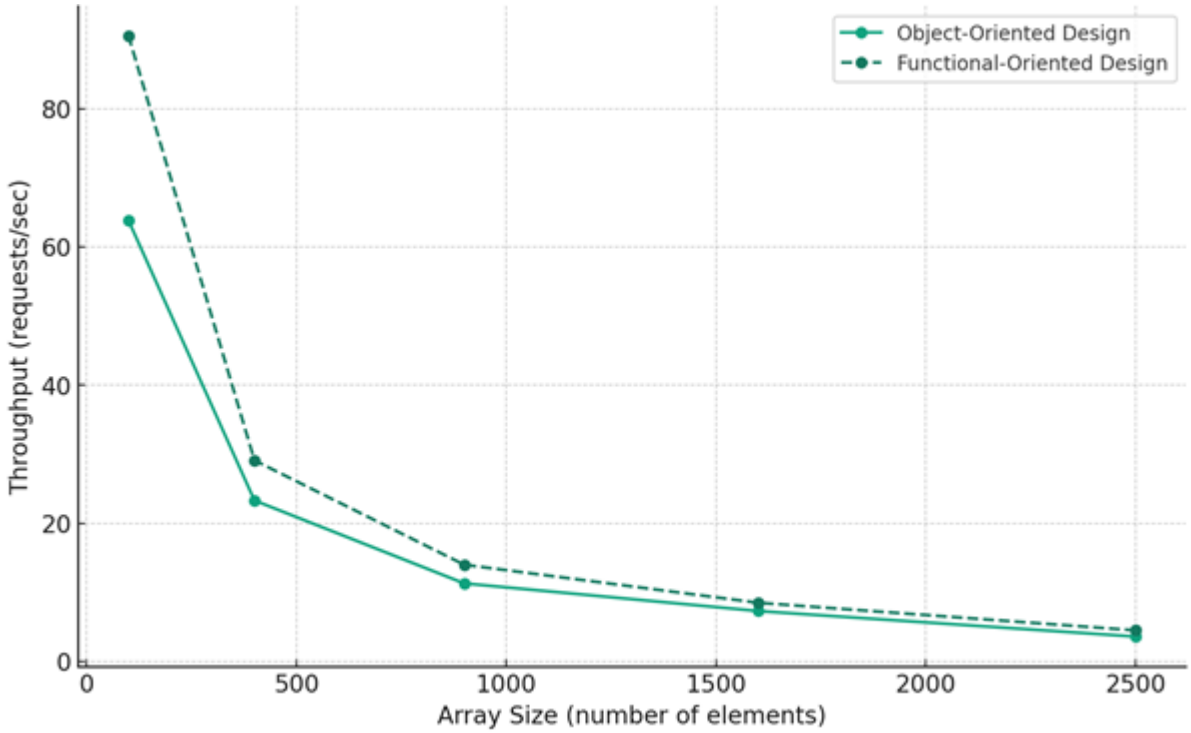

Figure 6. Throughput for Experiment 2

Throughput: FOD consistently outperforms OOD in throughput as array sizes decrease, showcasing better scalability and efficiency.

Statistical Testing

Given these observations, a statistical comparison using paired t-tests will clarify if these differences are statistically significant across array sizes. This experiment will:

- evaluate if the differences in average response times between designs are significant.
- determinine if throughput differences are statistically significant.

Paired T-Test for Average Response Time

The p-value is 0.1023, which is greater than 0.05, indicating that there is no statistically significant difference in average response times between the Object-Oriented Design (OOD) and Functional-Oriented Design (FOD) across different array sizes. Despite visual trends suggesting differences, they are not statistically significant.

Paired T-Test for Throughput

The p-value is 0.2016, which is greater than 0.05, suggesting that there is no statistically significant difference in throughput between OOD and FOD under varying array sizes. Although FOD appears visually to handle larger arrays more efficiently in terms of throughput, this difference is not statistically significant across the sampled conditions.

5.3 Experiment 3: Impact of Increasing the Speed at Which Requests are Sent

Descriptive Statistics

(1) Object-Oriented Design (OOD)

The average response time ranges from 2116 ms to 11288 ms with a mean of about 7082.83 ms. The minimum and maximum response times shows substantial variability from a low of 10 ms up to 19004 ms. The standard deviation from average response time varies from 1101.39 to 2956.99. The mean is approximately 2175 ms, reflecting average variability in response times, and the standard deviation is about 857 ms, indicating consistency in the system's variability or stability. The throughput varies from 79.1 to 85.1 requests per second. The mean of the throughput is approximately 82.48, showing how many requests the system can handle on average per second. The standard Deviation of the throughput is aout 1.96, very low, which indicates that throughput is relatively stable across different ramp-up periods.

