APPENDIX A

QUICK DEVELOPMENT OF WEB APPLICATIONS FOR SMEs USING XP PRACTICES

Quick Development of Web Applications for SMEs Integrating Requirement Analysis and Designing Phase using XP Practices

Extreme Programming (XP) is one of the most popular agile approaches to develop the web applications. As defined in section 5.1.1, XP covers up 12 practices (listed in Table 2) to be followed during development process and project management.

1. *On Site Customer* – Customer is continuously involved during development phase.
2. *Small Release* – Rapidly changing requirements are handled by introducing quick releases of the web application.
3. *Metaphor* – Metaphor is a mean of communication among team members and customer. Also, it is a way of describing the architecture of the web application in easiest possible way to any technical/nontechnical person.
4. *Testing* – Unit tests are written before starting actual coding thus, making developers known to the code properly.
5. *Simple Design* – Simple design helps in handling changing requirements. This has minimal number of classes and methods possible. Designing is done considering present trend of the market not the future possibilities because cost of change is not exponential in case of web application.
6. *Refactoring* – Source code must be easier and simpler to understand when documentation is kept minimal. That's why, developers refactor the code base. The structure of code is changed to improve its understandability without changing its functionality.
7. *Pair Programming* – Two persons work simultaneously on the same machine switching their role after completing the task. Role of one person is to handle mouse and keyboard and analysing any other possible way to simplify the approach. Role of other person is to think strategically and decide that the way he is doing his work is best approach to implement the functionality.
8. *The Planning Game* – The planning game is all about prioritizing the requirements and deciding what requirements are to be included in the current release and what are to be postponed for next release. Postponing requirements do not affect the market value of the application as releases are made very shortly.
9. *Sustainable Development* – Over-time work and burnt out situation of developers is avoided in order to maintain software quality high.

10. *Collective Ownership* – The team members collectively own the code base. Whatever change they wish to make in code, they can do it without any fear of unknown repercussion.
11. *Coding Standards* – Coding standards enhances the understandability of the code and improves consistency among team members.
12. *Continuous Integration* – Continuous integration ensures an executable system version available containing all the new features and serving as the baseline for all the work.

Figure 31 shows over all process of development of web application for SMEs (small and medium enterprises) integrating requirement analysis and designing phase using XP practices.

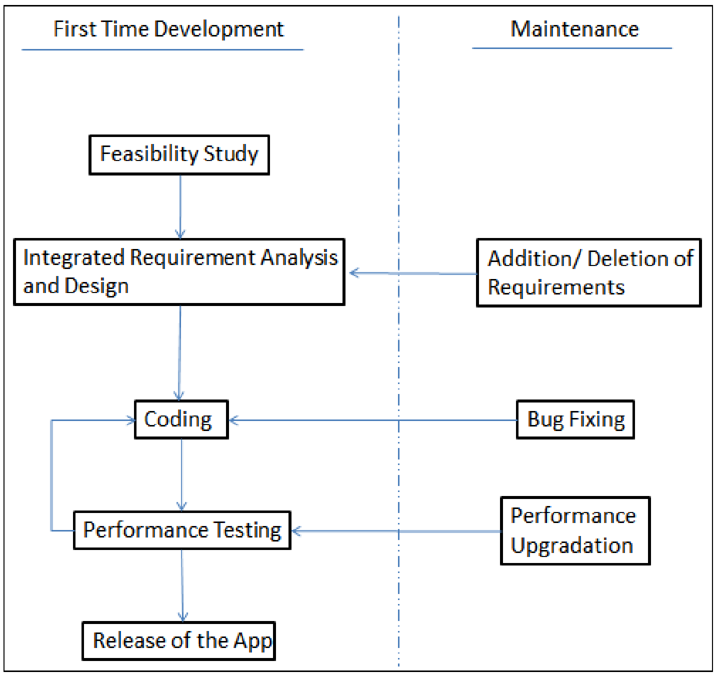


Figure 1 Development of Web Application for SMEs Integrating Requirement Analysis and Designing Phase using XP Practices

Figure 1 represents outline of the entire process of web application development which has been designed considering XP practices. First phase is *Feasibility Study*, which is actually study of already existing web applications addressing the same problem being resolved. This helps in ensuring quality introduction in application design and thus reducing a lot of time and effort in introducing quality aspects on later time. Next phase is *Integrated Requirement Analysis and Design* portioning the developers into teams handling requirements and web page designing together. The way of doing this is illustrated in real application development process described later in this section. After completion of web page design and approval by customer, real coding takes place by the same team who prepared the design of the web pages. Various types of testing like *integration testing,* and *black box testing* are done at time of development and integration of web pages. Next phase is *Performance Testing* after which first release is made.

