SE-Assignment 4

Team:	**Srn:**
Charvi Bannur	PES1UG20CS638
Kiran SM	PES1UG20CS655
Adithya M	PES1UG20CS621
Gowtham MS	PES1UG20CS642

Problem Statement-1:

Units:

- Login/Signup page
- Profile Page
- Create Ticket
- Quotation request
- View request.

Test Cases:

- Signup:
    - Test case 1 (email format verification):
        - Input email: abcd.com
        - Output: invalid email prompt
    - Test case 2 (email format verification):
        - Input email: abcd@something.com
        - Output: signup successful
    - Test Case 3 (incomplete fields)
        - Input: any empty field
        - Output: incomplete fields prompt

- Quotation request:
    - Test Case 1 (No service mentioned)
        - Output: Invalid request
- Ticket Creation:
    - Test case 1 (options validation)
        - Input: select one or more options
        - Output: Ticket created and updated in the database

Problem Statement –2:

a)Boundary Value Analysis:

Boundary Value Analysis is based on testing the boundary values of valid and invalid partitions.It is Black-box test design technique, which is used to find the errors at

boundaries of input domain(tests the behavior of a program at the input boundaries) rather than finding those errors in the center of input.

It checks for the input values near the boundary that have a higher chance of error.Every partition has its maximum or minimum values(valid and invalid inputs) these values are the boundary values of a partition.

EX:

For a proper email format to signup has following four conditions should be satisfied:

1.It should have unique username

2.It should contain the proper symbol

3.It should have proper mail server

4.It should belong to a specific domain.

So the Boundary condition for the above example are:

It should satisfy all the conditions mentioned above or it can not satisfy all the conditions(which includes even empty set)

If all conditions are satisfied then the mail format will be correct and the user can login or signup else it will not be considered as a proper mail the user can not login.

b)Mutation Testing:

Mutation testing is a form of an white box testing in which testers change specific components of an application's source code to ensure a software test suite will be able to detect the changes .Changes introduced to the software are intended to cause errors in the program.

This method enables the entire source code coverage and detection of parts which are not tested properly.It is possible to detect high-quality bugs,which are problematic to find by usual testing. Mutation helps in evaluating the test conditions and cases that are being used and helps in developing them.

EX:

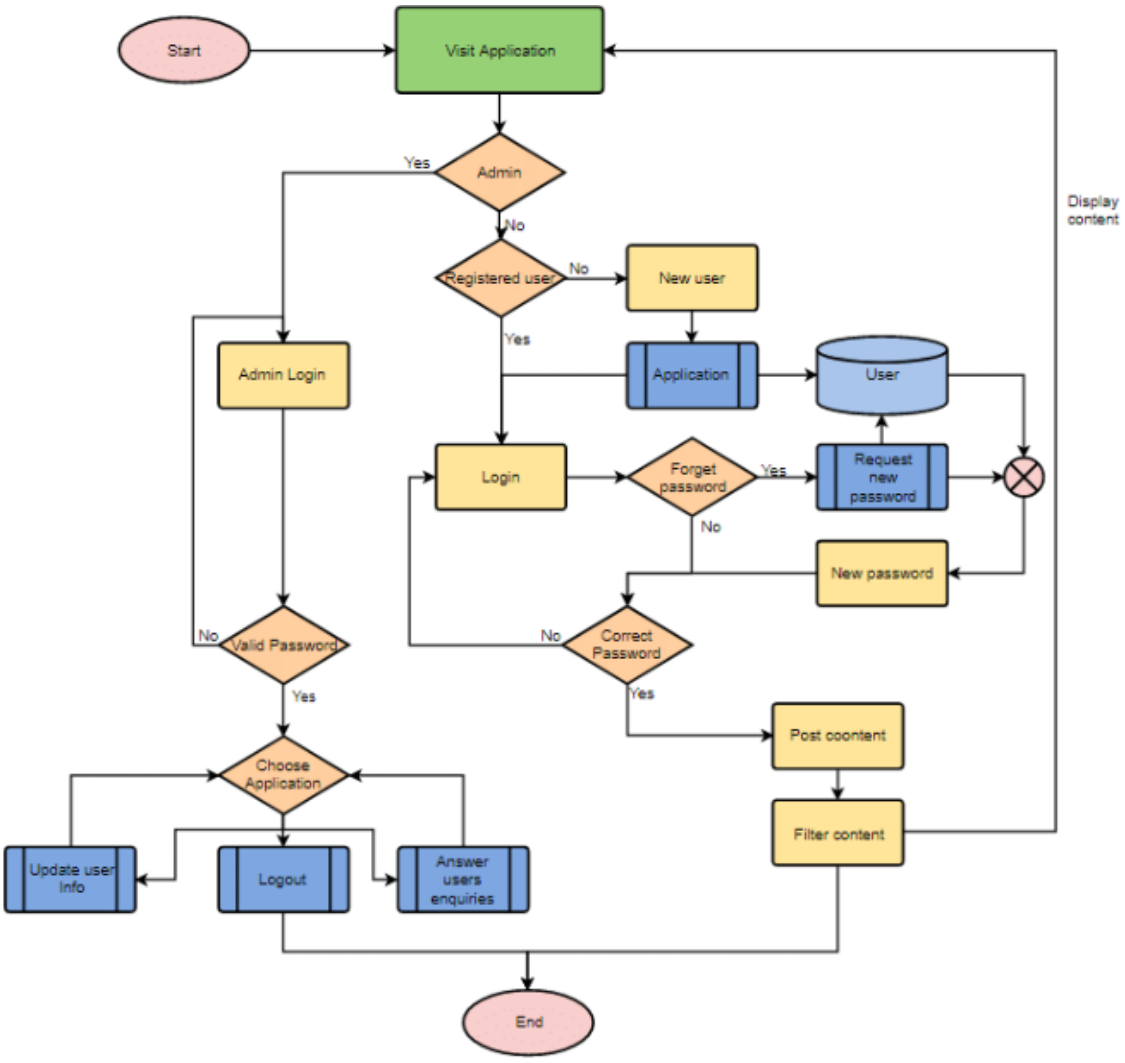
In the example mentioned in the Boundary value analysis we can change the condition from satisfying all four conditions to satisfying three conditions then we run the test cases and for many inputs we will get the wrong output.This shows that our email validation test cases are good and cover all the conditions mentioned.

