ASEAN Journal of Scientific and Technological Reports

Research article

Applying the Flyweight Design Pattern to Android Application Development

Wasana Ngaogate[1]

[1] Faculty of Science, Ubon Ratchathani University; 34190, Thailand; wasana.n@ubu.ac.th
* Correspondence: wasana.n@ubu.ac.th

Citation:
Ngaogate, W. Applying the flyweight design pattern to Android application development. *ASEAN J. Sci. Tech. Report.* **2023**, *26*(2), 49-57. https://doi.org/10.55164/ajstr.v26i2.247607.

Article history:
Received: October 24, 2022
Revised: March 17, 2023
Accepted: March 22, 2023
Available online: March 26, 2023

Abstract: This research aims to demonstrate how the Flyweight design pattern enhances programmers' ability to develop mobile applications while being aware of memory consumption flexibly. By following the systematic approach to software development, the two Android applications have been designed using a class diagram based on two paradigms: general object-oriented programming and the Flyweight design pattern. Both applications are developed in Java and installed on a physical Android phone. Memory usage is monitored by using the Android Profiler and using programming. The results show that both Android applications consume almost the same amount of memory, given all classes or package classes. Therefore, applying the Flyweight design pattern to mobile software development does not affect memory usage and follows a professional software design.

Keywords: Design patterns; flyweight design pattern; memory usage; Android application

1. Introduction

Some programmers develop software based on their familiar coding styles without concern for resource consumption, especially memory usage in mobile applications. In some situations, various information is shown in one view of the software, for example, a view of several items on one web page or mobile view. Each type of item contains information retrieved from different sources. Some programmers then create a new object for each item whenever needed. However, the information may be retrieved from the same source. It causes the applications to contain too many unnecessary objects. Moreover, a new source of information may be available after the software has been deployed. The programmers are asked to update the software to provide new items from the new source in the software. Based on the basics of object-oriented programming, they might recode the software by creating a new object for each new source without awareness of the memory usage. Also, the programmers might be confused with many objects and their messy codes.

It is important to disclose how coding styles or paradigms affect software performance so that programmers can easily decide which paradigms suit their work. The design patterns, frequently recognized as the best practices in programming, can be applied in various application development scenarios. Rimawi and Zein [1] encourage developers to use design patterns properly. They developed the PatRoid framework to detect design patterns in 1,400 Android applications. They found that only 4 design patterns were used, with a percentage ranging from 20% to 55%, and 38% of applications did not apply any design pattern. Design patterns enhance programmers' ability to develop more flexible and scalable software as reusable solutions for general problems, shorten software development time, and improve software quality. [2] New software engineers can also apply design patterns, as Lartigue and Chapman [3] confirmed.

This research aims to demonstrate how the design pattern paradigm can support programmers to flexibly develop mobile applications under the realization of memory consumption based on a systematic software design and development. Software performance is measured by comparing two coding paradigms; general object-oriented programming and the Flyweight design pattern. Both applications are installed and tested on a physical Android mobile. The memory usage of both applications developed by both paradigms is monitored using the Android Profiler and programming.

2. Materials and Methods

2.1 Software Performance: Memory Usage

Coding styles affect the software's performance. Therefore, software testing must be done carefully. The Android Studio Profiler is an efficient tool for monitoring a mobile application's performance. Hidayat and Sungkowo [4] used it for analyzing the CPU and the RAM usage when animations in JSON, PNG, and GIF types are run. Fatima et al. [5] compared the CPU and memory usage of a simple quiz Android application when it is developed using ListView compared to RecyclerView. They found that RecyclerView produced better application performance than ListView. Muhammad Ehsan Rana and Wan Nurhayati Wan Ab. Rahman [6] concluded that the Observer and the State design patterns are more effective for real-time application.

To measure the software performance of Android applications, Ghari et al. [7] are interested in problems around reliability, maintainability, and security. They studied the source codes of 10 applications and monitored their performances using MonkeyRunner and adb. Their findings confirm that quality is a multi-dimensional software complex and needs more tools to support software quality assurance. Abebaw Degu [8] reviewed 31 empirical papers on Android application memory and energy performance, resource leaks, and performance testing techniques and challenges. They identified several research gaps, such as memory and energy memory utilization optimization, including resource leaks, programming techniques, performance enhancement, and source code analysis tools. Cross-platform tools are also an option in mobile development. Dorfer et al. [9] are concerned about mobile applications built with cross-platform development approaches as they might consume system resources. So, they compared an application built using React Native to one built using native Android code and found that the React Native application consumed between 6% and 8% more energy than the Android native code.