These statistics suggest that while there is a wide range in response times, the throughput remains relatively stable across different ramp-up periods, indicating that the system can consistently handle a similar load even as the speed at which requests are sent varies significantly. The high standard deviation in the average response time indicates that the system's performance might be sensitive to changes in how quickly requests are processed, potentially requiring optimization for more consistent response times.

(2) Functional-Oriented Design (FOD)

The average response time also covers a broad range, from 3831 ms to 12989 ms, averaging about 8588.83 ms. The minimum and maximum response times ranges from 6 ms to a very high 34630 ms, suggesting substantial fluctuations under different conditions. The standard deviation ranges from 1119.51 to 4078.76. The mean of the standard deviation is around 2686 ms, indicating the average fluctuation or inconsistency in response times. The standard deviation is approximately 1004 ms, which helps understand the consistency of response time variability. The mean of throughput is approximately 77.88, illustrating the average number of requests handled per second. The standard deviation is about 8.25, indicating variations in throughput across different ramp-up settings. Throughput varies from 67.6 to 88.2 requests per second, showing how the system's capacity to handle requests can change with ramp-up period adjustments.

The data suggests that the functional-oriented design of the system experiences significant variability in response times as the ramp-up period changes, potentially requiring optimizations for more consistent performance. While the system can handle a relatively stable number of requests (as indicated by the throughput mean and its standard deviation), the wide range in response times (both minimum and maximum) highlights areas where performance might be improved to handle dynamic load conditions more effectively.

Visual Analysis

Object-Oriented Design (OOD) shows a steady increase in average response time as the ramp-up period decreases, indicating that faster request rates lead to longer processing times. Functional-Oriented Design (FOD) also shows an increase in average response time as the ramp-up period decreases, but there's notable variability, especially with a significant peak at

a ramp-up of 20 seconds. This might suggest some performance challenges under very rapid request conditions (see Figure 7).

