CHAPTER 4

STORY POINTS BASED ESTIMATION

4.1 INTRODUCTION

Project estimation plays a vital role in the success of software process. The outcomes of software estimation process such as, effort, and schedule are the main inputs for project planning and controlling activities. Unreliable and unrealistic estimates are the critical causes of software project failures [9]. Estimation is equally important during maintenance process of a software product as it is performed for the development. A good maintenance plan reduces risks and inefficiencies related to the maintenance tasks. Software maintenance planning involves activities such as, estimation of size, effort, duration, staff and costs to control the maintenance process. It is observed that the productivity of programmers during maintenance is different than new development and it is affected from people, process, and product factors. It is also observed that the productivity in the maintenance process is generally lower than the development process [17]. Productivity is influenced by the quality of architecture, detail design, documentation, configuration control, comment statements, maintenance personnel and their training, and complexity of the product [17].

As software maintenance is an important activity and accounts for the majority of the software total cost but the researchers have less attention on maintenance estimation as compared to the estimation of new software development [7]. There exist fewer effort estimation models for software maintenance as compared to software development. Software maintenance effort can be estimated by existing estimation models such as, Annual Change Traffic (ACT) model, Function Point (FP) model, and COCOMO 2.0 reuse model. The different estimation model considers Source Lines of Code (SLOC),
Function Points (FPs) and object points as sizing units for determining maintenance effort.

ACT model is proposed by Boehm for maintenance cost estimation [70]. This model assumes that the maintenance costs can be determined using the same cost drivers that determine the development costs [70, 78, 79]. Boehm defines the annual change traffic as fraction of the source instructions that change during a year through addition or modification [70]. Annual change traffic is calculated using following formula:

\[ \text{ACT} = \frac{\text{KLOC}_{\text{added}} + \text{KLOC}_{\text{deleted}}}{\text{KLOC}_{\text{total}}} \]

Where, \( \text{KLOC}_{\text{added}} \) is the total Kilo Lines of Source Code added during maintenance. \( \text{KLOC}_{\text{deleted}} \) is the total KLOC deleted during maintenance. And the \( \text{KLOC}_{\text{total}} \) is the total KLOC in the project. Boehm has established following equation to estimate maintenance costs using the data gathered from 63 projects [70].

\[ \text{Maintenance Cost} = \text{ACT} \times \text{Development Cost} \]

FP is another estimation model, which is based on five function types to estimate the size of the software. Different function types of FP model includes two data functions types; namely, internal logical files (ILF) and external interface files (EIF), and the remaining three transactional function types; namely, external inputs (EI), external outputs (EO), and external inquiries (EQ) [80, 81]. As this model is originally developed for the effort estimation of new software projects, therefore, its expanded version is developed by Albrecht for the maintenance. The maintenance version of FP model includes three components of functionality, i.e., application functionality, conversion functionality, and value adjustment factor for the function point calculation. The application functionality consists of FPs, which are added, changed, and deleted in a maintenance project. The conversion
functionality consists of FPs delivered during conversion. The fourteen value adjustment factors are considered with varying range of intensity level based upon the software characteristics.

The maintenance project FP is calculated using following formula:

$$EFP = (ADD + CHG + CFP) \times VAF_a + (DEL \times VAF_b)$$

Where, EFP is the enhancement project function point. ADD, CHG, and DEL are added, changed and deleted FPs respectively, which are counted as application functionality. CFP represents the unadjusted function points added during conversion. VAFa and VAFb are the value adjustment factors of the application after and before the maintenance.

COCOMO 2.0 reuse model is COCOMO version for software maintenance that uses KSLOC as sizing unit. It includes 17 effort VAF such as, required software reliability, database size, product complexity, required reusability, documentation, etc., in a multiplicative manner. It uses five scale values for each VAF factor such as, precededness, development flexibility, architecture or risk resolution, team cohesion, and process maturity.

Organizations having chronological data for annual maintenance cost estimation uses ACT model, which is derived from the original COCOMO model. There are some basic limitations in ACT model such as the results of estimation are likely to be unreliable if the systems are completely new and has no historic basis for estimating the ACT. ACT has no way to determine scientific and quantitative aspects for representing software maintainability and thus, it introduces significant risks into the model. The value adjustment factors of FP model are originally developed for new development, therefore, they are not applicable for the maintenance projects.
The existing models of software maintenance estimation are based on the traditional software development methodologies. These models produce realistic result for similar methodology because such models consider function points, source lines of code or object points as the basis of size measurement, which are not suitable for extreme programming based maintenance methodology.