This paper focuses on the memory usage of two Android applications: the first is developed using general object-oriented programming, and the second is developed by applying the Flyweight design pattern.

2.2 The Flyweight Design Patterns

Object-oriented design and programming are important paradigms in software development. It has been developed as a best practice called design patterns. Some open-source software with good architecture is built based on best practices, so the code is cleaner and has fewer infractions. The programs written using most of the design patterns were simpler compared to the ones written without them. Although the values of CK metrics, the number of classes, and the SLOC (Source Lines of Code) increased. [10-11] The GoF design patterns introduced by Erich et al. [12] are the most famous and applied in various business domains. For example, Bruno L. Sousa et al. [13] investigated large classes in five Java projects developed with design patterns. They concluded that the Composite and Factory Method patterns have a low co-occurrence with long methods. In contrast, the Template and Observer methods have a high co-occurrence with large classes and long methods.

The GoF Flyweight pattern intends to efficiently support large amounts of small-grained objects. A flyweight behaves as an independent object in each context and can be used in various contexts simultaneously. Therefore, if particular objects are stored on a server that many clients frequently request simultaneously, Flyweight can support those requests. The Flyweight pattern was compared with the Proxy [14] on software efficiency. They were applied to an online shooter game. The result showed that both design patterns spent less execution time and consumed less memory than expected. But the Proxy pattern consumes more memory than the Flyweight pattern. Peng Zhang et al. demonstrated how to apply Abstract Factory, Flyweight, Proxy, and Publisher-Subscriber patterns in a healthcare system called Smart Health (DASH), a distributed application that uses blockchain technology in the healthcare sector. They concluded that data sharing with the flyweight registry decreased the cost of changes to the common intrinsic state in blockchain-based apps. [15-16]

Dimitrichka Nikolaeva et al. [17] developed a Unity video game. They explained that the Singleton and Flyweight design patterns are applied to the MonoBehavior, such as the Update, LateUpdate, and other methods. The Flyweight is an object that minimizes memory usage by sharing as much data as possible with similar objects. Kirill Pupynin and Oleg Golovnin [18] developed the toolkit for modeling traffic flows based on a microscopic simulation model and the multimodal modeling system SUMO. They applied the Flyweight design pattern and found that the loading on memory was significantly reduced. Sepideh Maleki et al. [19] studied the impact of five design patterns on four object-oriented programmings (OOP) features inheritance, polymorphism, dynamic binding, and overloading. They found that the flyweight design pattern remarkably improved performance and energy efficiency.

However, Boyan Bontchev and Emanuela Milanova [20] received 82 usable responses from intermediate or higher-experienced software professionals in Bulgaria, with 65 experiencing one of the Gang of Four patterns. Most of them demonstrate good knowledge and a reasonable and responsible attitude regarding the advantages and drawbacks of design patterns. Their responses reveal that Singleton, Factory Method, and Iterator are the most recognized and valuable, while Memento and Flyweight are the most unrecognized and useless.

According to the benefits of the Flyweight design pattern discussed above, this research introduces how this pattern can be applied to developing an Android application embedded with information items retrieved from various sources. To reduce the number of unnecessary item objects created by programmers, which causes too much memory consumption, the design and development of an Android application using the Flyweight design pattern are demonstrated.

2.3 Method

The following subsections are the case study, two class diagrams for object-oriented programming and the Flyweight design pattern, the Java codes, and the software performance testing.

2.3.1. The case study

Systematic software development is important. Dong Kwan Kim [21] proposed guidelines for the software development activities and procedures for building mobile applications on the cloud service by applying UML diagrams and artifacts such as the UML profile extensions, class diagrams, and deployment diagrams. The experimental results suggest that the proposed guidelines can improve the productivity, scalability, and maintainability of software design models. For the case studies, they used the Android mobile platform, Amazon Web Service for cloud computing, and MySQL for data management.

