GALGOTIAS UNIVERSITY

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

# AUTOMATIC CAR PARKING SYSTEM

**A Report for the Evaluation 3 of Project 2**

***Submitted by***


**ANSHUMAN DWIVEDI**

**(1613101155)**


**in partial fulfillment for the award of the degree**

**of**

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**SCHOOL OF COMPUTING SCIENCE AND ENGINEERING**

**Under the Supervision of**

**Ms. Anisha, M.Tech, MCA**

**Professor**

**APRIL / MAY- 2020**


# SCHOOL OF COMPUTING AND SCIENCE AND ENGINEERING

## BONAFIDE CERTIFICATE

**Certified that this project report "AUTOMATIC CAR PARKING SYSTEN" is the bonafide work of "ANSHUMAN DWIVEDI (1613101155)" who carried out the project work under my supervision.**

**SIGNATURE OF HEAD** **SIGNATURE OF SUPERVISOR**

**Dr. Raju Shanmugam,**

**PhD (CS), ME(CS)**

**Professor & Dean,**

**School of Computing Science & Engineering**

**Ms. Anisha, M.Tech.,**

**MCA**

**Assistant Professor**

**School of Computing Science & Engineering**

## ABSTRACT

The Automatic car Parking System had been conceived with the view to automate the manual work-flows involved in the management of car Parking Lots. It drastically reduces the effort, inaccuracies, error-prone tendencies, delays and overheads involved in performing the same tasks by hand.

It was aimed to provide a fully automated system that was capable of checking in and out of cars entering and exiting the designated parking lot, and recording relevant information about them. Revenue calculation and data entry is automated to the largest possible extent. Only minimum intervention from the manual user is required, with a possibility of eliminating it altogether with further advancement in the technology encompassed by the project, and supporting hardware.

A rich and easy-to-use GUI aids the user in navigating the system easily and comprehensively. Features such as multiple searching and viewing options further add to the capabilities of the system and thereby also help in reducing the entry time. Transactions concurrency and their unambiguous nature have been carefully balanced and user sessions are purposefully managed. Direct implementation of the printing code helps

the entry clerks as well as managers/administrators, to print the parking slips, reports and user information as and when required.

## TABLE OF CONTENT

# 1. Introduction

## 1.1 Purpose:

The purpose of the report document is to specify all the information required to design, develop and test the software. The purpose of this project is to provide a friendly environment to maintain the details of Car and Employers of the Parking area.

The main purpose of this project is to maintain easy circulation system using computers and to provide different reports.

## 1.2 Definition, Acronyms, Abbreviation:

JAVA -> Platform independence

SQL -> Structured query Language

DFD -> Data Flow Diagram

CFD -> Context Flow Diagram

ER -> Entity Relationship

IDE -> Integrated Development Environment

SRS -> Software Requirement Specification

## 1.3 Overview:

Everyone who owns or drives a car in India or abroad would be all too familiar with the hassles of finding parking spaces, misbehaving parking attendants, inconsistent or monopolized rates and other problems associated with it.

What is proposed here, is not just another automation of a manual workflow system, it can also be viewed as a solution to the aforementioned problems of the everyday consumer. Rise to the occasion, an Automated car Parking System.

It not only rids the car owner from the hassles of finding parking spots, it ensures that there is never over or under accommodation of cars beyond the lot's capacity. The system completely eliminated even the possibility of embezzlements. The rates are fixed and predefined. No tampering can be done with the automated calculation of the revenue based on the time taken directly from the console.

# 2. System Analysis

## 2.1 Existing System:

Some of the problems being faced in Existing system are as follows:

1. Fast report generation is not possible.

2. Tracing a car is difficult.

3. Information about entry/exit of the cars are not properly maintained.

4. No central database can be created as information is not available in database.

## 2.2 Proposed System:

The proposed Car Parking System will take care of the current Car detail at any point of time. The Car entry, Car exit will update the current car details automatically so that user and employers of the parking area will get the update of current ongoing parking details.

## 2.3 Feasibility study:

The overall scope of the feasibility study was to provide sufficient information to allow a decision to be made as to whether the Car Parking System project should proceed and if so, its relative priority in the context of other existing car parking Technology.

The feasibility study phase of this project had undergone through various steps which as describe as under:

- Identity the origin the information at different level.
- Identity the expectation of user from computerized system.

- Analyze the drawback of existing system(manual system)

### 2.4 Assumption and dependencies:

All the data entered will be correct and up to date. This software package is developed using java as front end which is supported by sun micro system. Microsoft SQL server as the back end.

## 3. Specific Requirements

### 3.1 External Interface Requirement:

The user should be simple and easy to understand and use. Also be an interactive interface .The system should prompt for the user and administrator to login to the application and for proper input criteria.

#### 3.1.1 User Interface:

The software provides good graphical interface for the user any administrator can operate on the system, performing the required task such as create, update, viewing the details of the car.

- Allows user to view quick reports like car Entry/Exit etc in between particular time.
- Stock verification and search facility based on different criteria.

#### 3.1.2 Hardware Interface:

- Operating system : Windows
- Hard disk :160 GB
- RAM : 512 MB
- Processor : Pentium(R)Dual-core CPU

### 3.1.3 Software Interface:

- Java language
- Net beans IDE
- Oracle DB server

### 3.1.4 Communication Interface:

Window

## 3.2 Functional requirements:

- car entry: In this module we can store the details of the car.
- Manage Users: In this module we can manage all users.
- car entry: This module is used to keep a track of car entry details.
- car exit: This module enables to keep a track of exiting the car.

## 3.3 Performance requirements:

The capability of the computer depends on the performance of the software. The software can take any number of inputs provided the database size is larger enough. This would depend on the available memory space.

## 3.4 Design constraints:

Each User will be having a Parking receipt which can be used for the car entry, fine payment etc. whenever car member wish to take a car, the car entry by the car authority will be check both the car details as well as the user details and store it in car database. In case of retrieval of car much of human intervention can be eliminated.

