Case Study: Seat Allocation Program

Zina Abohaia
Faculty of Computer Science
British University in Dubai
Dubai, UAE
20194621@ug.buid.ac.ae

Yousef Mamdouh
Faculty of Computer Science
British University in Dubai
Dubai, UAE
20196448@ug.buid.ac.ae

Sally Blata
Faculty of Computer Science
British University in Dubai
Dubai, UAE
20193300@ug.buid.ac.ae

***Abstract*—Object Oriented Programming (OOP) is a comprehensive solution and an efficient programming method that has gained recognition for its use to code applications large in size, or important in value. The value of OOP in more simple applications has been overlooked, especially in mundane, time-consuming tasks. Seat planning, or allocation, is one of those areas. An event seating plan seems easy to produce but the truth is, it is mainly why event planners are hired, and where most time and effort is spent when planning an event. In this paper, we show the OOP seat allocation program we have done, analyze its results, and discuss why it was easier with OOP. We have also followed the UP phases in the making of the project, and we have included our experience with it.**

Keywords—OOP, UP Phases, Seat Allocation, Seat Planning

I. Introduction (Problem/Domain)

Seating plans are a part of many organizations in a society, Schools, offices, weddings, and events in general. They have a big effect on all aspects of communication, for example in schools a teacher has to arrange students in a way that is most effective to their learning journey, in an office you want the employees to be most productive, and in weddings what matters most is their entertainment. As easy as seat allocation seems, it ends up being one of the most stressful tasks, one that ends up taking days to get done. Many issues arise while creating a seating plan, besides it being time consuming, it's also resource consuming. If any changes occur, more money needs to be spent on organizers to redo the seating plans. The Purpose of this paper is to use OOP to solve this problem, and to examine how OOP can be used to solve such a seemingly simple problem

Hypothesis—An OOP application that produces seating plans would save money, time, and effort.

Project Details—We need to create a program that generates a seating plan, allocating guests according to the input guest preferences. The task would be complete when the seating plan is checked and approved by the client. The resources, programming, and algorithmic experience are available, and the need to simplify this component of event planning (Seat allocation) is high. The program's input will be taken using guests' seating preferences by digital invitations, as well as clients' preferences.

II. Methodology

A. OOP vs POP (Procedural Programming)

OOP takes a problem, and divides it into smaller objects, while POP would take the same problem, and divide it into smaller functions. POP was the initial, common, way of doing things, and the examples of seat allocation programs we have seen, are done in OOP. The issue with POP, and other programming paradigms, when it comes to an application such as a seat allocation one, is that it is not easy to alter options, guest preferences, or the guests themselves. However, in OOP, for example, because we know which guest is related to which seat and invitation, we can easily access that object anytime to get or set information.

Another reason is that due to there being an object to store information in, the code involves more modifications to the object rather than endless "if" statements. Thus, that reduces the amount of the code written. Inheritance, of classes and objects, and encapsulation, hiding data, are also two of the other strong points of OOP. People tend to avoid complicated looking problems and interfaces, so having our program look and be easy to use was important to us.

B. Rational Unified Phases (RUP)

RUP is a software developmental method, such as the waterfall model, and it is meant to guide you in the stages of developing software. It is use-case driven, agile, and iterative, which aided us during this project since it was divided to two parts: Design Report, and Case Study. RUP has the following 4 main classes:

- Inception
- Elaboration
- Construction
- Transition

Since it is an agile method, meaning that the different parts of the method can be adjusted and readjusted throughout. The following model clarifies what each phase comprises, and we have found it true for us:

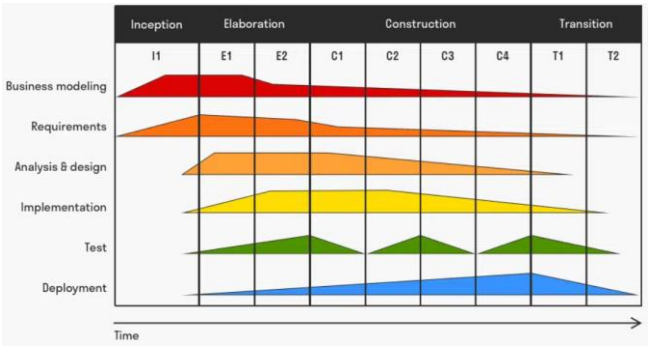


Fig 1. Picture taken from [1]

C. Functional and Non-Functional Requirements

Functional Requirements:

- System takes input from User
- System uses input taken from user to prepare guest list
- System connects to server to send invitations in emails to guests
- System uses input to allocate guests and produce seating plan

- System outputs seating plan to User

Non-functional Requirements:

- Security
- Technology Architecture
- System Management
- Development Standards
- On-going Support and Maintenance

D. The Seat Allocation program – in context

To put things into perspective, we'll start with a system context diagram:

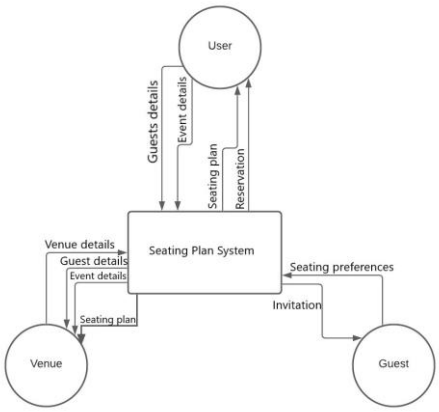


Fig 2. System Context Diagram

We see how the guest interacts with the system clearly here, receives an invitation and inputs their seating preferences. We also see how the user provides the main information to the system, and it returns the seating plan and reservation. Finally, the venue here provides the details, and the seating plan system provides it with the guest and event details to organize the venue, decorate it, and have an invitation checking team.

In the next section, we will take a look at the key use cases, and the structure of our program through various diagrams.

III. A Closer Look

A. Key Use Cases

The use case is meant to show how the user interacts or should interact with the system from the user's point of view, and this also acts as requirements to the developer to accommodate all possible scenarios.

- Brief Description

This Use Case starts when the user runs the program, and ends when the user approves the seating plan produced.

-Actors

Besides the system, the primary actor is the user of the system who can either be the client themselves, or the event planner. The secondary actor is the event guest.

-Pre-conditions

- User runs system
- User must have venue, date, name of event, the list of guests with their emails
- System checks that venue is free on the date input
- System must be connected to the internet
- User should have an invitation template to use, or the default is used

-Basic Flow/Main Success Scenario

1. User starts up program
2. User enters required details, such as venue, date, and name of event, when prompted by system
3. User enters list of guests, and their emails, when prompted by the system
4. User enters preferred invitation template, and any preferences, when prompted by the system.
5. System asks for set date allowed for guests to enter their preferences.
6. System fills in invitation template with the details
7. User confirms invitations.
8. System sends out invitations to Guests
9. All Guests add their preferences in the invitations, before set date
10. System records Guests' preferences
11. System analyses Guests' preferences, any user preferences, and possible seat arrangements in the venue.
12. System outputs seating plan
13. User approves seating plan
14. Tickets, including guests' seats, are sent out to guests to be used on the day of the event.

-Alternate Flows

4a. User does not enter invitation template: System uses default invitation template already available to system.

6a. Some guests don't add preferences before set date: System records their preferences as none.

13a. User does not approve seating plan: System prompts user if they would like to add more preferences. If User adds more preferences: System records additional preferences and repeats steps from 9 – 11. If User doesn't add more preferences: System repeats steps 9 – 11.

-Exceptions

2a. Venue is not free on that day: System terminates with the message that the venue is not free on that day. Use Case Ends.

7a. User does not confirm: System terminates. Use Case Ends.

13b. User aborts the Use Case: The System terminates. Use Case Ends.

-Post-condition

Program is terminated until started by user.

-Diagram

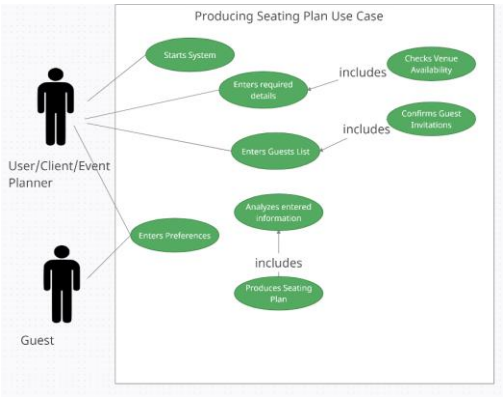


Fig 3. Use Case Diagram

B. *System Sequence Diagram*

The system sequence diagram brings into consideration a very important component: time. It shows us how the objects in the system interact with each other over time, and this helps us visualize and understand the sequence of actions and interactions, hence it is called a system sequence diagram.