To demonstrate the situation where many objects can be created in one particular view of the software, this research proposes a case study of a food list shown in one view of an Android application. The food has several types: meal, cocktail, dessert, or starter. Many types of food are randomly shown in one view of the application. Each type of food is retrieved from different sources, such as themealdb.com and thecocktaildb.com. Each source is managed using a different application object, created every time it retrieves the food information. Because the application needs to use different controller objects for different food types. Moreover, new food types might be added to the application later. For example, instead of the four food types described above, more types, such as seafood, noodles, or pizza, could be added to the application after deployment. Therefore, programmers must know the new controller objects for such new food types. Programmers create them whenever they retrieve the data, either by the new or existing controller objects. Consequently, the code is getting messy, and the controller objects are getting too many.

Instead of creating a new controller object every time the application retrieves the food information, the Flyweight design pattern, which was claimed to support the flexible use of the objects, has been applied. This research follows a systematic approach to software development by designing two class diagrams; one for the application developed by general object-oriented programming and another for the application developed by the Flyweight design pattern. The application is developed in Java using Android Studio. The food data are stored on a public server and provided freely via the REST API (REpresentational State Transfer Application Programming Interface). The speed of information retrieval depends on the data communication and does not impact memory usage.

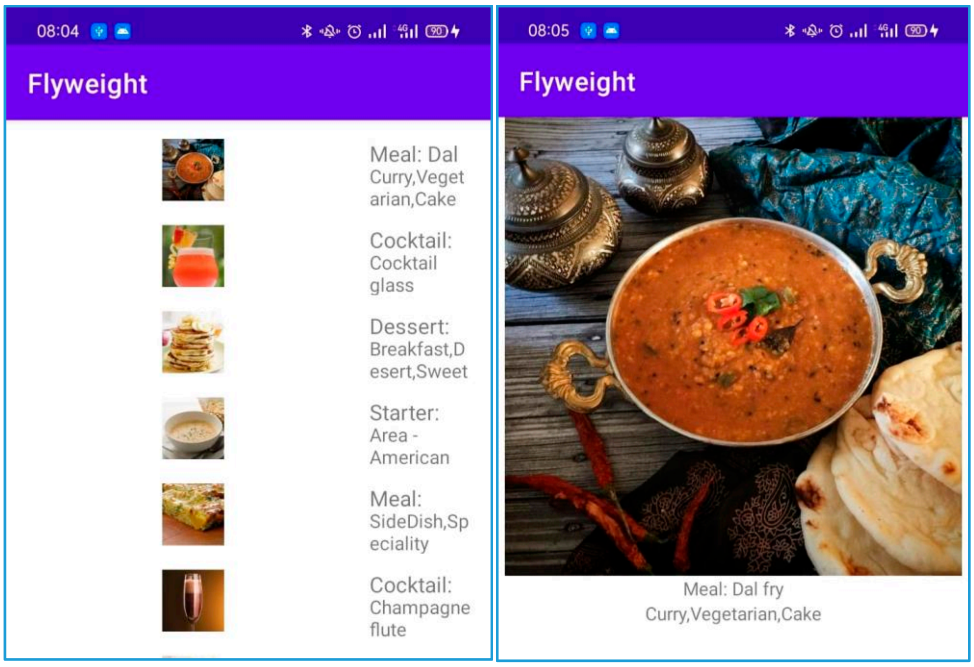
08:04
Flyweight
Meal: Dal Curry,Vegetarian,Cake
Cocktail: Cocktail glass
Dessert: Breakfast,Desert,Sweet
Starter: Area - American
Meal: SideDish,Speciality
Cocktail: Champagne flute
08:05
Flyweight
Meal: Dal fry Curry,Vegetarian,Cake

Figure 1. Main page and ShowDetail page

The list of various types of food is the main view of both applications. Types are selected in the same amount to be shown in the main view. The data of each type is randomly retrieved from a different REST API. Figure 1 is the main view, which shows a list of various kinds of food. Once the user clicks on a particular image, the details of that image are shown in the view on the right.

2.3.2. The class diagram for General Object-Oriented Programming

The common attributes of the food are id, name, description, and image. Therefore, Data is defined as an abstract class with two children at the beginning: Meal and Cocktail. Each data object's child has its controller class for data retrieval. Those controller classes implement the interface class RESTData, as shown in the class diagram below.