Problem Statement – 3:

Static Testing is a type of Software Testing method which is performed to check the defects in software without actually executing the code of the software application. It is performed in the early stage of development to avoid errors as it is easier to find sources of failures and it can be fixed easily. The errors that can’t be found using Dynamic Testing, can be easily found by Static Testing.

Cyclomatic Complexity:

The cyclomatic complexity of a code section is the quantitative measure of the number of linearly independent paths in it. It is a software metric used to indicate the complexity of a program. It is computed using the Control Flow Graph of the program. The nodes in the graph indicate the smallest group of commands of a program, and a directed edge in it connects the two nodes i.e. if the second command might immediately follow the first command.



M = E – N + 2P

where,
E = the number of edges in the control flow graph
N = the number of nodes in the control flow graph
P = the number of connected components

N=22
E=28
P=1

According to the flow chart,
M=28-22+2(1)=8

This implies that there are 8 linearly independent paths.
Considering all the possible edge cases to be covered for successful-secured login/ creation of customer's account, the above-mentioned graph cannot be changed. Also, as we know that cyclomatic complexity between 1-10 is considered to be at minimal risk, there is no modification done to the abovementioned cycle and hence the code.

Problem Statement – 4: Acceptance Testing Assume your neighbouring team is the client for your code. Give them an idea of what your product is and the software requirements for the product. • Exchange your code base and test each other's projects to see if it meets user requirements • If you identify a bug in the project you are testing, inform the opposing team of the bug • As a team, based on the client's experience, ideate modifications to the existing project that could improve client experience

After much deliberation, another team proposed significant changes that could be incorporated into the current project. The suggestions mostly revolve around having a better user interface and allowing users to view the existing tickets as part of our user interface. This feature was added and is now one of our primary features. Another feature that was specified was the ability to change the password.

As a result, we improved the user experience by providing these features, allowing the user to proceed smoothly. All user information is saved and maintained.

Problem Statement – 5: Maintenance Activities Once a product is completed, it is handed off to a service based company to ensure all maintenance activities are performed without the added expenditure of skilled developers. However, a few tasks are performed by the maintenance team to gauge the product better. In this problem statement, you will be asked to experiment with your code. • Exchange code bases with your neighbouring teams and reverse engineer a block of code in order to understand its functionality • After understanding the code block, Re-Engineer the code o Ideate how to refactor the code and the portion of the code base you would have to change o Discuss how the new changes would impact the time and space

complexity of the project during execution • After Reverse Engineering and Re-Engineering the code, perform acceptance testing between the teams