## 3.5 System attributes:

**3.5.1 Maintainability**: There will be no maintained requirement for the software. The database is provided by the end user and therefore is maintained by this user.

**3.5.2 Portability**: The system is developed for secured purpose, so it is can't be portable.

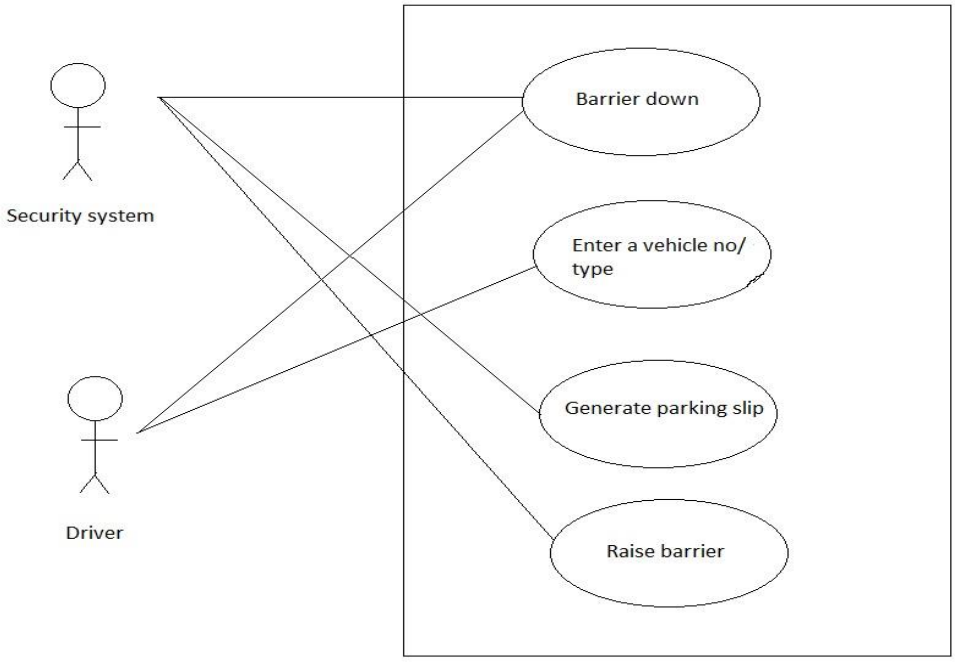
**3.5.3 Availability**: This system will available only until the system on which it is install, is running.

**3.5.4 Scalability:** Applicable.

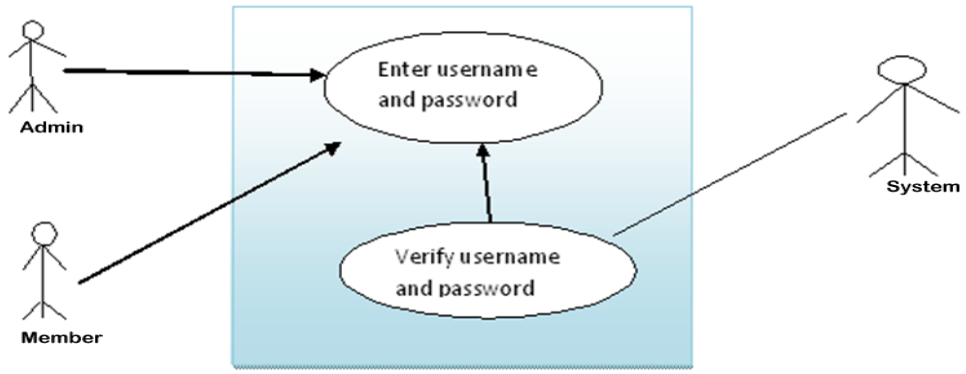
# 4. Analysis and Design

## 4.1 Use Case Diagram:

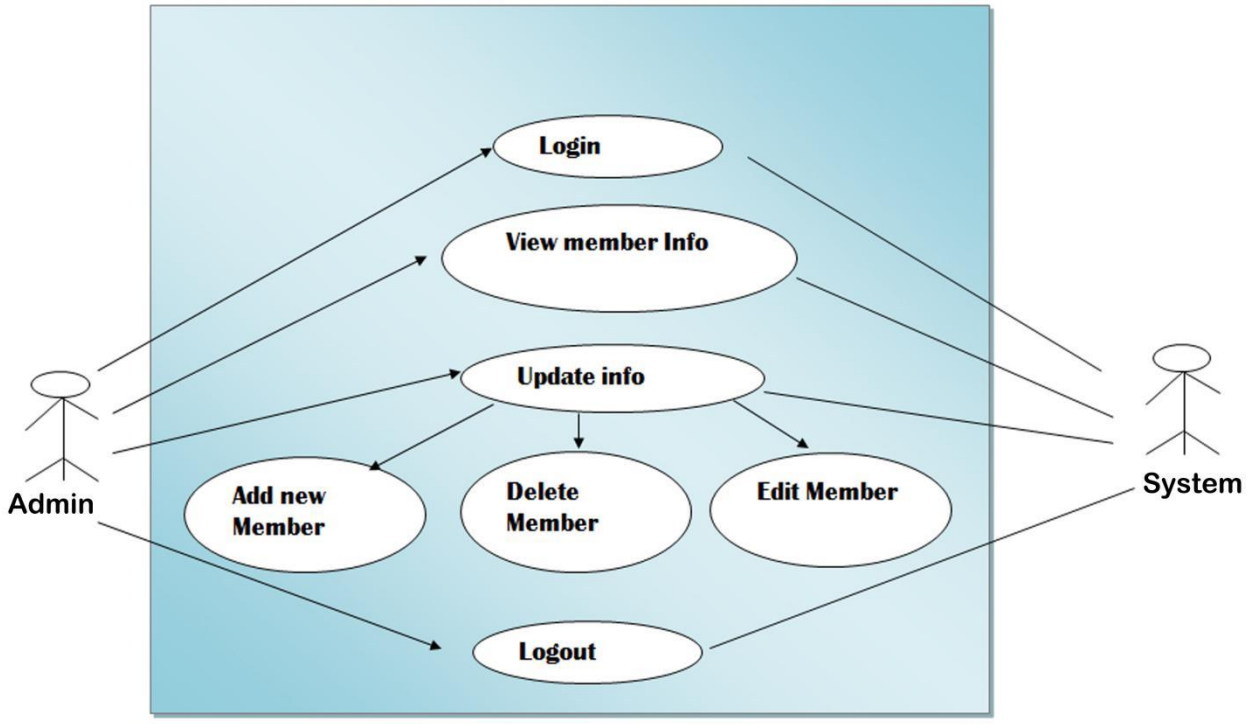
### 4.1.1 Use Case Overview:




### 4.1.2 Login User Case:




### 4.1.3 Manage Member Information Use Case:




## 4.2 Activity Diagram:

Activity diagrams are graphical representations of work-flows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-

by-step work-flows of components in a system. An activity diagram shows the overall flow of control.