In case of web applications, handling of rapidly changing requirements is must be attended as seriously as first time development. Figure 1 shows three possibilities after first release of the application: *Addition/ Deletion of requirements, Bug fixing, and Performance issues.* If some new requirement is to be added or to be deleted then it must first be recognized for its requirement group and it must be passed to that particular team handling the requirement group (see real development process depiction described in the section below). If there is some bug it need not be passed to requirement analysis team but it should directly go for implementation. If web application faces some performance issues then the application must be placed for performance testing for respective issue and must be recoded accordingly.

DEVELOPMENT OF WEBSITE *RED DROP* USING XP PRACTICES

Red Drop is a social website motivating people for donating blood to needy ones. This website provides information about blood donors available in required location. Also, people, who are willing to donate their blood, can register with the website and update their contact information on the server so that needy persons could contact them and get blood.

PHASE 1: FEASIBILITY STUDY

Feasibility study is evolution of the proposed project before starting its actual development in order to ensure successful development of the project and maximal benefit gain after launch of the project. Following steps were followed during feasibility study of *Red Drop*.

1. *Project Description*: *Red Drop* is a social beneficiary website which aimed to help people in availing them blood of required blood group in minimal time with minimal effort. Also, people willing to denote their blood or the people who already denoted their blood can register on the website and provide their details on their profile. Stack holders will be person willing to donate blood, people willing to acquire for blood, and admin of the website.
2. *Goals:* Goals of *Red Drop* can be classified into two parts: short term goals and long term goals. Short term goals are to help needy persons quickly by providing them details of blood donors. Long term goals are getting collaboration with blood banks and hospitals thus helping people not only by providing information but also accomplishing their blood requirements.

   Short term goals can be accomplished by providing needy and donors some useful facts about blood donation thus inspiring them for donating blood. Also, donors could be able to register on the website and provide their personal details. Long term goals can be achieved by gathering a huge database of donors and catching popularity and establishing trust among needy persons.

3. *Purpose:* Purpose of the project is establishing a platform for information transfer between blood needy and blood donors and thus helping them in their crisis time. Area of benefiters will get expanded after having a large number of people connecting to the website.
4. *Market Analysis:* There are already some websites running on the same purpose for example, http://www.tngovbloodbank.in/, http://www.bharatbloodbank.com/, http://www.bloodbankindia.net/ etc. These sites are running good and having a huge database. Main purpose of Feasibility Study is getting idea of already existing successfully running websites and extracting good features out of them for using in *Red Drop*.
5. *Team Management:* Team management graph is one of the main outputs after completion of feasibility study. For *Red Drop*, team management graph is as follows:

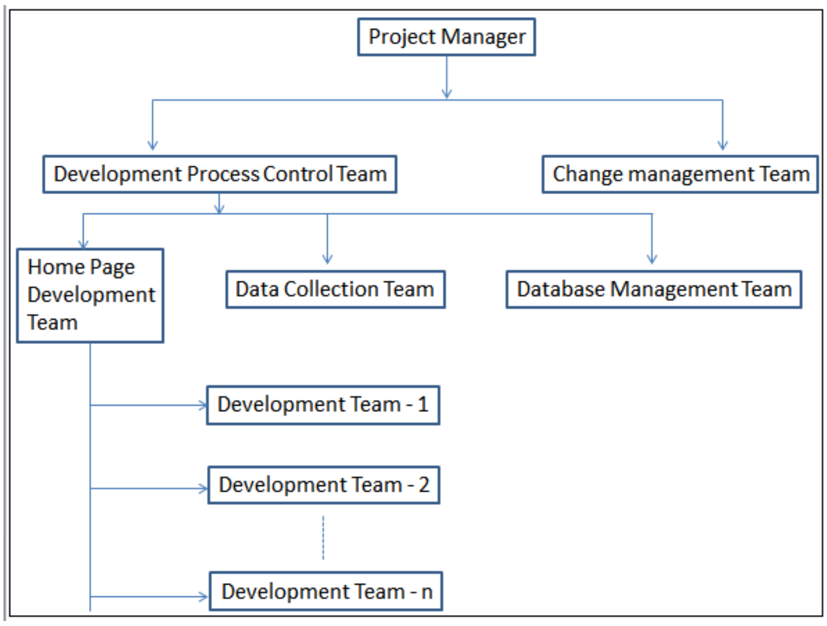


Figure 2 Team Management Chart

6. *Development Process Timeline:* Figure 3 shows timeline of First Time Development Process. There are two criteria for measuring time: Minimum Period and Maximum Period. This time chart was prepared for *Red Drop* only. It may vary depending on requirements of the project.