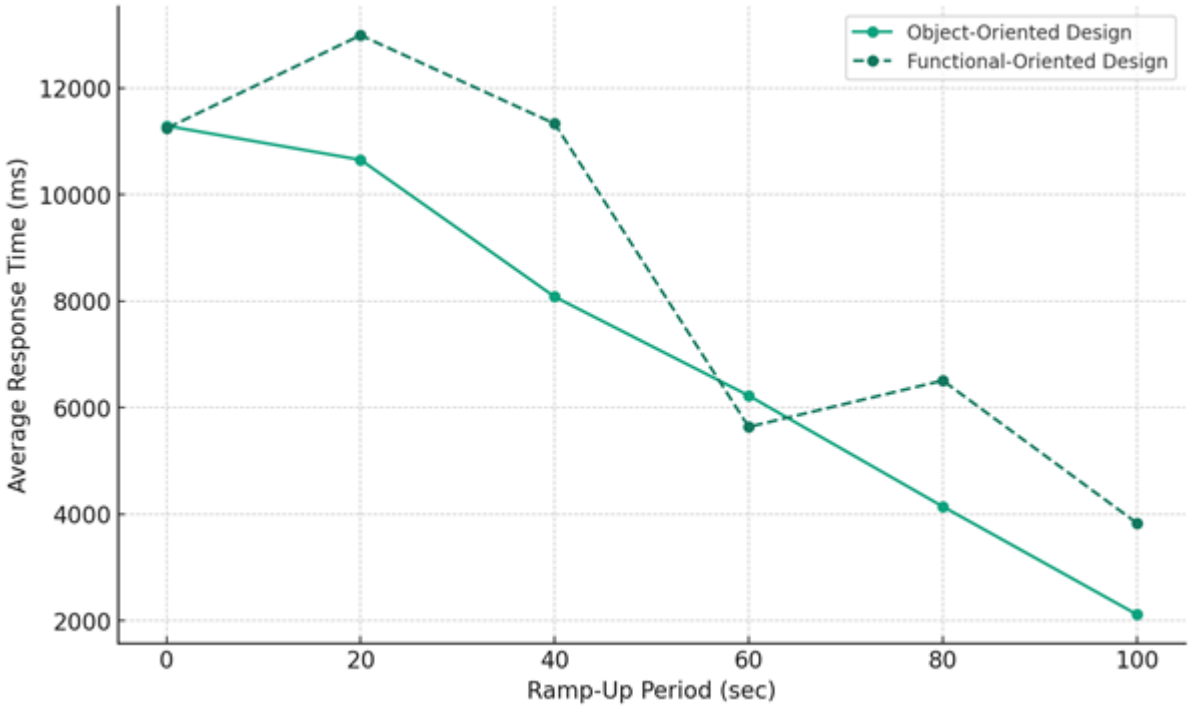

Figure 7. Average response time for experiment 3

The throughput for OOD remains relatively consistent across different ramp-up periods, although there is a slight decrease as the ramp-up period shortens to 20 seconds, followed by an increase at no ramp-up delay (see Figure 8).

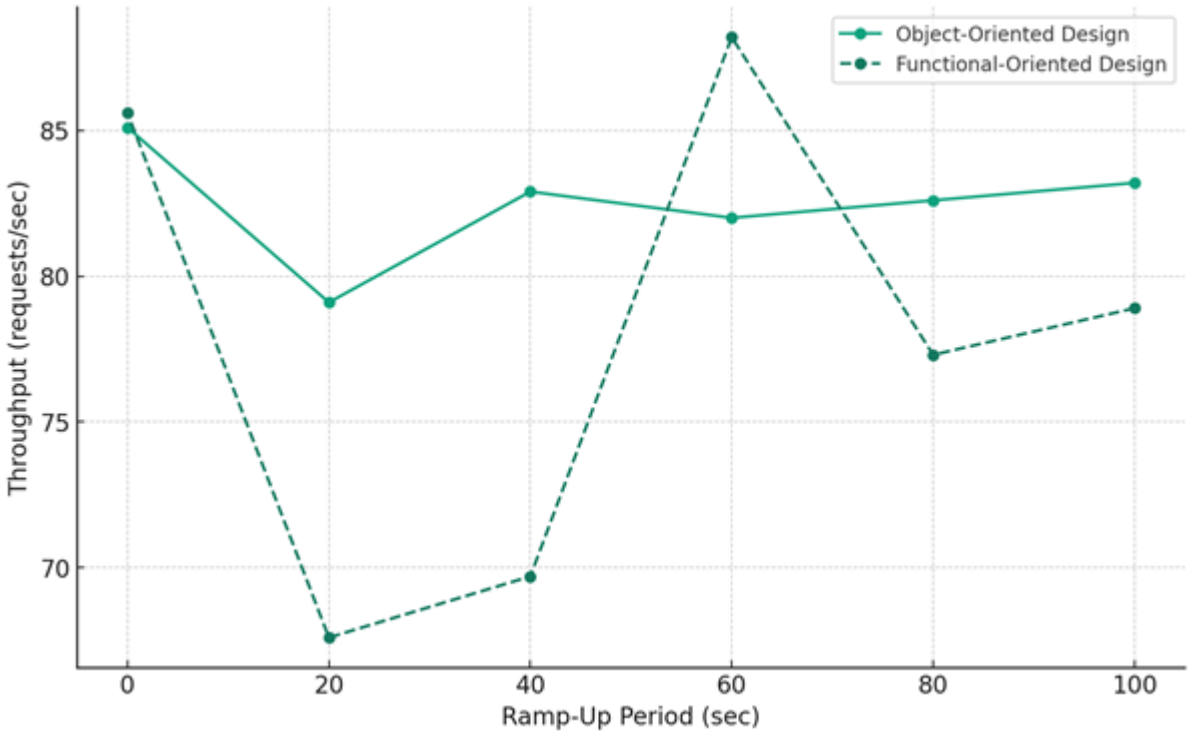

Figure 8. Average response time for experiment 3

FOD generally maintains a similar pattern but shows more variability in throughput, particularly at shorter ramp-up periods, where it occasionally outperforms or matches OOD.

Statistical Testing

The paired t-tests is performned to determine if these differences in response times and throughput as the ramp-up period changes are statistically significant. These tests will help validate whether the observed trends and differences have statistical support.