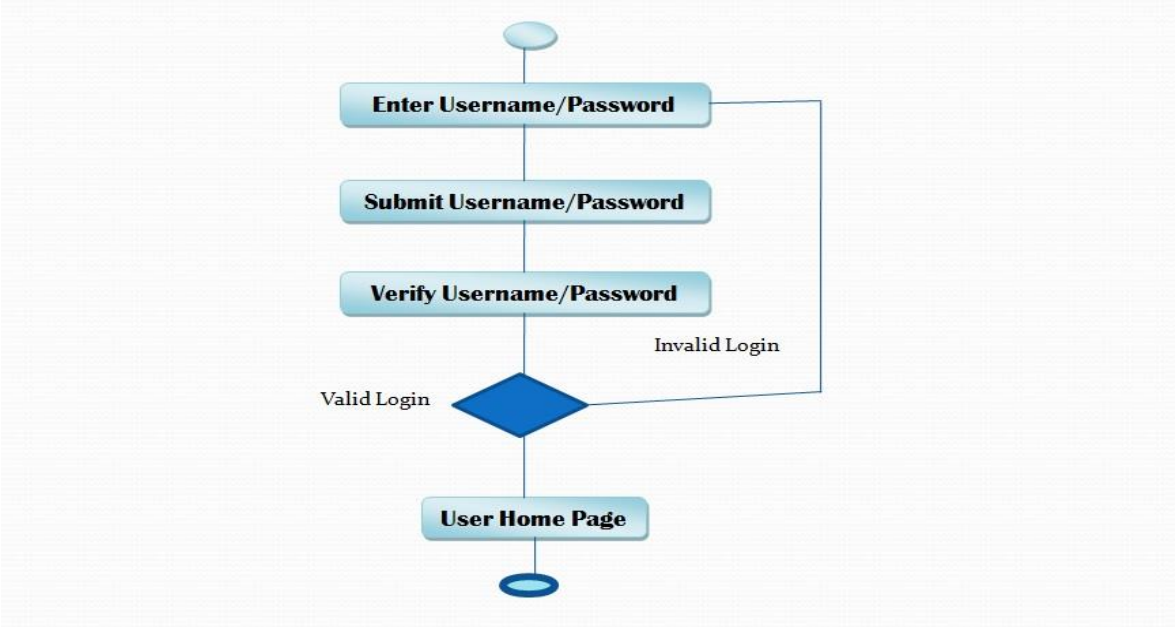
Activity diagrams are constructed from a limited repertoire of shapes, connected with arrows. The most important shape types:

- rounded rectangles represent activities;
- diamonds represent decisions;
- bars represent the start (split) or end (join) of concurrent activities;
- a black circle represents the start (initial state) of the work-flow;
- an encircled black circle represents the end (final state).

Arrows run from the start towards the end and represent the order in which activities happen.

Hence they can be regarded as a form of flowchart. Typical flowchart techniques lack constructs for expressing concurrency. However, the join and split symbols in activity diagrams only resolve this for simple cases; the meaning of the model is not clear when they are arbitrarily combined with decisions or loops.

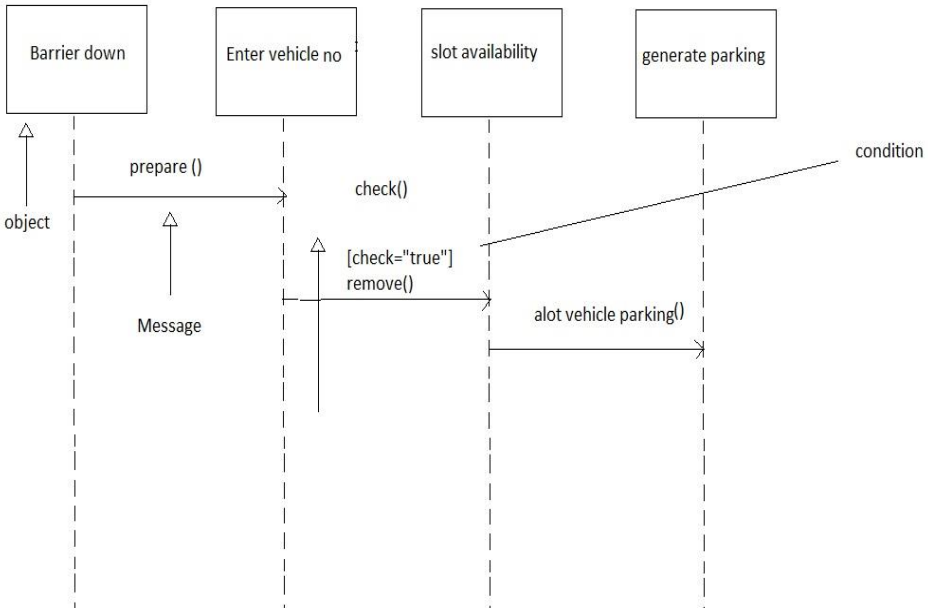
### 4.2.1 Login Activity Diagram:




## 4.3 Sequence Diagram:

A sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message sequence chart. A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur.

### 4.3.1 Sequence Diagram Overview:




.