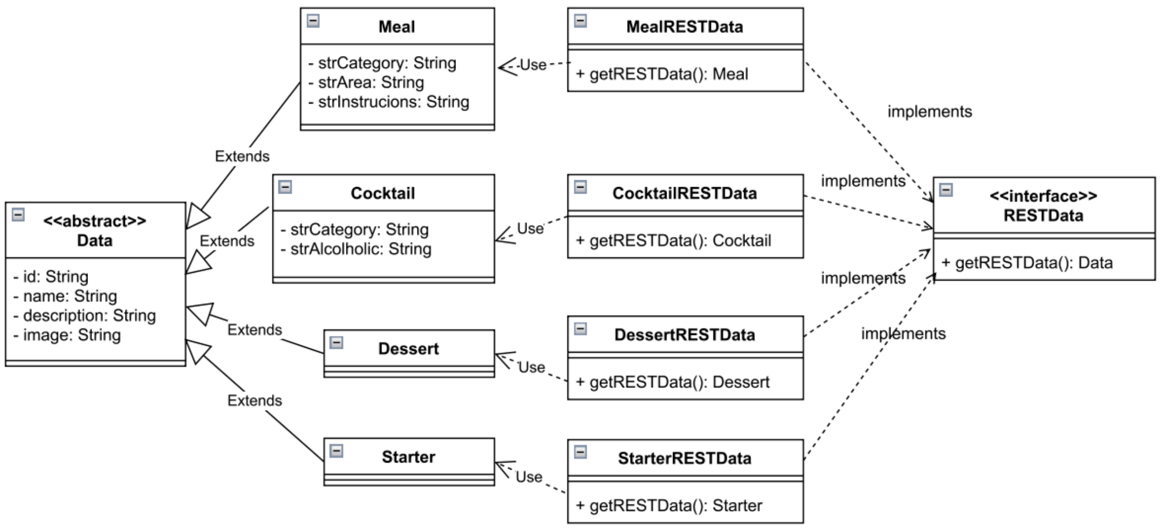

Figure 2. The class diagram for general object-oriented programming

2.3.3 The class diagram for the Flyweight design pattern

The class diagram for the Flyweight design pattern is almost the same as the one in the above subsection, except for the class FoodRESTFactory. According to the Flyweight design pattern, the controller classes are kept in one collection, such as an ArrayList or HashMap. Once the main program needs to retrieve a particular piece of data provided by the controller, it is checked to see if it has already been created and stored in the collection. If it is stored, it will be retrieved and used. Otherwise, it is created and stored in the collection.

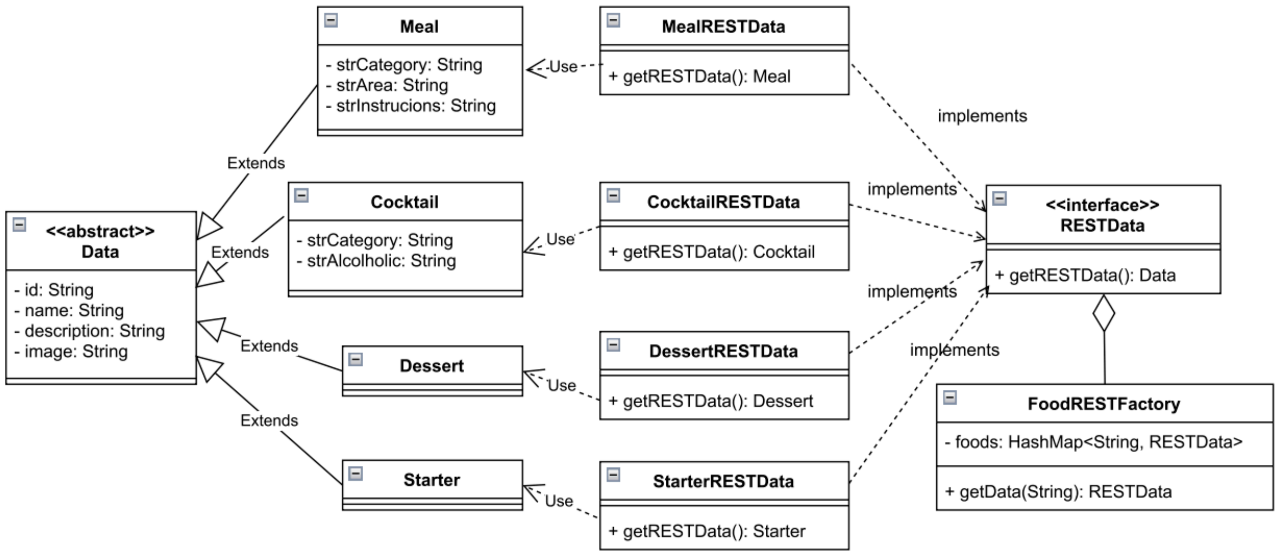

Figure 3. The class diagram for the Flyweight design pattern

2.4 Java Codes

The following codes show how both applications launch four services, 10 times and 50 times, respectively.