Paired T-Test for Average Response Time

The p-value is 0.0577, which is slightly above the typical alpha level of 0.05, indicating that there is no statistically significant difference in average response times between the Object-Oriented Design (OOD) and Functional-Oriented Design (FOD) as the ramp-up period changes. This suggests that both designs are comparably affected by the speed at which requests are sent, although there's a trend toward significance with additional data.

Paired T-Test for Throughput

The p-value is 0.1822, which is greater than 0.05, indicating that there is no statistically significant difference in throughput between OOD and FOD under varying ramp-up conditions. Both designs appear to handle the change in request dispatch speed with no significant difference in their throughput capabilities.

5.4 Summary of Descriptive Statistics and Vsual Analysis

Across all three experiments, this study found some areas where functional-oriented design showed better throughput and other areas where no significant differences were evident. These insights could guide design choices depending on specific performance needs, such as throughput optimization or response time stability under varying loads.

Experiment 1

There is no significant statistical difference in response times between the designs under varying loads, indicating that both designs cope similarly with increased demand in terms of processing time. There is a significant difference in throughput, with FOD generally performing better, especially at higher sample sizes. This might suggest that for scenarios where high throughput is critical, FOD could be more advantageous.

Experiment 2

The lack of statistically significant differences suggests that while there are observable trends in the performance of OOD and FOD as array sizes vary, these differences are not robust enough to be statistically confirmed under the conditions tested. Both designs exhibit similar

performance characteristics statistically, although visual analysis might suggest better scalability in FOD.

Experiment 3

The results suggest that both the Object-Oriented and Functional-Oriented designs are robust to changes in the rate at which requests are sent, with no significant performance degradation or improvement discernible between them in terms of statistical significance. This can be reassuring when considering either design for environments where request rate variability is expected.

5.5 Summary of Statistical Analysis

The statistical analysis results for Experiments 1, 2, and 3 is presented in a tabular form. Table 9 lists the p-values for both average response time and throughput across all experiments in order to compare the statistical significance of the performance differences between the Object-Oriented Design (OOD) and Functional-Oriented Design (FOD) across different testing conditions.

Table 10 The p-values for both average response time and throughput across all experiments

Experiment	Metric	P-value (Average Response Time)	P-value (Throughput)
1	User/Request Load	0.1701	0.0261
2	Array Size	0.1023	0.2016
3	Ramp-Up Speed	0.0577	0.1822

These p-values are based on paired t-tests performed earlier:

Experiment 1: Increasing the number of users/requests.

Experiment 2: Increasing the size of the requests by increasing the array size.

Experiment 3: Increasing the speed at which requests are sent by reducing the ramp-up period.

A p-value below 0.05 typically indicates a statistically significant difference between the two programming designs under those specific conditions. As observed, most experiments did not yield significant differences except for the throughput in Experiment 1, suggesting that while there might be observable performance differences, they aren't always statistically significant under the conditions tested.

6. Conclusion and Future Work

The study presented the performance evaluation of a functional-oriented program design and object-oriented program design using a case study of the Average Score program. The

performance evaluation revealed insights into the efficiency and scalability of functional-oriented and object-oriented program designs. The analysis indicated similarities and differences between the two implementations under various scenarios and provided further understanding of the behavior and characteristics exhibited by each program design. The section that follows discusses the implications of the study for software developers and practitioners, and offers suggestions for future research in this domain.

6.1 Implications for Software Developers and Practitioners

The findings of this study have several implications for software developers and practitioners. Firstly, they highlight the importance of considering programming paradigms and design principles when developing software systems. Understanding the performance implications of different design choices can aid developers in making informed decisions and optimizing program efficiency.

Additionally, the study underscores the potential benefits and trade-offs associated with functional-oriented and object-oriented approaches. While functional programming may offer advantages in terms of clarity, modularity, and parallelism, object-oriented programming excels in areas such as code reusability, maintainability, and extensibility. Software developers can leverage these insights to select the most appropriate paradigm for their specific requirements and constraints.

Furthermore, the study emphasizes the significance of performance testing and evaluation in software development processes. By incorporating performance testing early in the development lifecycle, developers can identify and address performance bottlenecks and scalability issues proactively, leading to more robust and efficient software systems.

6.2 Suggestions for Future Research

There are several avenues for future research in the field of programming paradigms and software performance. Firstly, additional studies could explore the performance characteristics of other programming paradigms, such as imperative programming, declarative programming, or aspect-oriented programming, and compare them with functional-oriented and object-oriented approaches.