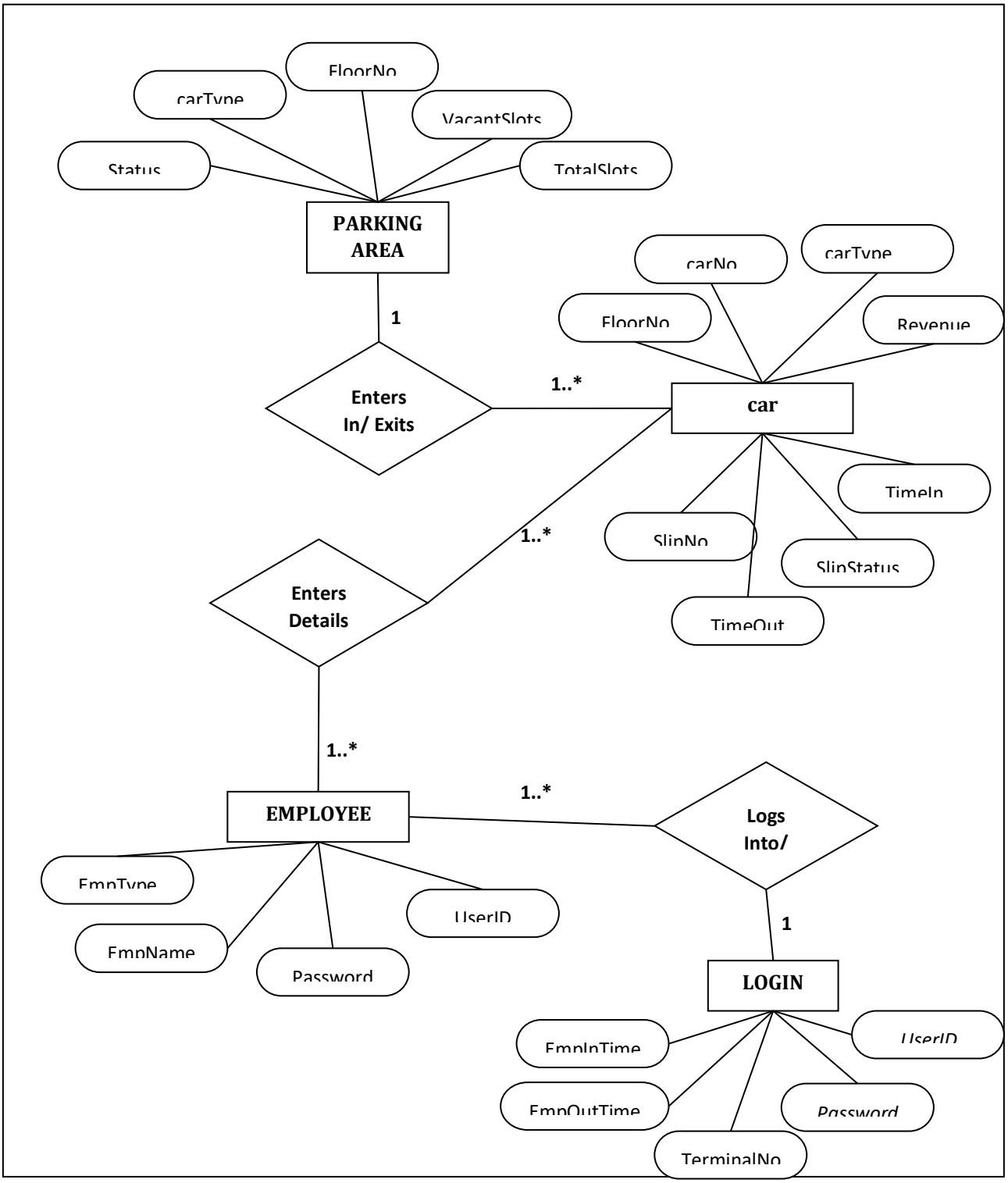
## 4.4 ER Diagram:

It is clear that the physical objects from the previous section – the member, cars, and car – correspond to entities in the Entity-Relationship model, and the operations to be done on those entities – holds, checkouts, and so on – correspond to relationships.

However, a good design will minimize redundancy and attempt to store all the required information in as small a space as possible.

### 4.4.1 ER Diagram Overview:

.



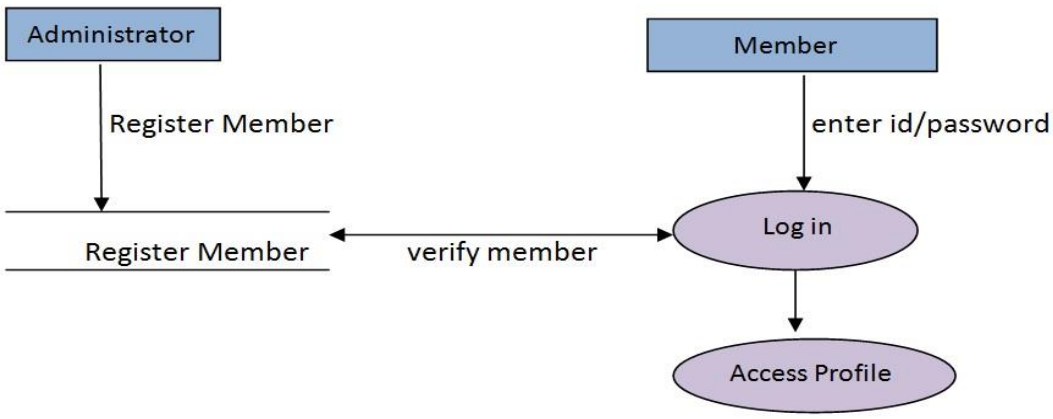

## 4.5 Data Flow Diagram (DFD):

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system. DFDs can also be used for the visualization of data processing (structured design).

