Designing an emergency management system using software design patterns

Ajay Bandi* and Mark Corson†

**Department of Mathematics, Computer Science, and Information Systems*
†Department of Natural Sciences
**†Northwest Missouri State University, Maryville, USA*
**ajay@nwmissouri.edu, †mcorson@nwmissouri.edu*

Keywords: design patterns, composite pattern, façade pattern, adapter pattern, iterator pattern

Abstract

Design patterns in software projects are very useful for enhancing the functionality of the software. However, many software developers implement code without using design patterns. This often causes problems during the software maintenance phase. This paper presents the hypothetical views and lessons learned in applying design patterns if we knew about design patterns during the design phase of the software development lifecycle (SDLC). We report our ideas for using design patterns based on real experience in developing software to verify specific properties of managing disasters. We explain the design patterns and how they fit in our project. The lessons learned recommendations, and conclusions of this paper help to motivate and educate future software developers and practitioners about the design patterns in object-oriented programming.

1 Introduction

The different phases of the SDLC, such as design, implementation, testing, and maintenance, take significant amounts of time to accomplish by software engineers. To reduce the time spent in these phases, software engineering practitioners use various techniques such as design patterns and automated testing while implementing code [4]. During the design phase, using design patterns can save considerable amounts of time. The main motivation of this paper is to contrast the difficulties in developing a real software example with and without the use of design patterns. This paper presents how to use design patterns effectively based on our hypothetical experience. This also includes the hypothetical lessons learned and recommendations for software developers. We also describe our experience using design patterns in an Emergency Disaster Management System (EDMS) to manage and rescue people during various emergency situations.

The use of design patterns in software projects often makes it easier to understand and modify the code, as well as enhance other functionalities based on the requirements of the project. This paper explains how the facade, composite, iterator, and adapter design patterns are a good fit for EDMS.

2 Overview of EDMS

An emergency disaster management system is the creation of plans through which communities reduce vulnerability to hazards and cope with disasters. Disaster management does not avert or eliminate the threats; instead, it focuses on creating plans to decrease the impact of disasters. Failure to create a plan could lead to damage to assets, human mortality, and lost revenue. These plans include mitigation, prevention, preparedness, rescue, and recovery.

The main purpose of the EDMS is to ensure public safety at the time of a disaster by adapting to disaster management plans. This system has an administrator who is responsible for the implementation of the management plans. An administrator responds at the time of disaster and informs the rescue teams regarding the disaster and allocates certain locations to the teams to take actions to implement the rescue and recovery operations. Rescue teams are the trained personnel who voluntarily take part in the training sessions conducted by the rescue experts. For example, sessions on tornado rescue, earthquake rescue, flood rescue, emergency medical help, intruder rescue etc.

Initially, the administrator has to register a disaster into the system. The administrator responsibilities include form rescue teams, add the people into the rescue team, delete people from the teams, move people from a team to another team, assign a team leader, send alert messages, view the location of the rescue teams, filter the information sent by the teams based on time, severity and type of hazard, and accept and reject the requests from the team members.

The administrator will be using the web application and mobile application to monitor the rescue operations and prioritize the resources based on the information from the rescue teams. Team members need to register to the system and log in to send information back to the administrator. This information includes the location of the teams, the severity of the damage occurred, different types of hazards that are prone to the rescue location, number of remains at the location and also the resources that are required to accommodate people

and move them to relocate to the safe zones. The administrator also maintains the history of the disasters occurred in the past.
As rescue teams are not stationary and move from place to place, the mobile application will be used to communicate back to the administrator. There is a need to have effective communication between the rescue commander and the team members while rescuing the victims. Sometimes it results in an instant need of the trained people for rescuing due to high severity. The team members should create a clear view for the rescue commander from the rescue location. The rescue commander should able to have full control over the team members so that rescue operations will hold under certain rules and regulations.

This application was developed in our graduate directed project course, a capstone project, of our graduate program (MS-ACS). Two teams, each team of size 10 implemented EDMS in Android and iOS applications without software design patterns. This causes maintenance [6, 7, 8] and performance issues such as harder to enhance the functionality of the system, repair defects, and modifying source code. This is a challenging task for the programmer analysts. The solution for this problem by applying software design patterns while designing the system.