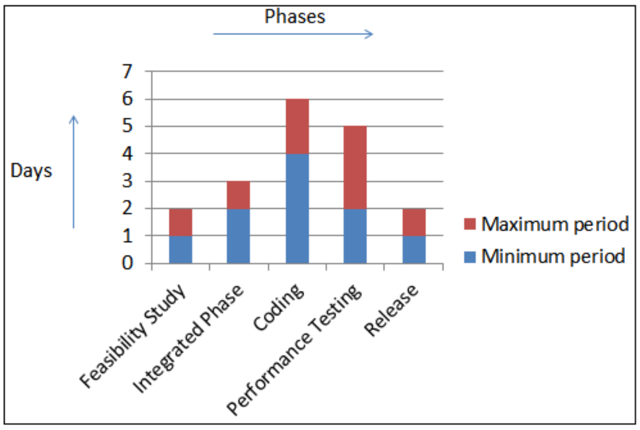


Figure 3 Development Process Timeline

PHASE 2: INTEGRATED REQUIREMENT ANALYSIS AND DESIGN

For requirement elicitation, *Use Case* diagrams have been used. First of all, home page designing takes place for which use case diagram is drawn in order to capture main requirements set. Figure 4 shows use case diagram for *HomePage*.

Use Case Diagram 1*: HomePage*

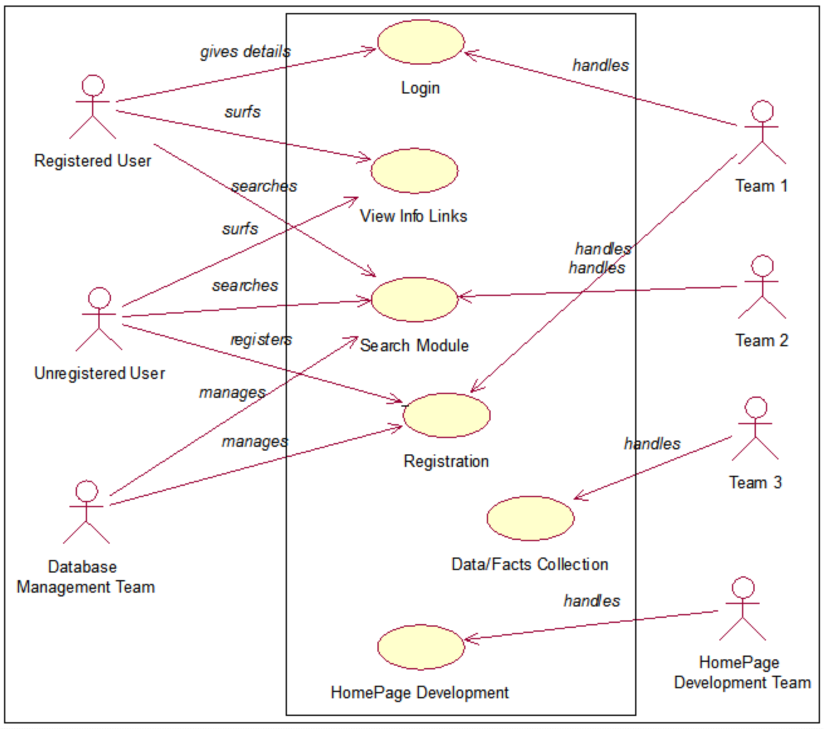


Figure 4 HomePage Use Case Diagram

After completion of requirement gathering for home page, its design takes place. Appendix C covers screenshot of homepage.