Agile methodology has its own estimation techniques. The most commonly used techniques for estimating software development in agile are expert opinion, estimation by analogy, disaggregation, and planning poker [33, 34]. In expert opinion technique, expert relies on his/her intuition feel and provides an estimate and assigns story points or ideal-days to user stories. For this method, estimator should have the knowledge of different domains. The expert opinion-based approach of estimation reduces the time of estimation. An alternative approach is estimation by analogy. In this technique, estimator compares the story being estimated with one or more other stories. If the size of the story is twice, it is assigned as an estimate twice as large. For easy estimation and story comparison, the disaggregation technique breaks the large story or feature into multiple smaller stories. The planning poker technique is an effective and pleasant approach, which combines expert opinion, estimation by analogy, and disaggregation way of estimation. In planning poker, a group is formed by involving experts from all disciplines of software development for the estimation such as, analysts, programmers, testers, database engineers, and GUI designers. This technique provides reliable estimations, and takes less time duration. Participants in planning poker include programmers, testers, database engineers, analysts, user interaction designers etc. It is similar to the Wideband Delphi (WBD) approach of estimation that estimates the effort through interaction and communication among the coordinator and experts [34].

The non-algorithmic techniques of agile for the estimation such as, expert opinion, analogy, disaggregation, and planning poker are
unpredictable in the absence of the past data and experts [33, 34]. These techniques are used to derive the estimates on the basis of opinion of experts and historical data. Further, these methods may generate different estimates for same project depending on the intuition of the estimators [33, 82, 83]. Therefore, there is a need of algorithmic approach for the estimation in the iterative maintenance life cycle using XP. We have proposed an algorithmic estimation model, Software Maintenance Effort Estimation Model (SMEEM), which uses story points as a size measure.

The SMEEM model incorporates value adjustment factors for estimation of size and effort of a maintenance project. These value adjustment factors are project specific, and hence, cover the related factors, mainly documentation quality, structuredness, modularity, reusability of existing software modules, conformity with software engineering standards, familiarity with programming language and knowledge of application domain, and changeability and readability of source code [6, 84, 85, 86]. The task-level SMEEM model calculates size of maintenance in terms of story points. The release-level model of SMEEM calculates effort in term of Person-Days (PD). The proposed model of estimation also provides phase wise effort estimation. The process of estimation using SMEEM is illustrated using a case study of maintenance project.

The proposed SMEEM model calculates overall effort as well as phase wise effort distribution. The organization of the chapter is as follows. Section 4.2 discusses the VAF that can affect effort of maintenance. The working procedure of SMEEM model is described in Section 4.3. An illustration of SMEEM is presented in Section 4.4. The phase wise effort estimation using SMEEM is presented in Section 4.5. Section 4.6 covers empirical study of maintenance effort. A case study is presented in Section 4.7 that considers three RC stories of a university web portal, a maintenance project, for estimation. Finally, Section 4.8 describes the summary of presented research.
4.2 FACTORS AFFECTING SOFTWARE MAINTENANCE ESTIMATION

Productivity of programmers during maintenance is different than the new development. It is affected by current status of the existing system defined by the quality and availability of artifacts. Productivity during maintenance also depends upon the skills of the maintainers. There are several factors that affect the effort estimation in maintenance projects such as, documentation quality, structuredness, modularity, reusability of existing software modules, conformity with software engineering standards, familiarity with programming language and knowledge of application domain, and changeability and readability of source code. These factors are the value adjustment factors that may affect the value of story points of an RC story, which are considered as sizing unit in the proposed SMEEM model. The Value Adjustment Factor (VAF) and their influencing activities are shown in Table 4.1 and described in following subsections.

4.2.1 Documentation Quality

Documentation of a software system contains different deliverables, which are generated as the outcomes in different phases of development process such as, requirement document, feasibility report, design document, test document etc. These documents are not only used as a basis in development process but treated as milestones and is considered as a contract between developer and customer. During maintenance, these documents provide information regarding the system. The initial activity of software maintenance, i.e., understanding, is highly affected by quality and availability of documentation. Documentation offers comprehensive, clear and short information concerning the system [6, 87]. It provides understanding about the different functionality and their relationships.