2.4.1. General Object-Oriented Programming

The application developed using general object-oriented programming creates each object of MealRESTData, CocktailRESTData, DessertRESTData, and StarterRESTData one by one every time it needs more REST data.

Table 1. General Object-Oriented Programming: MainActivity.java

General Object-Oriented Programming: MainActivity.java

```
List<Data> dataList = new ArrayList<Data>();
for(int i=0; i < 50; i++){
  if(i%4 == 0){
    dataList.add(new MealRESTData().getRESTData());
  }else if(i%4 == 1){
    dataList.add(new CocktailRESTData().getRESTData());
  }else if(i%4 == 2){
    dataList.add(new DessertRESTData().getRESTData());
  }else{
    dataList.add(new StarterRESTData().getRESTData());
  }
}
```

[1] This table demonstrates only two instances of a Data object creation to minimize the page space.

2.4.2. Flyweight Design Pattern Programming

The application creates only one instance of FoodRESTData, which creates an instance of MealRESTData, CocktailRESTData, DessertRESTData, or StarterRESTData when it is necessary. Once one of them is created, it is kept in a HashMap and retrieved when it is needed again without creating another instance.

Table 2. Flyweight Design Pattern Programming: MainActivity.java

Flyweight Design Pattern Programming: MainActivity.java

```
List<Data> dataList = new ArrayList<Data>();
for(int i=0; i < 50; i++){
  if(i%4 == 0){
    dataList.add(FoodRESTFactory.getData("meal").getRESTData());
  }else if(i%4 == 1){
    dataList.add(FoodRESTFactory.getData("cocktail").getRESTData());
  }else if(i%4 == 2){
    dataList.add(FoodRESTFactory.getData("dessert").getRESTData());
  }else{
    dataList.add(FoodRESTFactory.getData("starter").getRESTData());
  }
}
DataAdapter dataAdapter = new DataAdapter(dataList, this);
recyclerView.setAdapter(dataAdapter);
```

[1] This table demonstrates only two instances of a Data object creation to minimize the page space.

Table 3. Flyweight Design Pattern Programming: FoodRESTFactory.java

Flyweight Design Pattern Programming: FoodRESTFactory.java

```
public class FoodRESTFactory {
  // HashMap of RESTData
  private static final HashMap<String, RESTData>
        foods = new HashMap<String, RESTData>();

  // method: getData()
  public static RESTData getData(String foodType){
    RESTData restData = foods.get(foodType);
    if (restData == null){
      restData = new MealRESTData();
    }else if (foodType.equals("cocktail")){
      restData = new CocktailRESTData();
    }else if (foodType.equals("dessert")){
      restData = new DessertRESTData();
    }else if (foodType.equals("starter")){
      restData = new StarterRESTData();
    }
    foods.put(foodType, restData);
   }
   return restData;
  }
}
```

As the code in Table 3, programmers do not need to know the controller class names, such as MealRESTData or CocktailRESTData, as in Table 2. They pass a string of their names to the FoodRESTFactory, which is simpler, for example, "meal" or "cocktail." Once the new controller classes are introduced, for example, "seafood," "noodle," or "pizza,." Programmers pass those strings to the FoodRESTFactory. The code is clean and easy to manage.

2.5 Software Performance Testing: Memory Usage

This research deployed both applications on the same physical Android device: Oppo A5 2020 RAM 3GB Qualcomm Snapdragon 665, Device storage: available 29.1GB, total 64.0 GB, Android version 10, and ColorOS version v7.1. Memory usage is detected by using the Android Profiler embedded in the Android Studio and by programming.

3. Results and Discussion

The two Android applications are deployed on a physical mobile device, Oppo A5 2020 RAM 3GB Qualcomm Snapdragon 665. The memory usage is then monitored by the Android Profiler embedded in Android Studio. Table 4 shows the amount of memory used by each application; the application developed based on a general object-oriented paradigm and the application developed based on the Flyweight design pattern. Each application is tested twice: once with 10 items of food and again with 50 items.

There are 3 views shown in the Android Profiler; View all classes, Show project classes, and the class MainActivity.