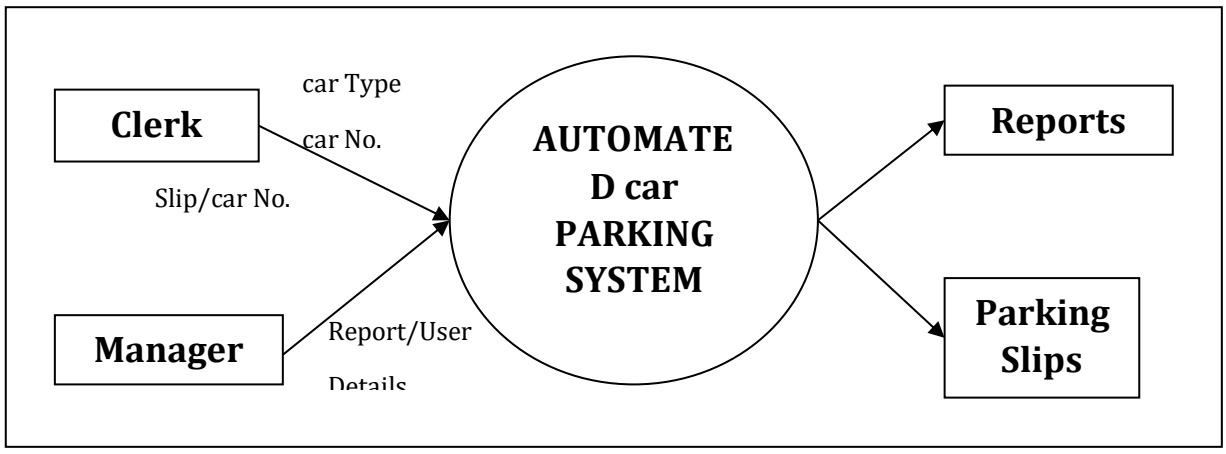
On a DFD, data items flow from an external data source or an internal data store to an internal data store or an external data sink, via an internal process.

A DFD provides no information about the timing of processes, or about whether processes will operate in sequence or in parallel. It is therefore quite different from a flowchart, which shows the flow of control through an algorithm, allowing a reader to determine what operations will be performed, in what order, and under what circumstances, but not what kinds of data will be input to and output from the system, nor where the data will come from and go to, nor where the data will be stored (all of which are shown on a DFD).

### 4.5.1 Login Process:




### 4.5.2 Working Process



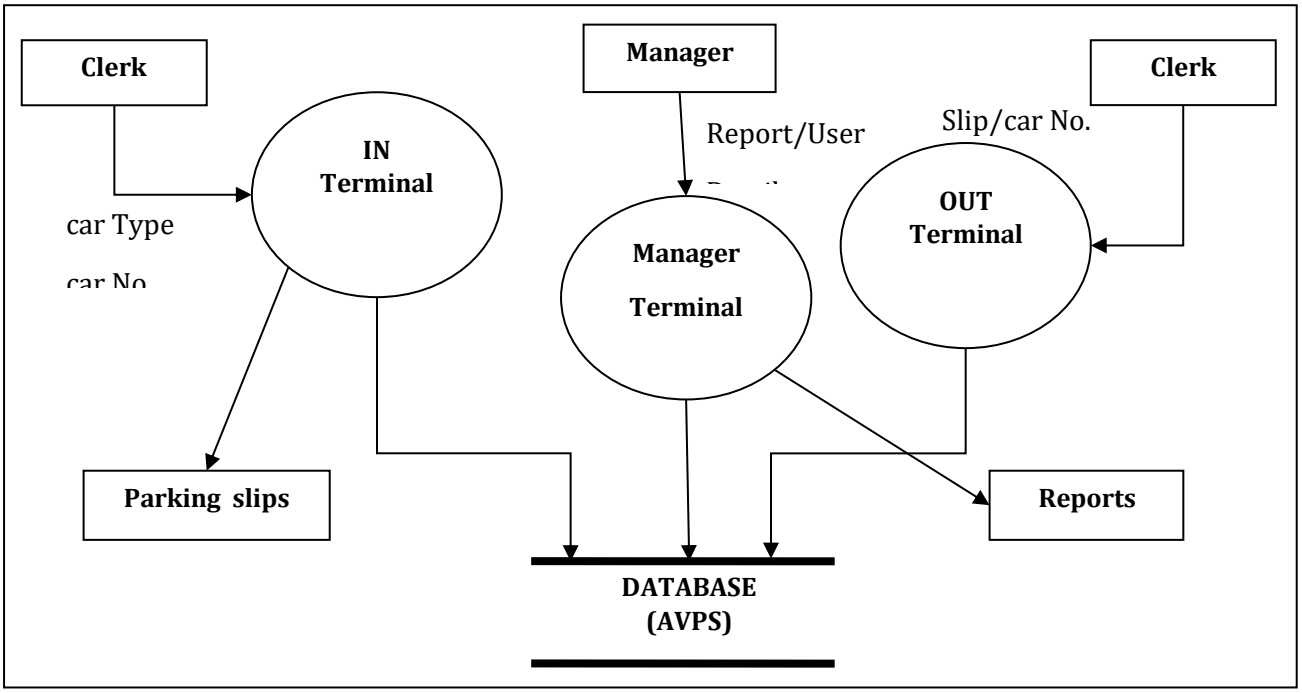
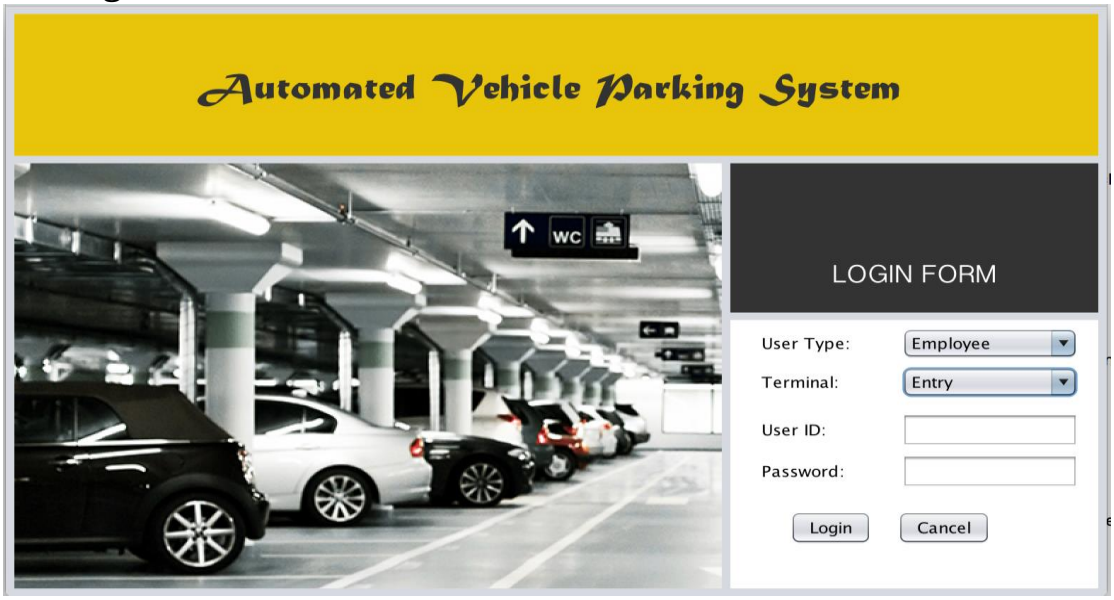