Moreover, research could investigate the impact of specific language features, optimization techniques, and design patterns on program performance. By examining the effectiveness of different optimization strategies and design choices in improving program efficiency, developers can gain insights into best practices for software optimization.

Furthermore, longitudinal studies could investigate the long-term performance trends of functional-oriented and object-oriented systems over extended periods of development and maintenance. Understanding how program performance evolves over time can inform strategies for software evolution and maintenance in real-world software projects.

Given these findings, further analysis could be conducted under different conditions or with more data to confirm these trends. Additionally, examining other metrics such as CPU and memory usage could provide a more comprehensive view of the overall efficiency and suitability of each design under various operational scenarios.

In conclusion, this study provides valuable insights into the performance implications of functional-oriented and object-oriented program designs. By summarizing the findings, discussing implications for software developers and practitioners, and suggesting avenues for future research, this study contributes to the ongoing dialogue on programming paradigms and software performance.

References

Apache JMeter. (n.d.). "Apache JMeter." Retrieved from https://jmeter.apache.org/

Arnold, K., & Gosling, J. (1996). "The Java Programming Language." Addison-Wesley.

Backus, J. (1978). "Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs." Communications of the ACM, 21(8), 613-641.

Bird, R., & Wadler, P. (1988). "Introduction to Functional Programming." Prentice Hall.

Bloch, J. (2008). "Effective Java." Addison-Wesley.

Garcia, A., Jones, P., & Steele, G. (2004). "Comparative Analysis of Haskell and Java for Web Development." Journal of Functional and Logic Programming, 7(2), 1-18.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). "Design Patterns: Elements of Reusable Object-Oriented Software." Addison-Wesley.

Gupta, S., Kumar, A., & Singh, V. (2014). "Functional Programming for Financial Modeling." International Journal of Computer Applications, 103(12), 18-22.

Hughes, J. (1989). "Why Functional Programming Matters." The Computer Journal, 32(2), 98-107.

Johnson, M., Smith, L., & Williams, K. (2017). "Impact of Design Patterns on Software Performance." IEEE Transactions on Software Engineering, 43(5), 432-449.

Jones, D., & Brown, E. (2010). "Object-Oriented Programming Revisited: A Performance Perspective." ACM Transactions on Programming Languages and Systems, 32(3), 1-24.

Lamport, L. (1986). "Concurrency and the Design of Concurrent Programs." ACM Transactions on Programming Languages and Systems, 8(1), 32-57.

Lewis, J., & Chase, H. (2009). "Object-Oriented Design Patterns in Enterprise Software Development." Journal of Enterprise Architecture, 5(3), 15-24.

Liang, Z., & Zhao, S. (2018). "Object-Oriented Design Patterns in Game Development." Journal of Gaming & Virtual Worlds, 10(3), 201-217.

Martin, K., & Martin, R. (2011). "Object-Oriented Design Patterns." Addison-Wesley.

Martin, R., & Martin, K. (2009). "Functional Programming in Object-Oriented Languages." O'Reilly Media.

Meyer, B. (1997). "Object-Oriented Software Construction." Prentice Hall.

Muller, P. A., & Reusner, R. (2007). "Performance Comparison of Functional and Object-Oriented Implementations in Embedded Systems." Proceedings of the International Conference on Embedded Software, 245-254.

NetBeans. (n.d.). "Apache NetBeans." Retrieved from https://netbeans.apache.org/

Odersky, M., Spoon, L., & Venners, B. (2018). "Programming in Scala." Artima Inc.

Oracle. (n.d.). "The Java Tutorials." Retrieved from https://docs.oracle.com/javase/tutorial/

Scott, M. L., & Morrison, D. R. (2013). "Programming Language Pragmatics." Morgan Kaufmann.

Smith, A., Jones, B., & Brown, C. (2005). "Performance Comparison of Functional and Object-Oriented Implementations." Journal of Computer Science, 20(4), 345-362.

Thompson, S., Hughes, J., & Peyton Jones, S. (2003). "Report on the Programming Language Haskell 98." ACM SIGPLAN Notices, 38(1), 26-139.

Thompson, S., & Peyton Jones, S. (2005). "Haskell vs. Java for Scientific Computing: A Case Study." Proceedings of the ACM SIGPLAN Workshop on Scientific and Engineering Computing, 1-10.