3 Why considering design patterns in EDMS

The logical structure of EDMS consists of the input data read by the data input reader and sends a message to the Command Post. The EDMS software is difficult to modify due to its complexity, which also inhibits efficient testing and debugging. Adding functionality for other objects becomes more complex because of this bad design. Based on knowledge from the software design patterns course, we will use some of the design patterns in the EDMS (both Android and iOS projects).

3 Applying design patterns

This section describes all design patterns suitable for use in the EDMS. The main structure of the EDMS uses the facade pattern. Furthermore, the composite pattern is also implemented for representing different data objects. The iterator pattern is used for the collection of data items, and an adapter pattern is used when interfaces do not match each other. For every pattern used, we evaluate the following criteria:

- Lessons learned after applying the pattern.
- Modularity, reusability and understandability of code.
  - **Modularity:** Subdividing the system into further simpler subsystems.
  - **Reusability:** Using a particular working module in different subsystems.
  - **Understandability:** Readability & comprehending the code.

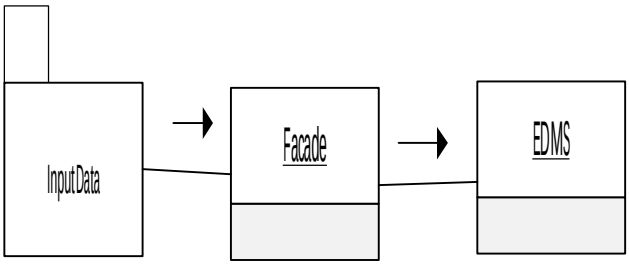

Fig 1. Collaboration diagram for façade pattern

3.1 Façade pattern

Application of pattern: We used the facade pattern in between the data input (subsystem) and the emergency disaster management system (Client).

Explanation: The logical structure of the EDMS represented by the data, is read and parsed using the parser to extract all the information about various disasters. The extracted data is then used for the verification. Here the facade pattern is used in between the data input and the EDMS. In Figure 1, the subsystem is the data input and cannot be modified, and the client is the EDMS, which can be modified using the parser data. Figure 1 shows the collaboration diagram for the facade pattern. Data input is a subsystem, which cannot be modified, and the EDMS is the client.

Evaluation: After using the facade pattern, the EDMS system can be easily developed in different modules reducing the coupling between the client and subsystems. It makes the application more modular because all of the code not using the facade pattern has been in the input data subsystem, which causes more confusion for the programmer. Translating the required data objects from input data to the EDMS becomes easy using the facade pattern. Understandability of the code is much better after documenting the usage of the facade pattern [2]. There is no reusability of code with the facade pattern. However, we can design other interfaces based on the requirements of clients using these patterns.

3.2 Composite pattern

Application of pattern: Composite pattern is used to represent the data objects of the EDMS.

Explanation: In general the composite pattern is used for hierarchical data [3]. The Command Post manages list of rescue teams, team members, and different types of rescue teams (earthquake rescue team, flood rescue team, power outrage rescue team etc.) The Command Post supervised a list of rescue teams and each rescue team consists of an array of team members. The composite pattern is the best pattern to use for the EDMS to represent the data

objects because it manages the children at a run time. Figure 2 shows the class diagram for the composite pattern. Here compartment and side are two composite objects and the leaf is node object.

Evaluation: The composite pattern reduces the complexity of hierarchical or tree structure data into the simple structure. This pattern makes the EDMS more modular. The code is implemented as different modules for each of the data objects. Testing becomes easier after implementing the pattern. Readability of source code is simpler when we use the composite pattern with documentation in the code [2]. Reusability of modules is not done frequently in our project.