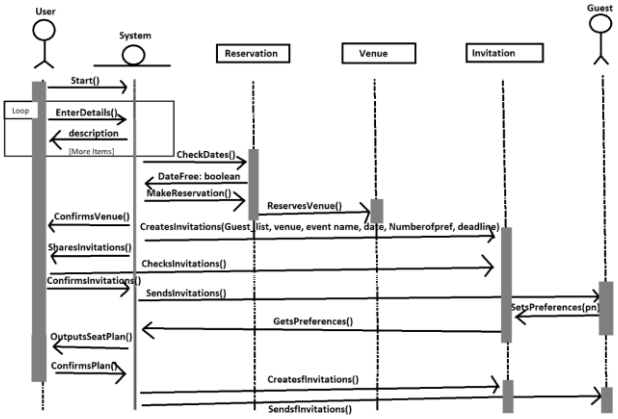
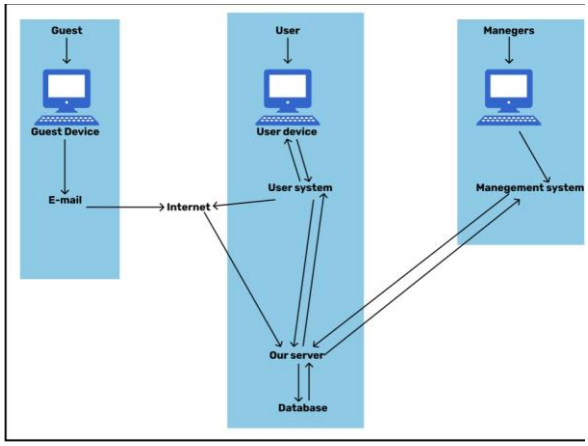


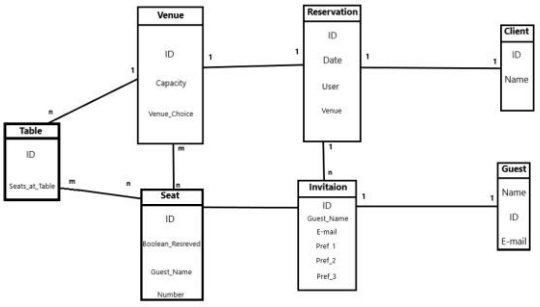
Fig 4. System Sequence Diagram

C. *Logical Architecture Diagram*

The logical architecture diagram, on the other hand, shows us how the system is from the backend, how it interacts with databases, servers, and saved data that is part of the preconditions in the use case, etc.

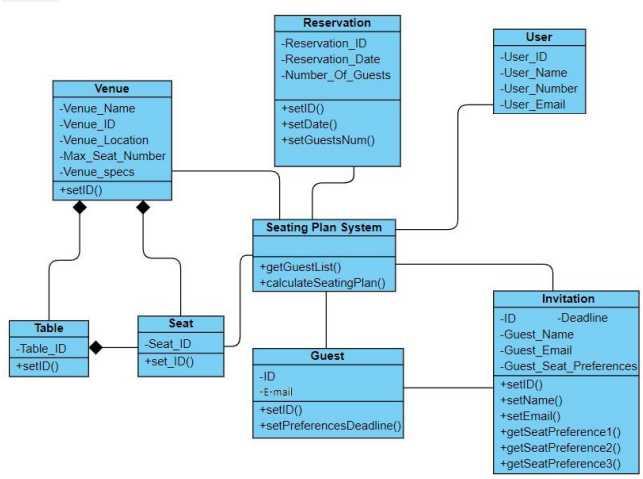


D. *Class Diagrams*



The purpose of this partial domain model is to represent how a real-world example of this case would seem as a model drawn out. In our system we can see that a client is connected to reservation, since the client makes the reservation, and they reserve a venue which is why reservation relates to venue. Furthermore, the venue has tables and seats, and tables have seats, and seats are at tables in a venue, so we can see why there is that connection in the diagram between them.

After a reservation, an invitation uses the details from the reservation, and seat, as well as the guest-details it has, to send it to the guest. The guest receives the invitations, confirms, and lists their preferences to be used in our system. The Class diagram we developed from this one to map the classes to object in our code is the following:



IV. Results And Discussion

The project was coded in the programming language java on Eclipse IDE, on 64-bit operating system, x64-based processor.

A. *Results*

The code was tested using different guest lists, and preferences combinations, and produced a good seating plan. When there was only one person preferring a certain seat, the program would allocate it to them. However, if it was more than one, then it would use the algorithm we designed to decide between the choices and make sure that most, if not all, of the guests could be allocated to their second seat. The program succeeded in fulfilling the following objectives:

- Testing the use of OOP in a simple application, specifically seat allocation
- Benefitting from the iterations in the UP phases to output the best result

- Learning how to code an efficient algorithm
- Providing the knowledge for improving our use of OOP

The results were much better than we expected; we have successfully solved a problem with OOP, and are working towards designing an interface.