Documentation is used in different phases of maintenance. Documentation is an important tool for the communication among team
members and it can be useful even if it is not up-to-date [88, 93]. The main quality characteristics of software documentation are clarity, consistency, completeness, and conciseness [87, 89]. Documentation clarity can be measured by calculating the inverse of document ambiguity. Consistency is evaluated on the basis of contradiction in software documentation. Completeness evaluates whether the documentation covers all of the software components and their relations. The conciseness presents the ratio of the unrelated or inadequate statements to the total statements in the documentation. Documentation affects to system understanding, which is directly related with the productivity and the cost of maintaining a software product.

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Value Adjustment Factor</th>
<th>Influenced Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Documentation quality</td>
<td>Understanding</td>
</tr>
<tr>
<td>2.</td>
<td>Structuredness</td>
<td>Understanding, analysis, modification, and testing</td>
</tr>
<tr>
<td>3.</td>
<td>Modularity</td>
<td>Understanding, analysis, modification, and testing</td>
</tr>
<tr>
<td>4.</td>
<td>Reusability of existing software modules</td>
<td>Productivity and reliability</td>
</tr>
<tr>
<td>5.</td>
<td>Conformity with software engineering standards</td>
<td>Understanding, analysis, and modification</td>
</tr>
<tr>
<td>6.</td>
<td>Familiarity with programming language and knowledge of application domain</td>
<td>Understanding, analysis, and modification</td>
</tr>
<tr>
<td>7.</td>
<td>Changeability and readability of source code</td>
<td>Analysis and modification</td>
</tr>
</tbody>
</table>
4.2.2 Structuredness

Structuredness refers to the relationship between different modules of a software system. It is an important quality of a system that affects the different activity of software maintenance such as, understanding, analysis, modification, and testing [84, 87]. If a software system has good structure then maintainers can easily understand it for error inspection and can include new features. The characteristics of good structure in software enables for analysis and modification. Regression testing is major parts of testing in software maintenance, therefore, a good structure allows the system testable. Structuredness is a feature that presents a specific pattern of the organization of its interdependent components [88]. Structuredness measures coupling between different modules of the system. Structuredness property can be improved by minimizing coupling between modules of a system.

4.2.3 Modularity

Modularity is a concept, which allows design, development and testing of a software system into multiple components. This method solves a complex problem through decomposition. Modularity of a software system is important for maintenance activities. Modularity supports important aspects of maintenance such as, understandability, analyzability, modifiability, and testability [87]. It allows to understand a system component wise instead of understanding the whole system at a time. It helps to modify components of a system independent from other components. The general metrics of modularity consists of coupling, cohesion, and the separation of concerns [85]. Errors or functions can be identified effortlessly and proficiently, thereby reducing maintenance efforts. Modularity reduces efforts of testing, error inspections and causes of the errors. For example, modularity in object-oriented language code is achieved by dividing the whole system into several manageable classes and their relationships.
4.2.4 Reusability of Existing Software Modules

Reusability of existing software modules refers to the ability of a typical maintainer to reuse software artifacts of the existing system’s modules without major changes. Software reusability improves efficiency, value and maintainability of software products by implementing or updating software systems using existing software systems or legacy system assets [86, 90]. A reusability approach may possibly saves up to 20% of development costs.

4.2.5 Conformity with Software Engineering Standards

It shows the status of the existing system from the standards followed in the development and in the previous maintenance. In maintenance process, different activities such as, program comprehension, readability and modification are supported by conformity with software engineering standards. It is considered as the productivity factors that reduces maintenance effort [91].

4.2.6 Familiarity with Programming Language and Knowledge of Application Domain

Experience of maintainers about programming language in which the existing system developed is important aspect from maintenance point of view. As the program comprehension is an important activity of maintenance, familiarity of programming language plays an important role in it. Experience of the programming language improves the productivity of maintainers by increasing the capability of understanding, analysis and modification [91].

As the experience of programming language is important, in the same way, domain in which the existing system belongs is also important from maintenance perspectives. Application domain knowledge of existing software system refers to the ability of a typical maintainer to understand the requirements and terminology of the existing system without significant support. The domain knowledge
affects entire activity of maintenance; therefore, the system requirements analysis requires good domain knowledge [90].

4.2.7 Changeability and Readability of Source Code

It shows the changeability and readability of source code of the existing system from modification perspective. The changeability and readability of source code is software quality factor, which can be increased by modularity and structure of source code. It is also affected by the programming language of existing system [91].