Table 4. Memory usage monitored by the Android Profiler (in bytes)

	General Object-Oriented		**Flyweight design pattern**	
View app heap: Arrange by class: **View all classes**				
	Show 10 items	Show 50 items	Show 10 items	Show 50 items
Classes	1,289	1,435	1,263	1,400
Count	24,443	29,975	20,564	29,723
Native size	382,989	3,001,440	312,273 bytes	2,899,867
Shallow size	2,666,908	2,309,909	2,166,562 bytes	2,121,671
Retained size	8,585,130	25,547,875	6,667,429	24,058,038
View app heap: Arrange by package: **Show project classes**				
	Show 10 items	Show 50 items	Show 10 items	Show 50 items
Classes	8	9	12	13
Count	32	83	36	87
Native size	0	0	0	0
Shallow size	1,892	3,984	1,904	4,016
Retained size	6,167	87,448	6,296	121,348
View app heap: **Class MainActivity**				
	Show 10 items	Show 50 items	Show 10 items	Show 50 items
Allocations	1	1	1	1
Native Size	0	0	0	0
Shallow Size	324	324	324	324
Retained Size	4,040	4,076	4,220	4,040

The shallow size means the total amount of Java memory used by this object, and the retained size means the total amount of memory being retained due to all instances of this class and ready to be cleaned by the garbage collector. As with Table 4, the memory usage of both applications is nearly the same, whether showing only 10 items or 50 items. The Flyweight application consumes a little bit less memory than all other classes. But it consumes a little bit more memory from the perspective of project classes. The memory consumption in the MainActivity is also nearly similar.

Table 5. Memory usage monitored programmatically: ActivityManager.MemoryInfo (in bytes)

	General Object-Oriented		**Flyweight design pattern**	
	Show 10 items	Show 50 items	Show 10 items	Show 50 items
Available memory	1,060,016,128	976,662,528	1,098,248,192	1,024,471,040
Threshold	467,364,864	467,364,864	467,364,864	467,364,864
Total Memory	2,771,116,032	2,771,116,032	2,771,116,032	2,771,116,032

Table 5 shows memory usage in a physical mobile device: the Oppo A5 2020. It is detected programmatically using a class called ActivityManager, showing its memory information at the Android Studio's run shell. According to the available memory of MainActivity, the application developed using the Flyweight design pattern consumes almost the same amount of memory as the general object-oriented application. They both have the same threshold, a memory level at which the system begins to kill processes.

4. Conclusions

Programmers need a delicate software design before coding. They might be familiar with the general object-oriented paradigm, which is insufficient for building better software. This research demonstrates applying the Flyweight design patterns to Android application development. Following a systematic approach to software development, two class diagrams were designed; one for the application, which is developed based on the general object-oriented paradigm, and another for the application, which is developed based on the Flyweight design pattern. The example data is retrieved from a REST API. The applications are deployed on a physical mobile device. Memory usage of both Android applications was monitored by using the Android Profiler and programming. The results show that both applications consume almost the same amount of memory, either a view of all classes or project classes. The MainActivity class, which is the application's main view, also consumes the same amount of memory.

Applications of design patterns could be used in various problem domains, either GoF design patterns or other patterns experienced software engineers introduce. Especially software designers are responsible for making decisions about the software components. Design patterns should be applied along with software testing to prove that the patterns do not affect the software's performance.

5. Acknowledgements

Author Contributions: Literature review, software design, development and testing, discusssion and conclusion. Wasana Ngaogate.

Funding: -

Conflicts of Interest: The author declares no conflict of interest..

References

[1] Rimawi, D.; Zein, S. A Model Based Approach for Android Design Patterns Detection. Proceedings of the 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), Ankara, Turkey, 11-13 October 2019; IEEE: Turkey.

[2] Karavokyris, A.; Alpis, E. Software Measures for Common Design Pattern Using Visual Studio Code Metrics. Proceedings of the International Conference on Information, Intelligence, Systems and Applications, Zakynthos, Greece, July 23-25, 2018, ACM.

[3] Lartigue, W. J.; Chapman, R. Comprehension and Application of Design Patterns by Novice Software Engineers. Proceeding of the Annual ACM Southeast Conference, Kentucky, United States, March 29 – 31, 2018, ACM.

[4] Hidayat, T.; Sungkowo, B. D. Comparison of Memory Consumptive Against the Use of Various Image Formats for App Onboarding Animation Assets on Android with Lottie. Proceedings of the 3rd International Conference on Computer and Informatics Engineering (IC2IE), Yogyakarta, Indonesia, 15-16 September, 2020; IEEE.