Use Case Diagram 2*: Registration*

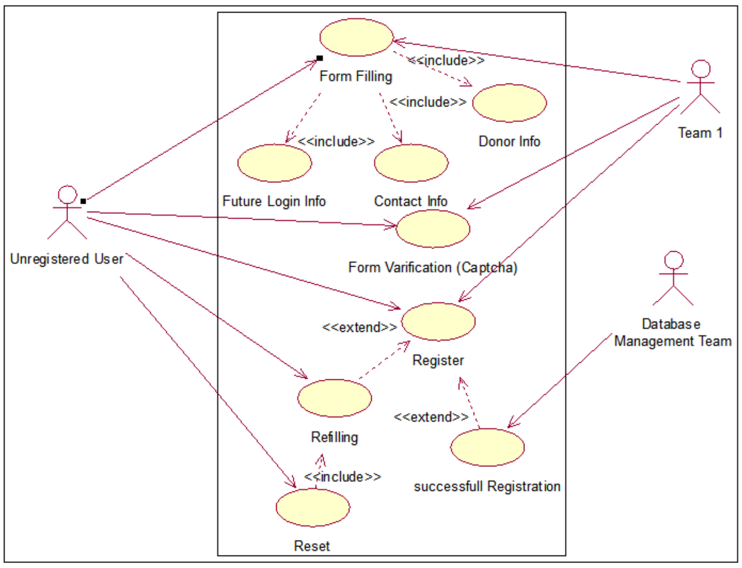


Figure 5 Registration Use Case Diagram

Once use case diagram is finished for *Registration*, same team starts designing for this module. Screen shots are available in Appendix C.

Use Case Diagram 3*: Login*

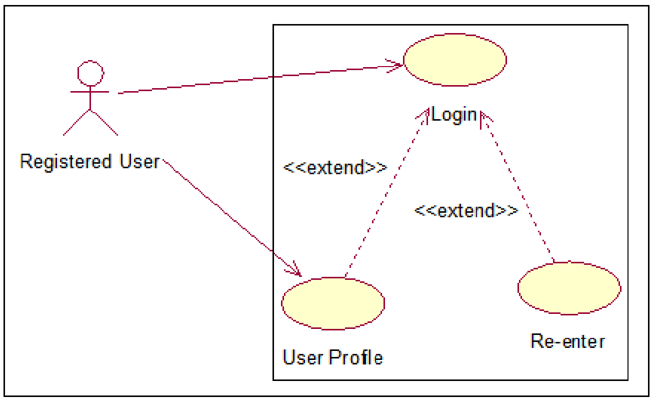


Figure 6 Login Use Case Diagram

Use Case Diagram 4*: Search Module*

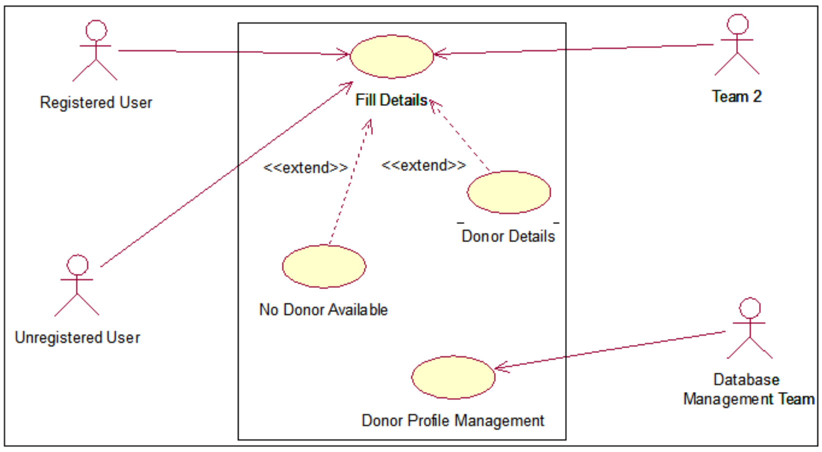


Figure 7 Search Module Use Case Diagram

Use Case Diagram 5: *HomePage Development*

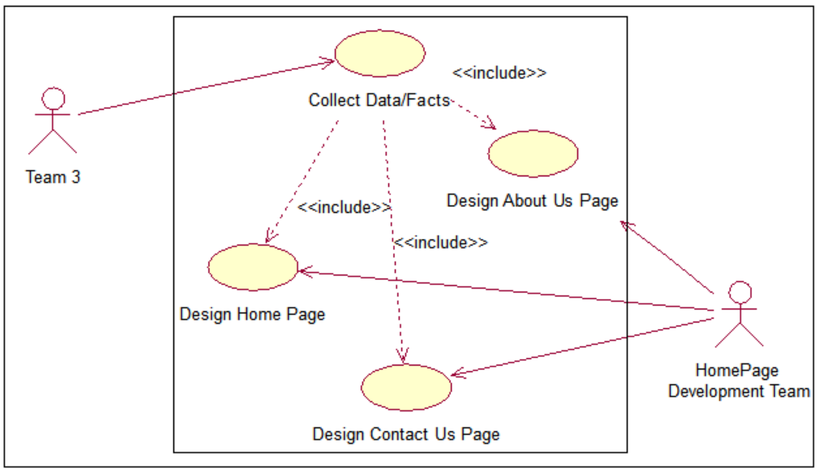


Figure 8 HomePage Development Use Case Diagram

Client side designing takes place along with completion of use case diagrams. 3 major teams are there for this purpose: HomePage Development Team, Data Collection Team, and Database Management Team. HomePage development team consists of 3 members further owning 3 teams: Development Team 1, Development Team 2, and Development Team 3. These all three teams work in parallel and thus speeding up the process of development.