Fig: 0 Level DFD

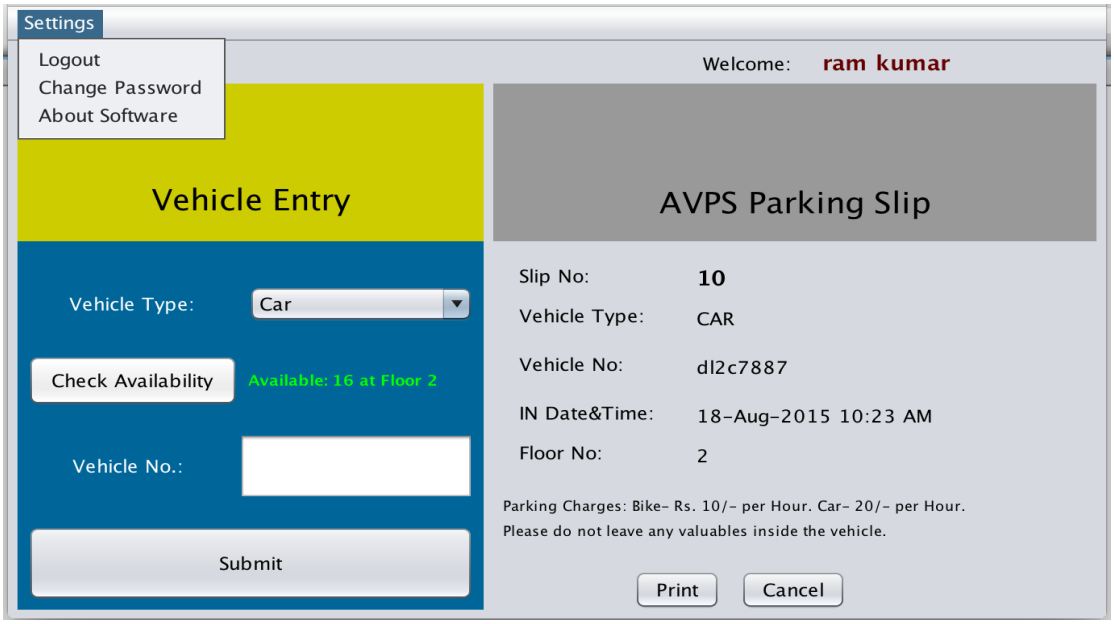
1 Level DFD

# 5. User Interface Implementation

## 5.1 Login:



### 5.2 Employee Login Page:




# 6. Testing

### 6.1 Testing Objective:

The main objective of testing is to finding a host of errors, systematically and with minimum effort and time. We can say as follows:

- Testing is a process of executing a program with the intent of finding an error.
- A good test case is one that has a high probability of finding error, if it exists.
- The tests are inadequate to detect possibly present errors.
- The software should be based on the quality and reliable standards.

### 6.2 Unit testing:

Unit testing focuses verification effort on the smallest unit of software i.e. the module. Using the detailed design and the process

specifications, testing is done to uncover errors within the boundary of the module. All modules must be successful in the unit test before the start of the integration testing begins.

In this project each service can be thought of a module. There are so many modules like administrator,user,visitor. Each module has been tested by giving different sets of inputs. When developing the module as well as finishing the development, the module works without any error. The inputs are validated when accepting them from the user.

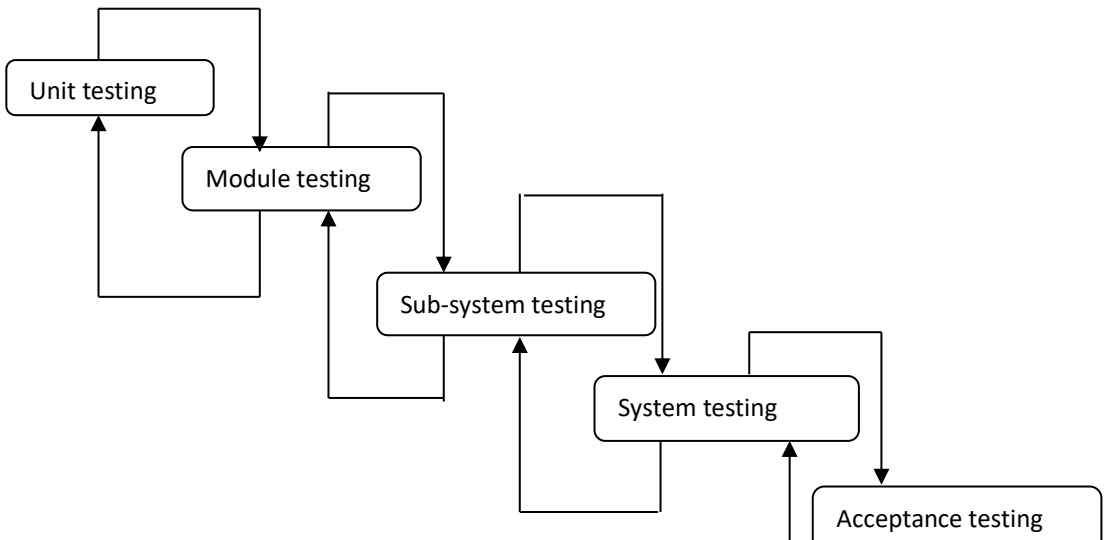
### 6.3 Integrated testing:

After unit testing, we have to perform integration testing. The goal here is to see if modules can be integrated properly, the emphasis being on testing interfaces between modules. This testing activity can be considered as testing the design and hence the emphasis on testing module interactions.