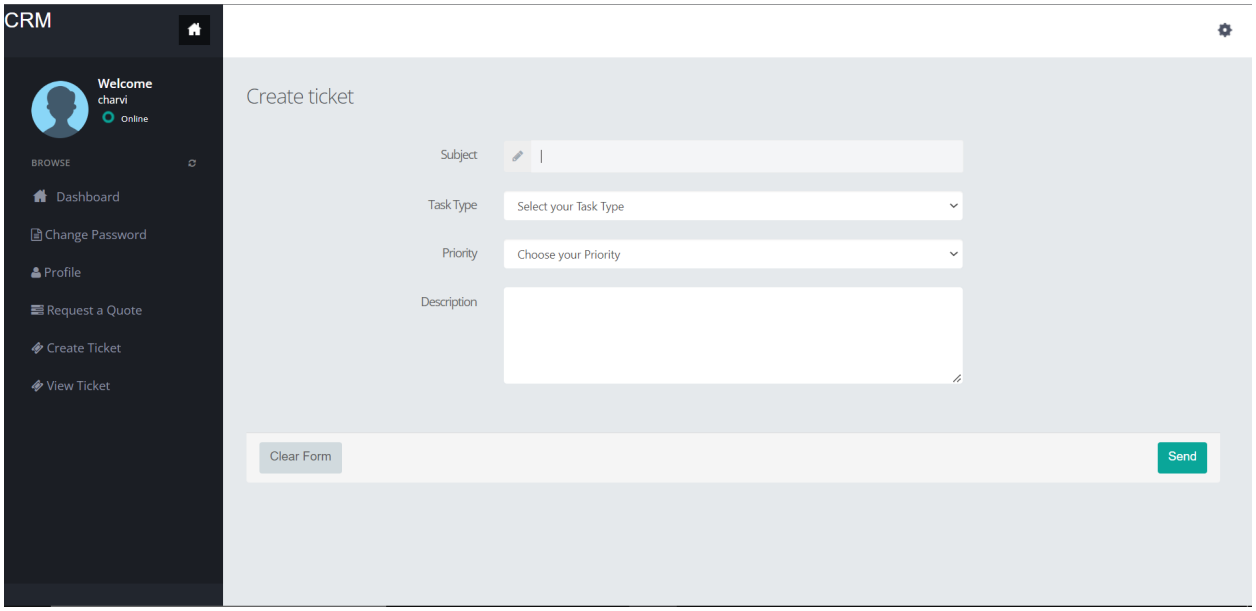
After reverse engineering and re-engineering our code with the other team, we decided to follow their advice and include the features listed above.

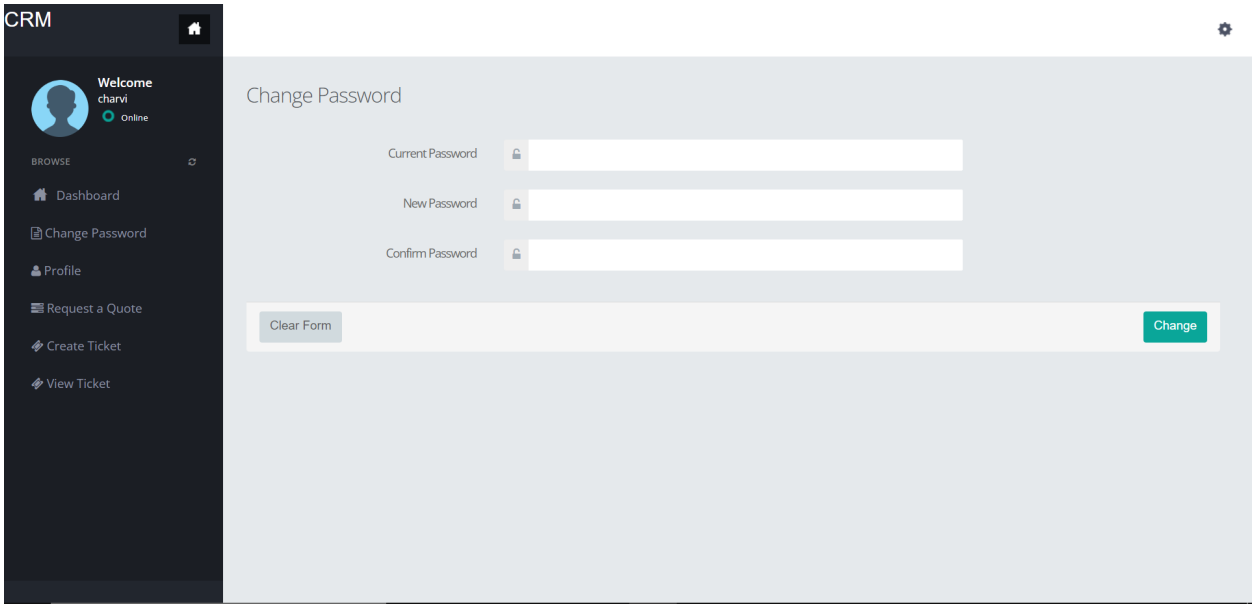
We began mutual acceptance testing after finalising the final code.

Acceptance / Qualification testing: All the requirements as mentioned in the system requirements specification document were met

Installation testing: The application is tested to work on the chrome browser.

Performance testing: The time taken to display the web pages is well within the time specified in the srs document.




CRM
Welcome
charvi
Online
BROWSE
Dashboard
Change Password
Profile
Request a Quote
Create Ticket
View Ticket
Change Password
Current Password
New Password
Confirm Password
Clear Form
Change