E-R Diagram

E-R diagram depicts entities and relationship between them. Entities form tables and attributes form columns in the table. In *Red Drop*, two tables are there: admin and danorlogin. Admin has all rights to access the database in sense of deletion, updation and addition of entries. Danorlogin contains personal information of donors like their name, address etc.

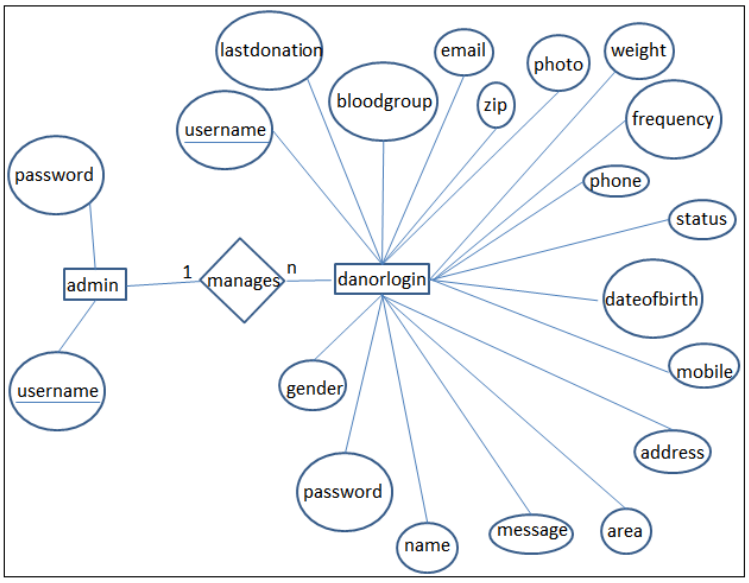


Figure 9 E-R Diagram

PHASE 3: CODING

CODING STANDARDS

A. *Coding Best Practices*

a. Adhere to "Don't Repeat Yourself" (DRY) principles of code design and organization.

b. Whitespace is free and one must not be afraid to use it to separate blocks of code.

c. Use self-documenting variable names. For example, $group_fruits is better than $array.

d. "positive" variable names must be used. `$enable = true` is preferable over `$disable = false`.

e. Functions returning an array should return an empty array instead of FALSE on no results.

f. Functions not throwing an exception on error should return FALSE upon failure.

g. Functions returning only Boolean should be prefaced with is_ or has_.

h. Ternary syntax is acceptable for single-line, non-embedded statements.

i. Comments must be used effectively. Good comments describe the "why", Good code describes the "how".

B. *CSS Coding Standards*

a. Use shorthand where possible. For example, instead of writing "*background-color: #333333; background-image: url(...); background-repeat: repeat-x; background-position: left 10px; padding: 2px 9px 2px 9px;*", it should be written as "*background: #333 url(...) repeat-x left 10px; padding: 2px 9px;*".

b. One property per line.
c. Property declarations should be spaced like so: “property: value;”.
d. Group vendor-prefixes for the same property together –
   i. Longest vendor-prefixed version first.
   ii. Non-vendor-prefixed version must be always included.
   iii. Extra newline must be put between vendor-prefixed groups and other properties.
e. Group declarations of sub-properties.

C. Deprecation of APIs

Occasionally, functions and classes must be deprecated in favour of newer replacements. Since 3rd party plugin authors rely on a consistent API, backward compatibility must be maintained, but will not be maintained indefinitely as plugin authors are expected to properly update their plugins. In order to maintain backward compatibility, deprecated APIs will follow these guidelines:

a. Incompatible API changes cannot occur between bug-fix versions (Ex: 1.6.0 - 1.6.1)
b. API changes between minor versions (Ex. 1.6 - 1.7) must maintain backward compatibility for at least 2 minor versions (Ex. 1.7 and 1.8).
c. Bug-fixes that change the API cannot be included in bug-fix versions.
d. API changes between major versions (Ex. 1.0 - 2.0) can occur without regard to backward compatibility.