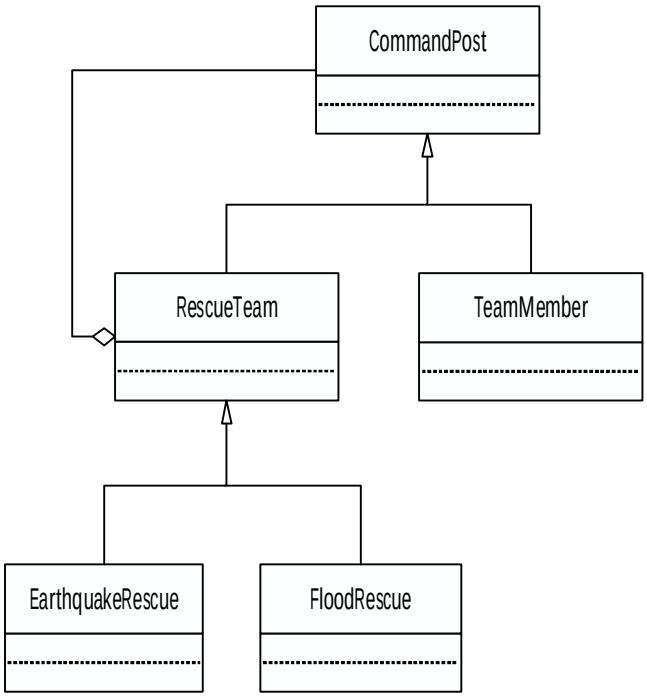

Fig. 2. Class diagram for composite pattern

3.3 Iterator pattern

Application of pattern: A simple way to access the elements in the disasters list.

Explanation: In our project, we have different collections of objects. They are disasters, rescue teams, and rescue team members. These collections can be handled by several data structures such as linked lists, array lists, or hash tables. These structures are part of the collections library in Java and .NET. In our project, the disasters can be stored by using `LinkedList` or an `ArrayList`. The main difference between them is in the underlying storage format and efficiency of accessing elements from these structures.

However, the program should behave the same whether we are using a LinkedList or an ArrayList. Therefore, an iterator pattern is useful. An `Iterator` is an interface that traverses through the elements of a list. If we use the iterator pattern to traverse disasters, then the programmer can use either a linked list or an array list to handle these collections of data items. Figure 3 shows the sequence diagram for the iterator pattern. The object shown in Figure 3 is the list of disasters. Using the `hasNext()` and `next()` methods in the iterator pattern we can traverse the list. Thus, we are not depending on the size of the list. So the programmer has the flexibility of using array list or link list or hash tables.

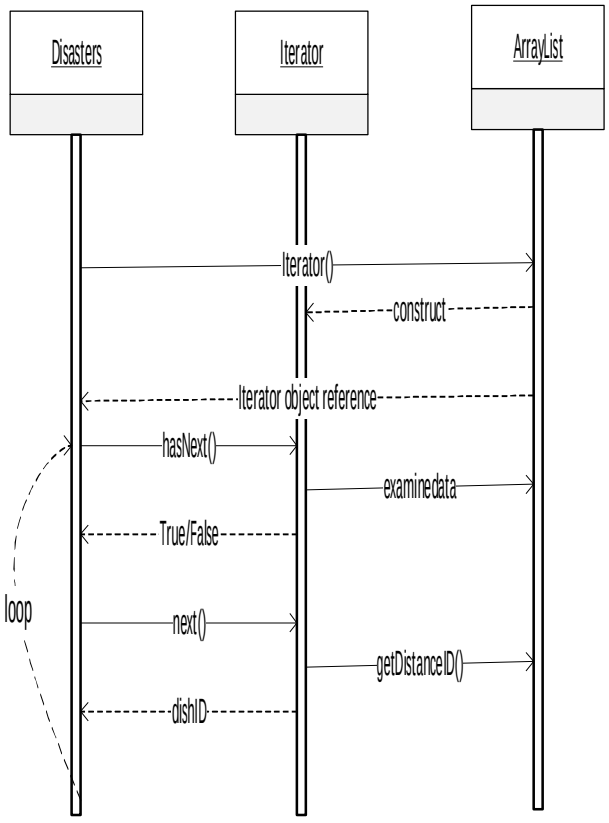

Fig. 3. Sequence diagram for Iterator pattern

Evaluation: Mainly the iterator pattern is used for decoupling usage and actual implementations. This pattern makes the code more flexible. By using the iterator pattern, there is no advantage for modularity of code. Documenting the pattern usage in source code improves the readability [2]. The iterator pattern is reusable for different types of objects.

3.4 Adapter pattern

Application of pattern: The adapter pattern is used when two interfaces in the EDMS do not match.

Explanation: There are several different places we used the adapter pattern in our EDMS. In the adapter pattern, both the client and the adaptee cannot be modified. The adding or changing of the requirements results in enhancing the functionality of the project. For this, modifying of the existing source code may lead to major rework of the existing code. To avoid this, we added some methods so that the code works for both the old and new requirements. This is the handiest design pattern used in our project. Figure 4 shows the collaboration diagram for the composite pattern. In our project, both the client and adaptees are methods.