4.3 SMEEM: AN ALGORITHMIC APPROACH TO ESTIMATION

The proposed Software Maintenance Effort Estimation Model (SMEEM) predicts maintenance effort in terms of story point. It is a task-level model that estimates the effort of implementing each maintenance task, which is considered in the form of an RC story. This model deals with small effort estimates. SMEEM incorporates the VAFs as shown in Table 4.1. The SMEEM divides software maintenance estimation process into following steps.

- Factor count computation
- Story point assignment
- Adjusting story points
- Calculate size of maintenance
- Calculate duration of maintenance

The SMEEM uses RC stories and old software as input and performs all the aforesaid steps to determine maintenance effort. The individual steps are described in the following subsections.

4.3.1 Factor Count Computation

The intensity of VAFs of maintenance project are identified and categorized into the range of low, medium and high values. These VAFs are scaled from 1 to 5 (1- very poor, 2- poor, 3- average, 4-good
and 5-excellent). The scale of a value adjustment factor is assigned on the basis of the status of existing system. For example, in university web portal, quality of documentation is good; therefore, the value of VAF for documentation quality is considered as 4. The value of all categories of various factors are summed up and denoted as Factor Count (FC). Thus, the FC is computed as follows:

\[
\text{Factor Count (FC)} = \sum_{i=1}^{n} VAF_i
\]

Where, \( n \) is the number of VAFs in a project

4.3.2 Story Point Assignment

A story point is assigned to each RC story using analogy and planning poker estimation technique by maintenance team. Story point is the unit of measure for size of an RC story. In the planning poker technique, the individual stories are presented for the estimation [92]. A group is formed involving experts from all disciplines of software development. It includes analysts, programmers, testers, database engineers, user interaction designers, and other interested for the estimation. Each member of the group holds a deck of cards, which contain values; for example, 0, 1, 2, 3, 5, 8, 13, 20, 40 and 100. These values represent the number of story points. The customer reads a story in front of estimators. Each member privately selects a card representing his or her estimate of the "size" for the story and cards are presented simultaneously. Size of the story represents complexity, risk etc. If the cards selected by all estimators have the same value then it considered as the estimate of the card.

For example, if all members selected card having value 3 for a particular story then value 3 will be considered as final estimate. Otherwise, estimators argue their estimates with reasons while selecting high and low values. For example, a particular member selected card of 2, another member selected card of 8 while other
members selected card might contain values between 2 and 8 for the same story. Then 2 and 8 value estimators represent low and high value, respectively and they would suggest their reasons for the selection. Furthermore, each estimator reselects an estimate card, and all cards are again revealed at the same time. This process is repeated for all stories.

4.3.3 Adjusting Story Point

Story point calculated in the previous step is unadjusted because it is decided from the new development standpoint. Story point can be adjusted from the maintenance perspective with the help of FC computed in the previous step. Adjusted Story Point (ASP) can be computed with the help of following equation:

$$\text{ASP} = \text{SP} - \left[ \text{CF} \times \left( \frac{\text{FC} \times \text{Max}_\text{VAF}}{\text{FC} + \text{Max}_\text{VAF}} \right) \right]$$  \hspace{1cm} (4.2)

Where, CF is the critical factor and its value shows the type of maintenance project. In this model, three types of maintenance projects are considered, which are web based application, MIS, and critical project. The CF values of these projects are 0.60, 0.35 and 0.20, respectively. The values of CFs are derived from the empirical study shown in Section 4.6. FC is the summation of VAF, which is derived from Equation (4.1) and Max_VAF is the summation of highest intensity of VAFs.

4.3.4 Calculate Size of Maintenance

The Size of Maintenance (SoM) is the size of a maintenance project. The SoM can be computed using following equation:

$$\text{Size of Maintenance (SoM)} = \sum_{i=1}^{n} \text{ASP}_i$$  \hspace{1cm} (4.3)

Where, \(\text{ASP}_i\) is the ASP of \(i^{th}\) RC story and \(n\) is the total number of RC stories in a maintenance project.
4.3.5 Calculate Duration of Maintenance

The Duration of Maintenance (DoM) shows the completion time of submitted RC stories. It can be computed as follows:

\[
\text{Duration of Maintenance (DoM)} = \frac{\text{SoM}}{\text{Velocity}} \quad (4.4)
\]

Where, Velocity is the number of ASPs developed in an iteration. Cost of a single RC story or component of a maintenance project can be computed from the DoM and imbursement paid to maintainers. Finally, summation of all RC stories provides overall maintenance cost of a project.