D. *PHP Coding Standards*

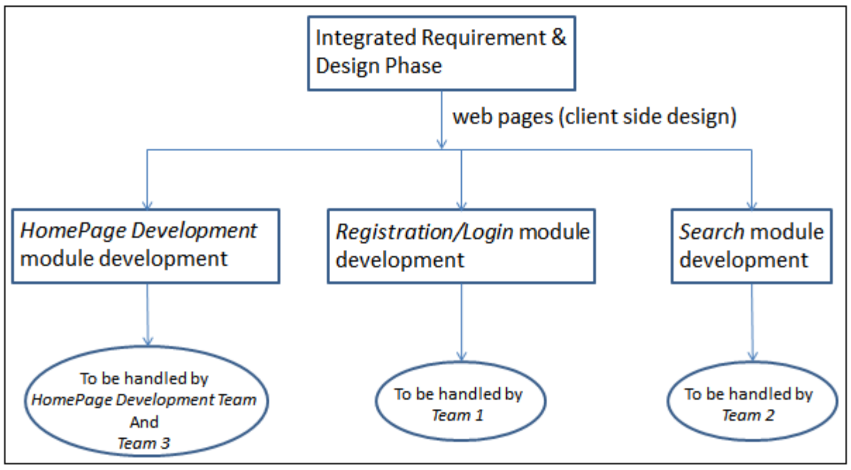
a. Unix line endings.
b. Hard tabs, 4 character tab spacing.
c. No PHP shortcut tags (<? or <?= or <%).
d. PHP Doc comments on functions and classes (all methods; declared properties when appropriate).
e. Mandatory wrapped {}s around any code blocks.
f. Standalone functions must be named using underscore_character().
g. Classes must be named using CamelCase() and methods using lowerCamelCase().
h. Global variables and constants must be named in ALL_CAPS (ACCESS_FRIENDS, $CONFIG).
i. Underscores / camel case must be used to separate Standard English words in functions, classes, and methods.
j. Correctly use spaces, quotes, and {}s in strings.

    k. Line lengths should be reasonable.
    l. // or /* */ must be used when commenting.

E. *Javascript Coding Standards*

    a. Same coding standards as PHP.
    b. Function expressions should end with a semi-colon.

XP practices favour test cases generation before going for actual coding part. Whenever code gets completed of some module, it is tested against test cases at the same time and errors are fixed. *Red Drop* team distribution is shown in Figure 10.

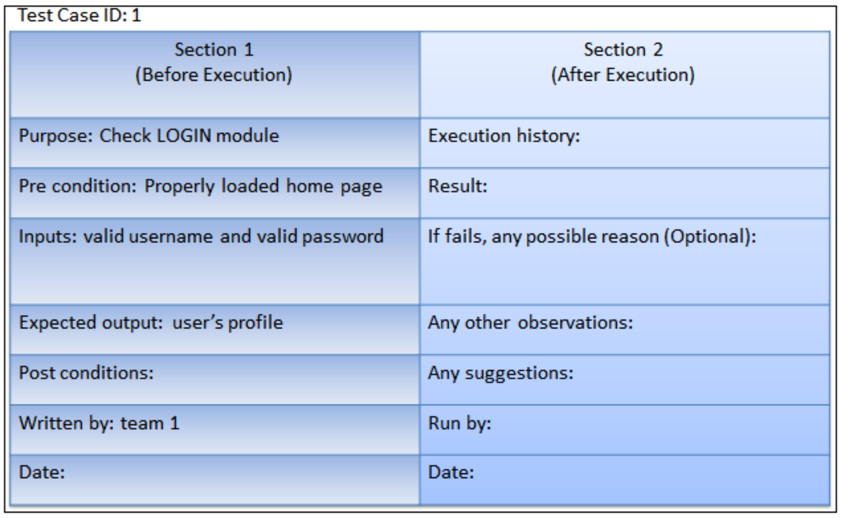


Integrated Requirement & Design Phase
web pages (client side design)
HomePage Development module development
Registration/Login module development
Search module development
To be handled by *HomePage Development Team* And *Team 3*
To be handled by *Team 1*
To be handled by *Team 2*