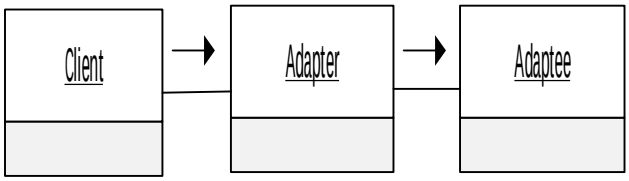

Fig. 4. Collaboration diagram for adapter pattern

Evaluation: The adapter pattern communicates between the interfaces without any modifications. By using the adapter pattern we can simplify the complexity of the code by designing adapter interfaces separately. Modularity of the code is improved after applying the adapter pattern. There can be no reusability of code when an adapter pattern is used. Understanding of source code is simpler when we use an adapter pattern with proper documentation [2].

4 Lessons learned and recommendations

Lessons Learned: This section explains the different lessons learned in applying design patterns in the EDMS. This section also highlights the top recommendations to software engineering practitioners for using these design patterns.

- Communication between the team members is very easy when we use design patterns in the software provided that team members have knowledge of the patterns.
- The complexity of software can be managed easily even though requirements are changed.
- Unit testing at the class level becomes significantly less expensive when we apply design patterns.
- By using design patterns modularity, reusability, and understandability of code increases. Thus, maintenance of software is relatively easy.
- The time taken for the software design process is reduced or automated.

Recommendations: The following are the recommendations to software engineering practitioners using design patterns in projects.

- Apply appropriate design patterns wherever applicable in the software projects.
- If any design patterns are used in the software, then document the names of these patterns in the comments of the source code.
- Using unsuitable design patterns may complicate the software.
- Students who are seeking a programmer analyst, business analyst or software architect jobs should take a course on design patterns!

Acknowledgements

We would like to thank the graduate directed project teams who worked on the implementation of emergency disaster management systems both in Android or iOS applications without using software design patterns.

5. Future work

In the future, we plan to design user interfaces of EDMS for Android and iOS applications and conduct usability testing [9] on the interfaces to find out the problems of using EDMS in real world situations.

References

[1] K. Beck, J. O. Coplien, R. Crocker, L. Dominick, G. Meszaros, F. Paulisch, and J. Vlissides, “Industrial Experience with Design Patterns,” *Proceedings: 18th International Conference on Software Engineering*. IEEE Press, 1996, pp. 103–114.

[2] L. Prechelt, B. Unger-Lamprecht, M. Philippsen, and W. F. Tichy, “Two Controlled Experiments Assessing the Usefulness of Design Pattern Documentation in Program Maintenance,” *IEEE Transactions on Software Engineering*, vol. 28, no. 6, June 2002, pp. 595–606.

[3] T. D. Thu and H. T. B. Tran, “A Composite Design Pattern for Object Frameworks,” *Proceedings: 31st Annual International Computer Software and Applications Conference*, Beijing, 2007, IEEE Computer Society.

[4] P. Wendorff, “Assessment of Design Patterns during Software Reengineering: Lessons Learned from a Large Commercial Project,” *Proceedings: 5th European Conference on Software Maintenance and Reengineering*. IEEE Press, 2001, pp. 77– 84.

[5] B. Wydaeghe, K. Verschaeve, B. Michiels, B. V. Damme, E. Arckens, and V. Jonckers, “Building an OMT-Editor Using Design Patterns: An Experience Report,” *Proceedings: Technology of Object-Oriented Languages TOOLS 26*, California, 1998, IEEE Computer Society.

[6] A. Bandi, E. B. Allen, and B. J. Williams, "Assessing code decay: A data-driven approach," in *Proceedings of ISCA 24th International Conference on Software Engineering and Data Engineering*. San Diego, California, USA: ISCA, Oct. 2015.

[7] A. Bandi, *Assessing Code Decay by Detecting Software Architecture Violations*, doctoral dissertation, Mississippi State University, Dec. 2015,

[8] A. Bandi, B. J. Williams, and E. B. Allen, "Empirical evidence of code decay: A systematic mapping study," in *Proceedings of 20thWorking Conference of Reverse Engineering*. Koblenz, Germany: IEEE, Oct. 2013, pp. 95—102.

[9] A. Bandi and P. Heeler "Usability testing: A software engineering perspective," in *Proceedings of 2013 International Conference on Human-Computer Interaction (ICHCI)*. Chennai, India: IEEE, Aug. 2013, pp. 1—8.