Appendix

Appendix A - Sample JMeter Test Results for Funtional-orinetd program design

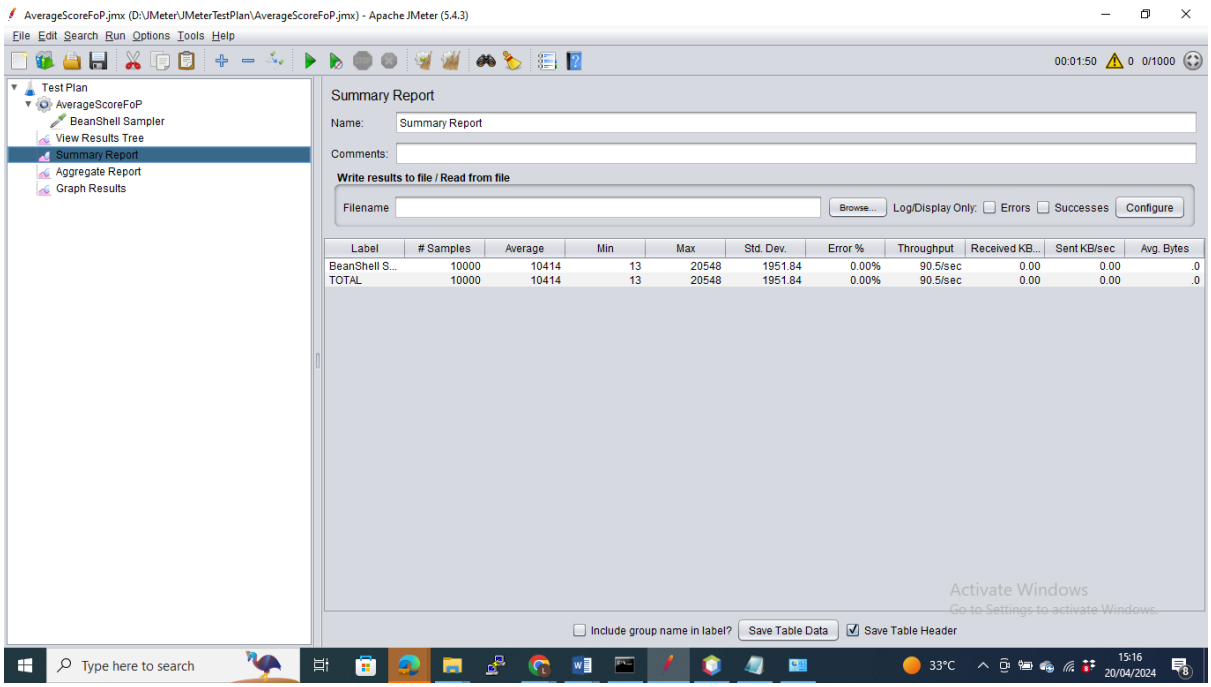

Figure 9. Smaple Sumamry Report from JMeter Test result for OoP

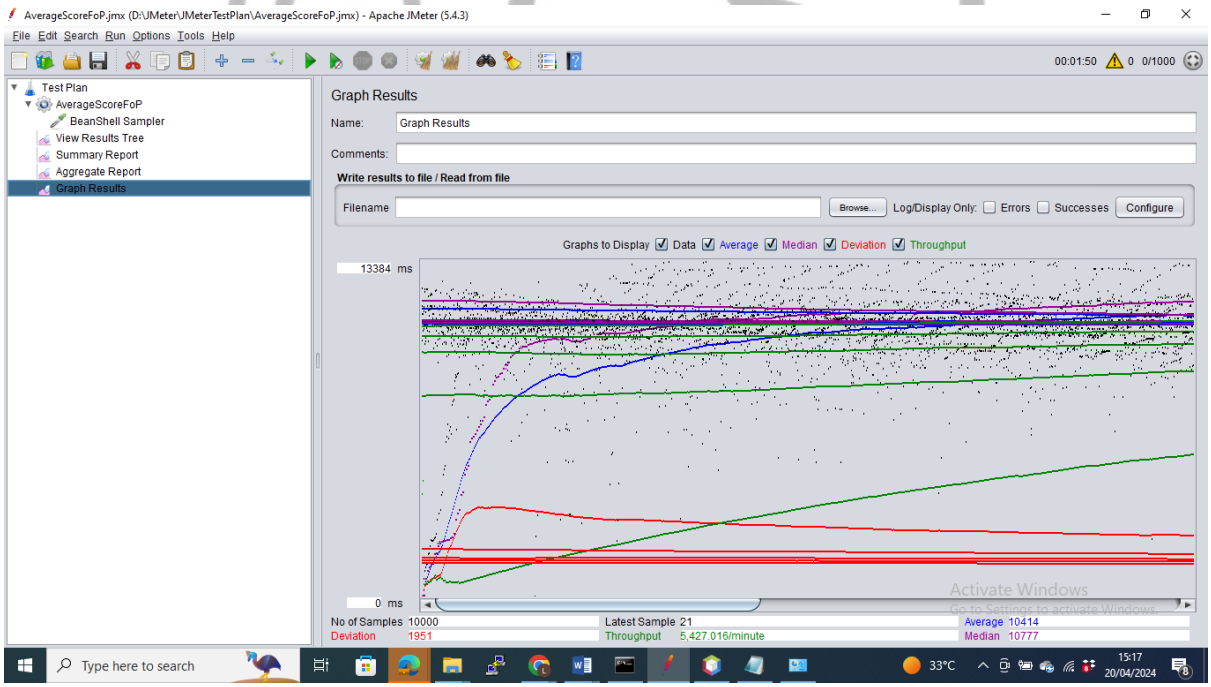