4.4 SMEEM: AN ILLUSTRATION

An RC story of 20 story points is considered here to illustrate the ASP computation through the proposed estimation model. Maintenance projects include web based application, MIS and critical project with CF values .60, .35 and .20, respectively. The working of proposed model of maintenance estimation has investigated for various intensity levels of VAFs such as, documentation quality, structuredness, modularity, reusability of existing software modules, reusability of existing software modules, conformity with software engineering standard, familiarity with programming language and knowledge of application domain, and changeability or readability of source code, as shown in Table 4.2.

The value of ASP is computed using SMEEM at different intensity levels of VAFs for various project categories. Table 4.3 shows the ASP computations. In a critical category of project, the input of CF is 0.20, and ASP is computed as 16.88. It has been observed that inclusion of VAF and CF in such category of projects changed the estimated ASP to approximately 84.4 % of its previous SP and estimated more realistic values. The ASP computation with respect to different values of FC and CF is shown Figure 4.1 through radar chart.
Table 4.2: Computation of Value Adjustment Factors

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Value Adjustment Factors</th>
<th>Intensity Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Documentation quality</td>
<td>4</td>
</tr>
<tr>
<td>2.</td>
<td>Structuredness</td>
<td>5</td>
</tr>
<tr>
<td>3.</td>
<td>Modularity</td>
<td>4</td>
</tr>
<tr>
<td>4.</td>
<td>Reusability of existing software modules</td>
<td>4</td>
</tr>
<tr>
<td>5.</td>
<td>Conformity with software engineering standard</td>
<td>3</td>
</tr>
<tr>
<td>6.</td>
<td>Familiarity with programming language and knowledge of application domain</td>
<td>4</td>
</tr>
<tr>
<td>7.</td>
<td>Changeability/ Readability of source code</td>
<td>4</td>
</tr>
</tbody>
</table>

\[ FC = 28 \]

Table 4.3: Computation of ASP with Story Point 20

<table>
<thead>
<tr>
<th>FC</th>
<th>Story Point 20 CF=0.60</th>
<th>Story Point 20 CF=0.35</th>
<th>Story Point 20 CF=0.20</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>16.50</td>
<td>17.96</td>
<td>18.83</td>
</tr>
<tr>
<td>10</td>
<td>15.33</td>
<td>17.28</td>
<td>18.44</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>16.50</td>
<td>18.00</td>
</tr>
<tr>
<td>18</td>
<td>12.86</td>
<td>15.84</td>
<td>17.62</td>
</tr>
<tr>
<td>21</td>
<td>12.12</td>
<td>15.41</td>
<td>17.37</td>
</tr>
<tr>
<td>25</td>
<td>11.25</td>
<td>14.89</td>
<td>17.08</td>
</tr>
<tr>
<td>28</td>
<td>10.67</td>
<td>14.55</td>
<td>16.88</td>
</tr>
<tr>
<td>31</td>
<td>10.13</td>
<td>14.24</td>
<td>16.71</td>
</tr>
<tr>
<td>35</td>
<td>9.50</td>
<td>13.87</td>
<td>16.50</td>
</tr>
</tbody>
</table>
4.5 PHASE WISE EFFORT ESTIMATION

The SMEEM is extended to estimate phase wise effort in a project. It is a release-level model that estimates the effort of a planned set of RC stories or a planned release. This model uses data from the past releases to estimate the effort for the next release. The steps of proposed model for phase wise effort estimation are shown in Figure 4.2. The proposed estimation approach uses RC stories and existing software system as inputs and performs all the steps in the proposed model. The individual steps are described in the following subsections. It also incorporates the VAFs such as, documentation quality, structuredness etc., which have been discussed in Section 4.2.

4.5.1 Story Point Assignment

Initially, SP is assigned to each RC story using analogy and planning poker estimation technique by the maintenance team.
4.5.2 Calculate Size of Maintenance

The size of maintenance is the sum of all RC stories of a single iteration or category of maintenance. SoM will be computed using following equation:

\[
\text{Size of Maintenance (SoM)} = \sum_{i=1}^{n} SP_i
\]

(4.5)

Where, \(SP_i\) is the SP of \(i^{th}\) RC story and \(n\) is the total number of RC stories in a maintenance project.
4.5.3 Unadjusted Effort Calculation

The size of maintenance is converted into PD, i.e., size is converted into effort. In this model, we assume that one story point is equal to two person-days. Effort calculated in this step is unadjusted because it is calculated from development standpoint. At the later steps, this effort is adjusted from maintenance perspectives with the help of VAFs. SoM is converted into unadjusted effort, i.e., story point into PD through the following equation:

\[ E_{\text{Unadjusted}} = 2 \times \text{SoM} \]  