B. Discussion

From the four RUP phases, I will list what we mainly did in each phase, and our experience in it:

- Inception: Modeling, Scoping the project out, outlining the requirements
- Elaboration: Revising the requirements and editing them, analysis and design, drawing out the diagrams that's also part of the modeling, planning the application's objects
- Construction: Mainly implementing the objects, and working on the seat allocation algorithm, updating the diagrams, Testing if the parts initially coded work
- Transition: Testing the complete code, and preparing it for deployment by editing it and making the code cleaner, making sure it fulfills all the requirements

We were surprised how modeling and requirements were covered and iterated over during almost all phases of the RUP phases. We spent a lot of the inception period brainstorming, sharing ideas, searching the internet, and discussing the idea and the possible requirements until we settled on the ones listed in this project. We also discussed it with Dr Piyush before beginning it. His intel on it was useful, and very helpful.

For Elaboration, we enjoyed making the different diagrams since we didn't have experience with some of them before, so learning why each one is used, how, and when, made us gain valuable knowledge, and gave us more hands-on experience with software development.

It was the first time for us to try making a partial domain model, and we learned a lot in reading about how it differs from the class diagram which we had done before. Constructing the system context diagram was a bit difficult since we didn't have previous experience with it too. Both these diagrams we discussed, and drafted, repeatedly.

Planning, and drawing the system context diagram was very interesting. Seeing how all the components in the system would interact together was very useful in helping us visualize it, and visualize the next step, implementing it. The use cases also helped in that, they highlighted the alternatives we need to accommodate, and any exceptions too.

The logical architecture diagram was like a back door to the program; how the databases interact, and it completed our system. We had previous experience constructing the logical architecture diagram, so we felt that we were able to use that experience when we designed this diagram, and that it added to our experience since it was a new application to us.

On the other hand, construction was fun and we were comfortable with using java since we have developed different software with it before. The algorithm was a bit of a challenge since there were a lot of variables to take into consideration. However, we managed to do it, and make it work. It definitely requires some more work since we are aiming to make it into an AI program instead. We think the success rate for that would be more accurate, and fluid, since there are a couple of scenarios we can't take into account with our rule-based algorithm. We made the algorithm with the assumption that all the first preferences aren't null, so that's one thing we want to improve.

Finally, for the deployment phase, we cleaned the output to make it presentable for submission, and for our live demo. We also revised it fully in the writing of this report, the diagrams, previous content, and how the code is a mapping of our requirements. The last thing I wanted to add here is that we checked the risks on our project by developing a risk management plan that I share here:

Identify Risks

A. IT security threats

B. Employee quitting

C. Meeting security requirements

D. Server crashing

Measure Risks

SCALE OF SEVERITY				
		ACCEPTABLE	TOLERABLE	UNACCEPTABLE
SCALE OF LIKELIHOOD	NOT LIKELY			D
	POSSIBLE	A	B	
	PROBABLE			C

Examine solutions

A. Ensure all data is backed-up regularly. Install a firewall to protect the network. If it fails, excuse multi-step control of access to the system.

B. Provide Employees with healthy working environment. Set clear quarterly and yearly goals. the contract a notice period that allows to find a replacement to maintain stability.

C. Limit the spread of classified information. Creating a private network with multiple clearances for different types of data. Use blockchain to protect data.

D. Check server logs to determine the fault and take action / have backup server running.

Implement solutions

Implementing the solutions includes having all these risks in mind at all times with the backup plan ready to be implemented at any given moment. Test runs and bug fixes will be done regularly as well. Hire employees that are well-versed and highly skilled at their job, as well as providing them with training based on the risk management plan.

IIV. CONCLUSION

This paper found that using OOP for seat allocation was a success. This application can definitely be improved using GUI, and AI modules, and deployed in the market. We were also delighted to experience working on a project for 9 weeks – a mini-capstone if I may say so. It must be the longest project we have worked on so far. We are happy we have achieved

our objectives, of trying the RUP phases, teamwork, and learning about the many different diagrams.

REFERENCES

[1] [1] "Rational Unified Process (RUP)", *toolshero*, 2022. [Online]. Available: https://www.toolshero.com/information-technology/rational-unified-process-rup/. [Accessed: 09- Mar- 2022].