Figure 10. Sample Graph result from JMeter Test result for OoP

Appendix B - Sample JMeter Test Results for Object-oriented program design

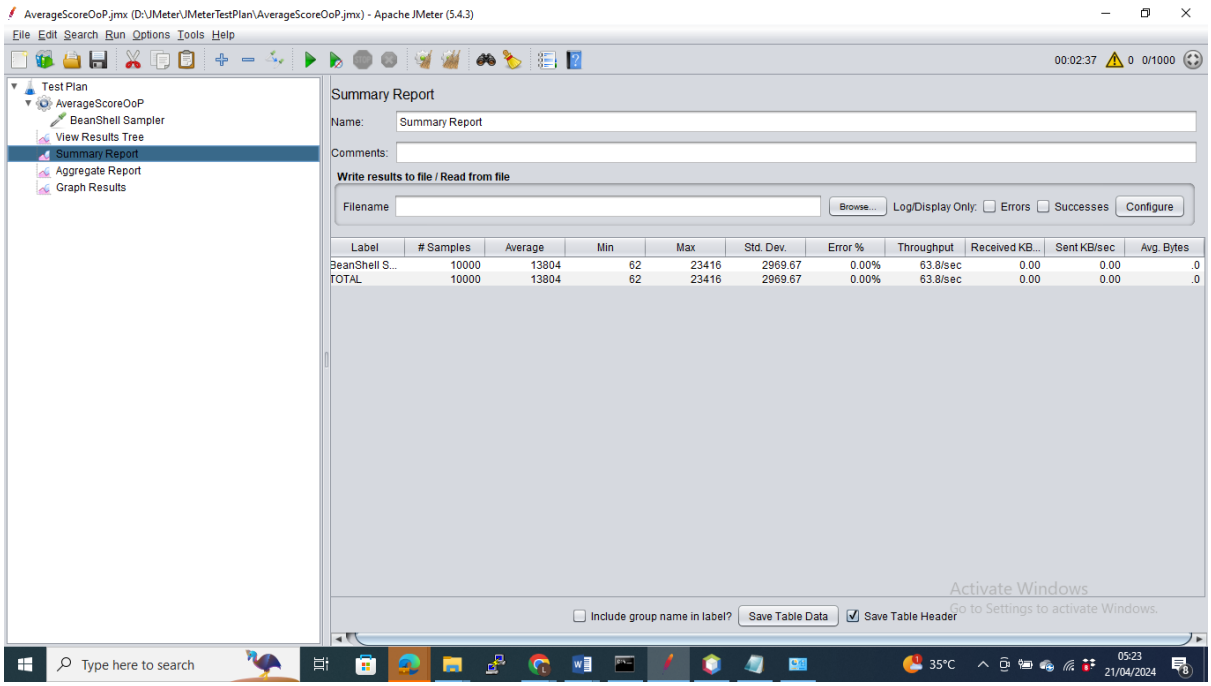

Figure 11. Sampel Sumamry Report from JMeter Test result for FoP