In this project the main system is formed by integrating all the modules. When integrating all the modules I have checked whether the integration effects working of any of the services by giving different combinations of inputs with which the two services run perfectly before integration.

### 6.4 System testing:

System testing involved is the most widely used testing process consisting of five stages as shown in the figure. In general, the sequence of testing activities is component testing, integration testing, and then user testing. However, as defects are discovered at any one stage, they require program modifications to correct them and this may require other stages in the testing process to be repeated.



**(Component testing) (Integration testing)**
**(User testing)**

**Fig. show Prototype model**

Testing is the process of detecting errors. Testing performs a very critical role for quality assurance and for ensuring the reliability of the software. The results of testing are used later on during maintenance also.

Testing is vital to the success of the system. System testing makes a logical assumption that if the parts of the system are correct, the goal will be successfully achieved. In adequate testing or non-testing leads to errors that may not appear until months or even years later. This creates two problems:

- The time lag between the cause and the appearance of the problem.
- The time interval effect of the system errors on files and the records on the system.

A small error can conceivably explode into a much larger problem. Effective testing early in the process translates directly into long term cost savings from a reduced number of errors.

Another reason for system testing is its utility as a user oriented car before implementation. The best program is worthless if it does not meet the user requirements. Unfortunately, the user's demands are often compromised by efforts to facilitate program or design efficiency in terms of processing time or design efficiency.

Thus in this phase we went to test the code we wrote. We needed to know if the code compiled with the design or not? Whether the code gave the desired outputs on given inputs? Whether it was ready to be installed on the user's computer or some more modifications were needed?

Through the web applications are characteristically different from their software counterparts but the basic approach for testing these web applications is quite similar. These basic steps of testing have been picked from software engineering practices. The following are the steps, we undertook:

The content of the Intranet site is reviewed to uncover content errors. Content errors covers the typographical errors, grammatical errors, errors in content consistency, graphical representation and cross referencing errors.

The design model of the web application is reviewed to uncover the navigation errors. Use cases, derived as a part of the analysis activity allows a web designer to exercise each usage scenario against the architectural and navigational design. In essence these non-executable tests help to uncover the errors in navigation.

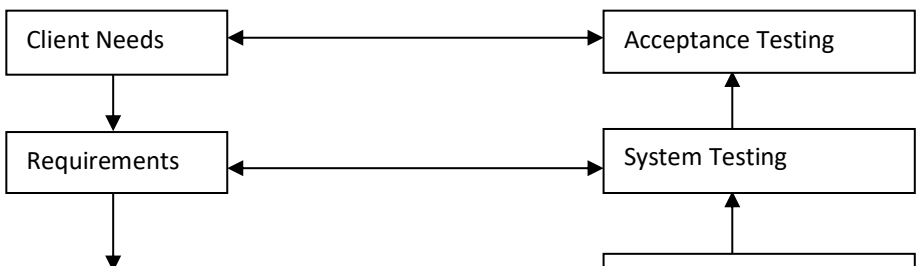
When web applications are considered the concept of unit changes. Each web page encapsulates content navigation links, content and processing elements. It is not always possible to test each of these individually. Thus is the base of the web applications the unit to be considered is the web page. Unlike the testing of the algorithmic details of a module the data that flows across the module interface, page level testing for web applications is driven by content, processing and links encapsulating the web page.

The assembled web application is tested for overall functionality and content delivery. The various user cases are used that test the system for errors and mistakes. The web application is tested for a variety of environmental settings and is tested for various configurations and upon various platforms.

The modules are integrated and integration test are conducted. Thread based testing is done to monitor the regression tests so that the site does not become very slow is a lot of users are simultaneously logged on.

Levels of Testing

In order to finding the errors present in different phases, we have the concept of levels of testing. The basic levels of testing are



## 6.5 Acceptance testing:

Acceptance Testing is performed with realistic data of the client to demonstrate that the software is working satisfactorily. Testing here is focused on external behavior of the system of the internal logic of program is not emphasized.

Test cases should be selected so that the largest number of attributes of an equivalence class is exercised at once. The testing phase is an important part of software development. It is the process of finding errors and missing operations and also a complete verification to determine whether the objectives are met and the user requirements are satisfied.

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing also provides an objective, independent view of the software to allow the business to appreciate and understand the risks at implementation of the software. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs.

Software testing can also be stated as the process of validating and verifying that a software program/application/product:

1. meets the business and technical requirements that guided its design and development;
2. works as expected; and
3. can be implemented with the same characteristics

**Functional Vs Non-functional Testing**

Functional testing refers to tests that verify a specific action or function of the code. These are usually found in the code requirements documentation, although some development methodologies work from use cases or user stories. Functional tests tend to answer the question of "can the user do this" or "does this particular feature work".

Non-functional testing refers to aspects of the software that may not be related to a specific function or user action, such as scalability or security. Non-functional testing tends to answer such questions as "how many people can log in at once", or "how easy is it to hack this software".

**Static Vs Dynamic Testing**

There are many approaches to software testing. Reviews, walkthroughs, or inspections are considered as static testing, whereas actually executing programmed code with a given set of test cases is referred to as dynamic testing. Static testing can be

(and unfortunately in practice often is) omitted. Dynamic testing takes place when the program itself is used for the first time (which is generally considered the beginning of the testing stage). Dynamic testing may begin before the program is 100% complete in order to test particular sections of code (modules or discrete functions). Typical techniques for this are either using stubs/drivers or execution from a debugger environment. For example, spreadsheet programs are, by their very nature, tested to a large extent interactively ("on the fly"), with results displayed immediately after each calculation or text manipulation.