[5] Fatima, S.; Steffy, N.; Stella, D.; Nandhini D.; Devi, S. Enhanced Performance of Android Application Using RecyclerView. *Advanced Computing and Intelligent Engineering. Advances in Intelligent Systems and Computing*, Springer, Singapore, 2020*, 1089*, 189–199.

[6] Ehsan, M. R.; Wan Nurhayati, W. W. The Effect of Applying Software Design Patterns on Real Time Software Efficiency. Proceedings of the Future Technologies Conference (FTC), Pan Pacific Hotel Vancouver, BC, Canada, 29-30 November 2017.

[7] Ghari, S.; Hadian, M.; Rasolroveicy, M.; Fokaefs, M. A multi-dimensional quality analysis of Android applications. Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering, Markham, Ontario, Canada, November 4-6, 2019. ACM.

[8] Abebaw, D. Android Application Memory and Energy Performance: Systematic Literature Review. *Journal of Computer Engineering (IOSR-JCE)*, 2019, *21*(3), 20-32.

[9] Dorfer, T.; Demetz, L.; Huber, S. Impact of mobile cross-platform development on CPU, memory, and battery of mobile devices when using common mobile app features. Proceedings of the 17th International Conference on Mobile Systems and Pervasive Computing (MobiSPC), Leuven, Belgium, August 9-12, 2020.

[10] Feitosa, D.; Ampatzoglou, A.; Avgeriou, P.; Chatzigeorgiou, A.; Nakagawa, Y. E. What can violations of good practices tell about the relationship between GoF patterns and run-time quality attributes?. *Information and Software Technology*, 2019, 105, 1-16.

[11] Qamar, N.; Malik, A. A. Impact of Design Patterns on Software Complexity and Size. *Mehran University Research Journal of Engineering and Technology*, (S.l.), *39*(2), 342-352.

[12] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, United States, 1994.

[13] Sousa, L. B.; Bigonha, A. S. M.; Ferreira, A. M. K. Evaluating Co-Occurrence of GOF Design Patterns with God Class and Long Method Bad Smells. Proceedings of the XIII Brazilian Symposium on Information Systems, Lavras, MG, Brazil, June 5-8, 2017.

[14] Rana, E. M.; Rahman, W. N.; Murad, A. A. M.; Atan, B. R. The Impact of Flyweight and Proxy Design Patterns on Software Efficiency: An Empirical Evaluation. *International Journal of Advanced Computer Science and Applications*, 2019, *10*(7), 161-170.

[15] Zhang, P.; White, J.; Schmidt, C. D.; Lenz, G. Design of Blockchain-Based Apps Using Familiar Software Patterns with a Healthcare Focus. Proceedings of the 24th Conference on Pattern Languages of Programs, Vancouver British Columbia, Canada, October 23 - 25, 2017; ACM.

[16] Zhang, P.; Schmidt, C. D.; White, J.; Lenz, G. Blockchain Technology Use Cases in Healthcare. *Advances in Computers*, Elsevier, 2018, 111, 1-41.

[17] Nikolaeva, D.; Safi, M.; Mihailov, M.; Georgiev, A.; Bozhikova, V.; Stoeva, M. Algorithm A* and Design Patterns used in Unity Video Game development. Proceedings of International Conference Automatics and Informatics (ICAI), Varna, Bulgaria, 2020, 1-3.

[18] Pupynin, K.; Golovnin, O. A Microscopic Traffic Simulation Web Toolkit. Proceedings of the 8th Scientific Conference on Information Technologies for Intelligent Decision Making Support, Ufa, Russia , 6-9 October 2020.

[19] Maleki, S.; Fu. C.; Banotra, A.; Zong, Z. Understanding the impact of object oriented programming and design patterns on energy efficiency. Proceedings of the 8th Int Green and Sustainable Computing Conference, Orlando, FL, USA, 23-25 October 2017, 1-6.

[20] Bontchev, B.; Milanova, E. On the Usability of Object-Oriented Design Patterns for a Better Software Quality. *Cybernetics and Information Technologies*, 2020, *20*(4), 36–54.

[21] Kim, K. D. Development of Mobile Cloud Applications using UML. *International Journal of Electrical and Computer Engineering* 2018, *8*(1), 596-604.