Figure 10 Code Development Team Distribution

EXAMPLE: *Development of LOGIN module*

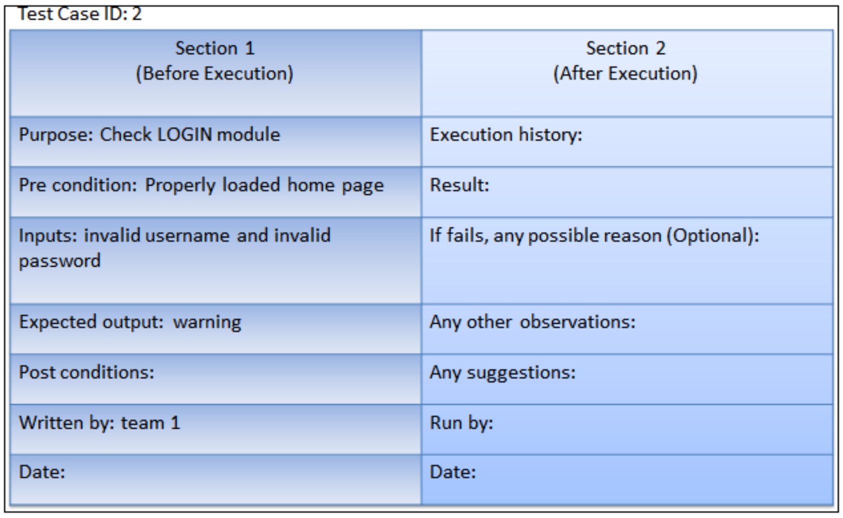
Before actually developing *LOGIN* module, test case will be written for same and after code completion the test cases will be run and code will be checked.



Test Case ID: 1

Section 1 (Before Execution)	Section 2 (After Execution)
Purpose: Check LOGIN module	Execution history:
Pre condition: Properly loaded home page	Result:
Inputs: valid username and valid password	If fails, any possible reason (Optional):
Expected output: user's profile	Any other observations:
Post conditions:	Any suggestions:
Written by: team 1	Run by:
Date:	Date:

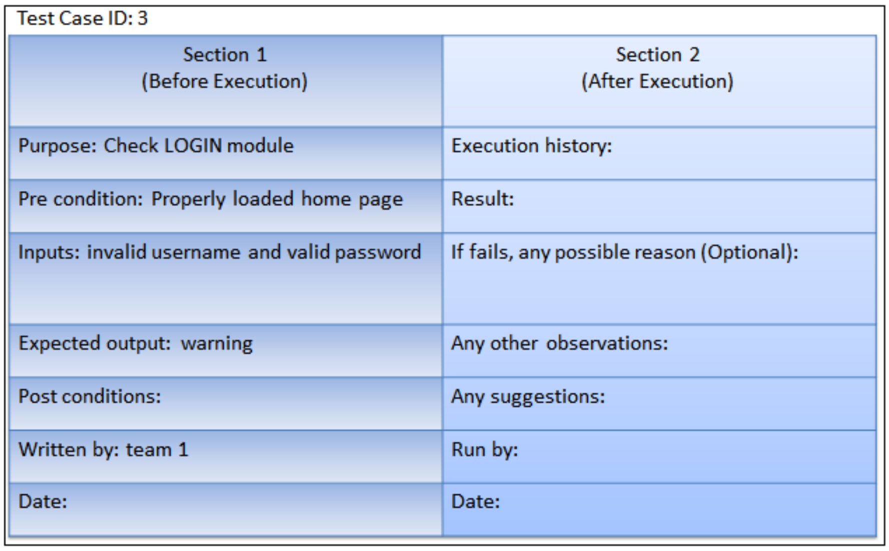
Figure 11 Test Case – 1 for LOGIN Module



Test Case ID: 2

Section 1 (Before Execution)	Section 2 (After Execution)
Purpose: Check LOGIN module	Execution history:
Pre condition: Properly loaded home page	Result:
Inputs: invalid username and invalid password	If fails, any possible reason (Optional):
Expected output: warning	Any other observations:
Post conditions:	Any suggestions:
Written by: team 1	Run by:
Date:	Date:

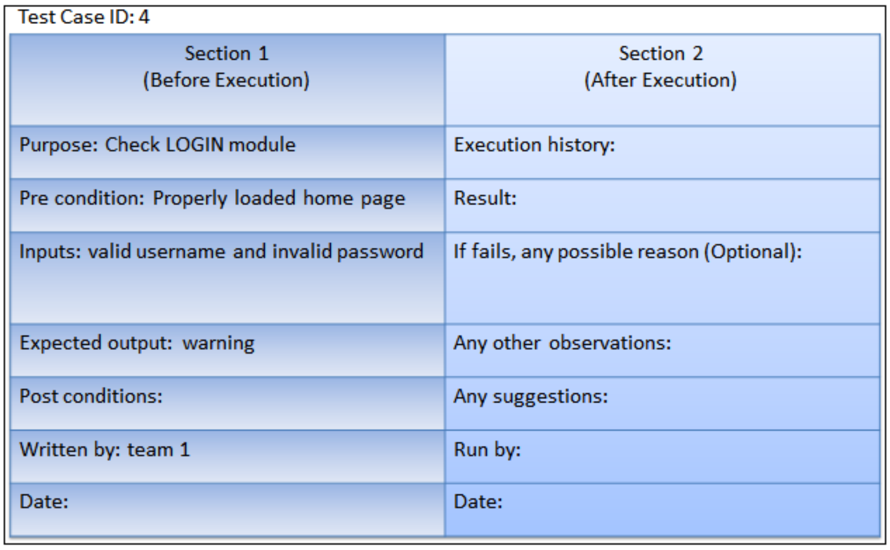
Figure 12 Test Case – 2 for LOGIN Module