(4.6)

4.5.4 Factor Count Computation

The intensity of VAFs in maintenance projects are identified and categorized into the range of low, medium and high. These VAF are scaled from 1 to 5 (1-very poor, 2-poor, 3-average, 4-good and 5-excellent). The scale of a value adjustment factor is assigned on the basis of status of existing system. For example, in university web portal project, quality of documentation is good; therefore, the value of VAF for documentation quality is considered to be 4. The value of all categories of various factors are summed up and denoted as FC. Thus, the FC is computed as follows:

\[ \text{Factor Count (FC)} = \sum_{i=1}^{7} \text{VAF}_i \]  

(4.7)

Here, \( i \) represent the VAFs, which are considered 7 in number.

4.5.5 Effort Adjustment using FC

At this step, effort is adjusted on the basis of FC that shows current status of project and the category of project. The actual effort (E) can be computed with the help of following equation:

\[ E = E_{\text{Unadjusted}} - [\text{CF} \times ((\text{FC} \times \text{Max}_{\text{VAF}}) / (\text{FC} + \text{Max}_{\text{VAF}}))] \]  

(4.8)

Here, the value of CF shows the type of maintenance project. In this model, there are three types of maintenance projects, i.e., web based
application, MIS, and critical project and their CF values are .60, .35 and .20, respectively. The value of CF derived from the empirical study is shown in Section 4.6. FC is the summation of VAF, which is derived from Equation (4.7) and Max_VAF is the summation of highest intensity of VAF.

4.5.6 Phase Wise Effort Computation

Phase wise effort computation is a vital part of estimation, which is crucial to determine the accurate estimates for proper planning. Phase wise effort can be computed using following equation:

\[
\text{Phase-name} = \text{EP} \times 0.01 \times E
\]  

(4.9)

Here, E is the actual effort, which is derived from Equation (4.8) and EP is the effort in percentage, which indicates the percentage of overall effort involved in the particular phase of maintenance. EP is derived from the empirical study shown in the next section of this chapter.

4.6 EMPIRICAL STUDY OF MAINTENANCE EFFORT

The empirical study was conducted through questionnaires presented in Appendix A-1, to determine efforts of maintenance. The questionnaires covered various aspects of maintenance process such as, phases and activities as well as effort involved in it. It also includes factors that affect effort in maintenance. The survey is performed to collect actual data from the industry and practitioners involved in maintenance projects regardless of their project identity and organization. The feedbacks of survey were collected from approximately 50 practitioners from industry and 150 PG students, who are only involved in maintenance projects. They have suggested that maintenance can be categorized into two classes, i.e., maintenance provided by the third party, and the maintenance carried out by the original developer.
The phases in each category of maintenance vary with each class having varying amount of efforts involved during maintenance. Several observations are drawn from this survey. In case of third party and corrective maintenance, major part of effort is involved in understanding the software and less effort is spent in coding part. In case of third party and adaptive maintenance, major part of effort is consumed in understanding the software and testing while less effort is consumed in problem analysis and coding part. If third party includes new requirements and performs perfective maintenance then major part of efforts is consumed to understanding the software and coding whereas less effort is spent in testing and in the impact analysis. From the survey, it is observed that large amount of effort is spent in understanding the software during third party maintenance. Table 4.4 presents the effort distribution in case of third party maintenance.

It is observed that if maintenance services provided by the original developer and maintenance type is corrective then the more effort is spent in testing phase while less effort is spent in coding part. In case of new requirement and perfective maintenance, more effort is consumed in coding and testing phase and less effort is spent in understanding of software. If maintenance type is adaptive then the more effort is consumed in testing phase. Table 4.5 presents the effort distribution in case of maintenance is performed by the original developer. More maintenance effort is contributed for testing and debugging phases.