**Software Quality Assurance (SQA)**

Though controversial, software testing may be viewed as an important part of the software quality assurance (SQA) process. In SQA, software process specialists and auditors take a broader view on software and its development. They examine and change the software engineering process itself to reduce the amount of faults that end up in the delivered software: the so-called defect rate.

What constitutes an "acceptable defect rate" depends on the nature of the software. For example, an arcade video game designed to simulate flying an airplane would presumably have a much higher tolerance for defects than mission critical software such as that used to control the functions of an airliner that really is flying!

Although there are close links with SQA, testing departments often exist independently, and there may be no SQA function in some companies.

Software testing is a task intended to detect defects in software by contrasting a computer program's expected results with its actual results for a given set of inputs. By contrast, QA (quality assurance) is the implementation of policies and procedures intended to prevent defects from occurring in the first place.

## Testing Methods

### The Box Approach

Software testing methods are traditionally divided into and white- and black-box testing. These two approaches are used to describe the point of view that a test engineer takes when designing test cases.

### White Box Testing

White box testing is when the tester has access to the internal data structures and algorithms including the code that implement these. White box testing methods can also be used to evaluate the completeness of a test suite that was created with black box testing methods. This allows the software team to examine parts of a system that are rarely tested and ensures that the most important function points have been tested.

### Black Box Testing

Black box testing treats the software as a "black box"—without any knowledge of internal implementation. Black box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, fuzz testing, model-based testing, traceability matrix, exploratory testing and specification-based testing.

### Grey Box Testing

Grey box testing (American spelling: gray box testing) involves having knowledge of internal data structures and algorithms for purposes of designing the test cases, but testing at the user, or black-box level. Manipulating input data and formatting output do not qualify as grey box, because the input and output are clearly outside of the "black-box" that we are calling the system under test. This distinction is particularly important when conducting integration testing between two modules of code written by two different developers, where only the interfaces are exposed for test. However, modifying a data repository does qualify as grey box, as the user would not normally be able to change the data outside of the system under test. Grey box testing may also include reverse engineering to determine, for instance, boundary values or error messages.

# 7. Implementation, Evaluation & Maintenance

## 7.1 Implementation:

System implementation is the stage when the user has thoroughly tested the system and approves all the features provided by the system. The various tests are performed and the system is approved only after all the requirements are met and the user is satisfied.

The new system may be totally new, replacing an existing manual or automated system, or it may be a major modification to an existing system. In case of, proper implementation is essential to provide a reliable system to meet organizational requirements. Successful implementation may not guarantee improvement in the organization using the new system, but improper will prevent it.

Implementation is the process of having systems personnel check out and put new equipment into use, train users, install the new application and construct any files of data needed to use it. This phase is less creative than system design. Depending on the size of the organization that will be involved in using the

application and the risk involved in its use, systems developers may choose to test the operation in only one area of the firm with only one or two persons. Sometimes, they will run both old and new system in parallel way to compare the results. In still other situations, system developers stop using the old system one day and start using the new one the next.

## 7.2 Evaluation:

The evaluation phase ranks vendor proposals and determines the one best suited. Evaluation of the system is performed to identify its strengths and weaknesses. The actual evaluation can occur along any of the following dimensions:

### 7.2.1 Operational Evaluation

Assessment of the manner in which the system functions, including case of use, response time, overall reliability and level of utilization.

### 7.2.2 Organizational Impact

Identification and measurement of benefits to the organization in such areas, as financial concerns, operational efficiency and competitive impact.

### 7.2.3 Development Performance

Evaluation of the development process in accordance with such yardsticks as overall development time and effort, conformance to budgets and standards and other project management criteria.

## 7.3 Maintenance:

Maintenance or enhancement can be classified as

- Corrective
- Adaptive
- Perfective

Corrective maintenance means repairing processing or performance failures or making changes because of previously uncorrected problems or false assumptions. Adaptive maintenance means changing the program function. Perfective maintenance means enhancing the performance or modifying the programs to respond to the user's additional or changing needs.

Maintenance is actually the implementation of the post implementation review plan. As important as it is, many programmers and analysts are reluctant to perform or identify themselves with the maintenance effort. There are psychological, personality and professional reasons for this. In any case, a first class effort must be made to ensure that software changes are made properly and in time to keep the system in tune with user specifications.

Maintenance is costly. One way to reduce maintenance costs is through maintenance management and software modification audits. Software modification consists of program rewrites system level updates, and re-audits of low ranking programs to verify and correct the soft spots. The outcome should be more reliable software, a reduced maintenance backlog, and higher satisfaction and morale among the maintenance staff.

## 8. Conclusion

From a proper analysis of positive points and constraints on the component, it can be safely concluded that the product is a highly efficient GUI based component. This application is working properly and meeting to all user requirements. This component can be easily plugged in many other systems.

# 9. References


- http://www.java2s.com/
- http://docs.oracle.com/javase/tutorial/java/TOC.html
- Database Programming with JDBC and Java by O'Reilly
- Head First Java
- A Programmer's Guide to Java (Khalid A. Mughal and Rolf w. Rasmussen)
- Sarthak Mendiratta, Debopam Deya and Deepika Rani Sona," Automatic Car Parking System with Visual Indicator along with IoT",IEEE 978-1-5386-1716-8/17/ 2017(Dec) .
- Smart Parking: an Application of optical Wireless Sensor Network, Proceedings of the 2007
- International Symposium on Applications and the Internet Workshops (SAINTW'07), 2007
- A Reservation-based Smart Parking System, The First International Workshop on Cyber-Physical Networking Systems, 2011
- Smart Parking Assist System using Internet of Things (IoT), International Journal of Control Theory and Applications, Volume 9-Number 40,2016