Test Case ID: 3

Section 1 (Before Execution)	Section 2 (After Execution)
Purpose: Check LOGIN module	Execution history:
Pre condition: Properly loaded home page	Result:
Inputs: invalid username and valid password	If fails, any possible reason (Optional):
Expected output: warning	Any other observations:
Post conditions:	Any suggestions:
Written by: team 1	Run by:
Date:	Date:

Figure 13 Test Case – 3 for LOGIN Module

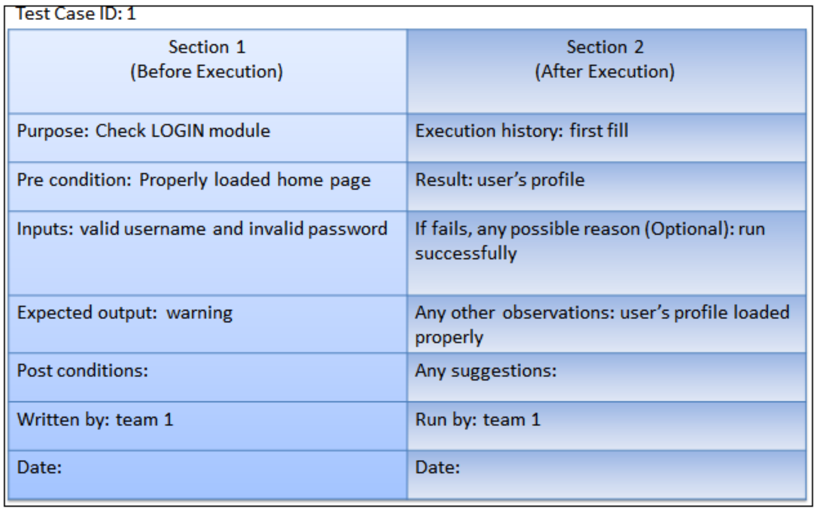


Test Case ID: 4

Section 1 (Before Execution)	Section 2 (After Execution)
Purpose: Check LOGIN module	Execution history:
Pre condition: Properly loaded home page	Result:
Inputs: valid username and invalid password	If fails, any possible reason (Optional):
Expected output: warning	Any other observations:
Post conditions:	Any suggestions:
Written by: team 1	Run by:
Date:	Date:

Figure 14 Test Case – 4 for LOGIN Module

After completion of test cases writing, *LOGIN* module is coded. Then Section 2 of test cases is filled with result of the test case.



Test Case ID: 1

Section 1 (Before Execution)	Section 2 (After Execution)
Purpose: Check LOGIN module	Execution history: first fill
Pre condition: Properly loaded home page	Result: user's profile
Inputs: valid username and invalid password	If fails, any possible reason (Optional): run successfully
Expected output: warning	Any other observations: user's profile loaded properly
Post conditions:	Any suggestions:
Written by: team 1	Run by: team 1
Date:	Date:

Figure 15 Test Case – 1 Result

In same order other test cases are run and results are noted and if any error is found then it is resolved at the same time. After completion of coding of each module, whole system is tested by putting all modules together.

PHASE 4: PERFORMANCE TESTING

Performance testing is applied in order to measure *Red Drop*'s performance under various conditions in terms of responsiveness and stability. Performance testing can be done in four different manners which are briefly described below.

1. *Load Testing*: In load testing, some specific numbers of users are made accessing the website simultaneously. This specific number is expected number of users supposed to surf the website at a single time.
2. *Stress Testing*: Motive of performing stress testing is getting idea of upper limit of the users accessing the website simultaneously without affecting the website's performance. This helps administrator in determining the load which can website undergo above maximum expected numbers of users.
3. *Endurance Testing*: Endurance testing is performed to check system's performance under continuous expected load not just for a small time. Main motive of this type of testing is to monitor memory utilization.
4. *Spike Testing*: As name suggests, spike testing is done by increasing the load on website suddenly thus creating spikes of load at random time interval.

PHASE 5: RELEASE OF WEBSITE

When website gets ready to release, it is deployed on the server. After deployment, it is ensured that website is running successfully on server and is accessible on internet. If some error is found in releasing, server administrator is consulted regarding the issue.