4.7 CASE STUDY

A university provides online facility to maintain academic activity of its affiliated colleges through a web portal. The affiliated colleges have very large number of students and teachers. Web portal provides services regarding finding college details, admission procedures, view faculty profiles, displaying notices, details of course syllabi and schemes, time table of examination, and results.
### Table 4.4: Phase Wise Effort Distribution in Case of Third Party Maintenance

<table>
<thead>
<tr>
<th>Phase Name</th>
<th>Effort (%)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Corrective</td>
<td>Adaptive</td>
<td>New Requirement/Perfective</td>
<td>Average</td>
<td></td>
</tr>
<tr>
<td>Software Understanding</td>
<td>55%</td>
<td>40%</td>
<td>40%</td>
<td>45%</td>
<td></td>
</tr>
<tr>
<td>Problem Analysis/Impact Analysis</td>
<td>20%</td>
<td>10%</td>
<td>20%</td>
<td>17%</td>
<td></td>
</tr>
<tr>
<td>Testing (Debug the problem)</td>
<td>15%</td>
<td>30%</td>
<td>10%</td>
<td>18%</td>
<td></td>
</tr>
<tr>
<td>Coding (Change as per need)</td>
<td>10%</td>
<td>20%</td>
<td>30%</td>
<td>20%</td>
<td></td>
</tr>
</tbody>
</table>

### Table 4.5: Phase Wise Effort Distribution in Case of Maintenance Performed by the Original Developer

<table>
<thead>
<tr>
<th>Phase Name</th>
<th>Effort (%)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Corrective</td>
<td>Adaptive</td>
<td>New Requirement/Perfective</td>
<td>Average</td>
<td></td>
</tr>
<tr>
<td>Software Understanding</td>
<td>20%</td>
<td>20%</td>
<td>10%</td>
<td>17%</td>
<td></td>
</tr>
<tr>
<td>Impact Analysis</td>
<td>20%</td>
<td>2%</td>
<td>20%</td>
<td>14%</td>
<td></td>
</tr>
<tr>
<td>Implementation/ Coding</td>
<td>10%</td>
<td>20%</td>
<td>40%</td>
<td>23%</td>
<td></td>
</tr>
<tr>
<td>Testing (Unit, Integration, User Acceptance Testing)</td>
<td>50%</td>
<td>58%</td>
<td>30%</td>
<td>46%</td>
<td></td>
</tr>
</tbody>
</table>
Colleges use their respective login to manage their faculty details, notices and circulars, departmental details, lab details, hostel and library information, student details with their attendance, grades and assignments and so on. Colleges can verify examination forms, enrollment forms, and admission forms. Students can use their login to submit or view admission form, examination form, enrollment form, examination fees, personal details etc.

The problems faced by end users in the web portal are discussed in the form of RC stores discussed in successive paragraphs. Therefore, web portal needs maintenance to update and include new features so that information to end users could be available in an efficient and convenient manner.

Requirements statement 1: The teachers of a college submit marks of students using online data entry form. Generally, teachers prepare MS-Excel sheet to maintain student evaluation record. It is observed that at the time of marks submission on college portal, the teacher has to submit data one by one into specific format containing text boxes. The same process of marks submission is repeated for all the subjects in all courses. But it is very time consuming and erroneous process. There is no other way to enter data if there are mass records. The huge data of students’ marks can be accepted in MS-Excel file format, which contains column entries. The RC story for this type of maintenance is as follows:

RC story 1:

Title: Upload marks from MS-Excel format

Maintenance type: New requirement

Description:

As a: User of college portal

I want: The ability to submit data in MS-Excel file format

So that: User can submit mass data properly and accurately in a short time
**Acceptance criteria:** The marks submission procedure has the facility to accept a document file in MS-Excel format through browsing and uploading process. The existing data entry form may be enhanced with browsing tool, which accepts an MS-Excel file and display all records one by one in existing text boxes and a save button, which can accept one record and display next record after submission.

Estimated story point- 15

**Requirements statement 2:** The teachers of colleges upload details of publication on the portal. After adding a publication, teachers want to modify some text. At the time of modification, instead of modifying existing copy, it creates a new record with modification as a new copy of the work. Here, system retains two same records for one entry, one with changes and another without modification. Portal holds existing record with the modification. The RC story for such maintenance is as follows:

RC story 2:

*Title:* Proper modification in publication  
*Maintenance type:* Corrective maintenance  
*Description:*  
*As a:* Teacher of college  
*I want:* The ability to modify publication details in a correct manner  
*So that:* Correct information of research papers is available for viewing

**Acceptance criteria:** When any modification is performed in a publication, it should save modification in existing record instead of adding copy of record with modification

Estimated story point- 5

**Requirements statement 3:** The teachers of college submit subject details on the web portal. After adding a subject record, teachers want
to save it. This feature works on limited web browsers. The RC story for such maintenance is as follows:

RC story 3:

*Title*: Insertion of subject details regardless of the web browser

*Maintenance type*: Adaptive maintenance

*Description:*

*As a*: Teacher of college

*I want*: The ability to submit subject details through any web browser

*So that*: Subject detail is available for viewing

*Acceptance criteria*: When subject details are filled by the teacher, it should be supported by all web browsers

*Estimated story point*: 10

The proposed technique on this maintenance case study can be applied to resolve aforesaid limitations of the web portal. Through judgment by analogy and expert opinion, it seems that the RC story 1, 2 and 3 will consume around 15, 5 and 10 story points, respectively. To determine phase wise effort in this project, proposed estimation technique, SMEEM, is applied. The SoM is calculated to be 30 using Equation (4.5). $E_{\text{Unadjusted}}$ is calculated using Equation (4.6), which is 60 person-days. The status of VAF in university web portal is calculated using Equation (4.7), and its FC value is 22. The project type is web application, and therefore, the CF is considered to be 0.20. Effort is calculated as 54.6 person days using Equation (4.8). If maintenance of university web portal performed by third party then phase wise effort computation using Equation (4.9) is shown in Table 4.6 and effort distribution is graphically shown in Figure 4.3. If maintenance of university web portal performed by the original developer then phase wise effort computation using Equation (4.9) is shown in Table 4.7 and effort distribution is graphically shown in Figure 4.4.
Table 4.6: Phase Wise Effort Computation in Case of Third Party Maintenance

<table>
<thead>
<tr>
<th>Phase Name</th>
<th>Effort (%)</th>
<th>Calculation of Effort</th>
<th>Effort (Person-Days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Understanding</td>
<td>45</td>
<td>45*.546</td>
<td>24.57</td>
</tr>
<tr>
<td>Impact Analysis</td>
<td>17</td>
<td>17*.546</td>
<td>9.28</td>
</tr>
<tr>
<td>Implementation/ Coding</td>
<td>18</td>
<td>18*.546</td>
<td>9.82</td>
</tr>
<tr>
<td>Testing (Unit, Integration, User Acceptance Testing)</td>
<td>20</td>
<td>20*.546</td>
<td>10.92</td>
</tr>
</tbody>
</table>

Figure 4.3: Phase Wise Effort Computation in Case of Third Party Maintenance
Table 4.7: Phase Wise Effort Computation in Case of Maintenance Serviced by Original Developer

<table>
<thead>
<tr>
<th>Phase Name</th>
<th>Effort (%)</th>
<th>Calculation of Effort</th>
<th>Effort (Person-Days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Understanding</td>
<td>17</td>
<td>17*.546</td>
<td>9.28</td>
</tr>
<tr>
<td>Impact Analysis</td>
<td>14</td>
<td>14*.546</td>
<td>7.64</td>
</tr>
<tr>
<td>Implementation/ Coding</td>
<td>23</td>
<td>23*.546</td>
<td>12.55</td>
</tr>
<tr>
<td>Testing (Unit, Integration, User Acceptance Testing)</td>
<td>46</td>
<td>46*.546</td>
<td>25.11</td>
</tr>
</tbody>
</table>

Figure 4.4: Phase Wise Effort Computation in Case of Maintenance Serviced by Original Developer
4.8 SUMMARY

Estimation is equally important during maintenance process of a software product as in the new development. There exist fewer effort estimation models for software maintenance as compared to software development. The existing models of maintenance estimation are based upon traditional software development methodologies, which produce realistic result for same methodology. If existing models are applied in agile and extreme programming based maintenance projects, then these models could lead to unrealistic results. Agile methodology has its own non-algorithmic estimation techniques based on analogy. In case of maintenance performed through iterative maintenance life cycle using extreme programming, there is a need of algorithmic approach for maintenance estimation. In this study, SMEEM is proposed, which is based on the story points to calculate the volume of maintenance. The SMEEM of effort estimation incorporates VAF with different intensity levels. The proposed model is illustrated with various types of maintenance projects. It is designed to help the maintenance manager to calculate the estimated software maintenance effort in terms of ASP, size, cost and duration.

Phase wise effort estimation for maintenance projects is also crucial to determine the accurate estimates. To compute phase wise effort estimation, an empirical study was conducted through questionnaire to determine efforts of maintenance. For validation of proposed technique, it is applied on maintenance case study of a university web portal. It provides more realistic result as compared to the existing techniques on maintenance effort estimation and helpful in estimating phase wise maintenance effort in extreme programming based maintenance environment.