International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

A Practical Approach to Software Cost Estimation Using Stochastic Modelling

[1]Dr. Swati Saxena, [2]Dr. Shiv Kumar Singh, [3]Mr. Nargish Gupta, [4] Meena Malik, [5]Ankur Goyal

Submitted: 23/12/2023 **Revised**: 29/01/2024 **Accepted**: 07/02/2024

Abstract— Software cost estimation plays a very critical role in Software Project Management. If the cost of the software has not been estimated properly, it can have a drastic impact on the project execution and delivery. Traditional models for software cost estimation fail to model correctly, the cost components associated with the project. There is enormous research literature related to software cost estimation but only a handful of them relate to cost measures that include both software development and software support. This is mainly because, in recent years, there has been a remarkable change in the way the software is now developed and supported. Needless to say, the software exists everywhere from elementary education to nuclear reactors and from civil engineering to genetic engineering. As such, one cannot bind it into the same set of measures related to development and delivery. In this research, we focus on customers like hotels, airways, banking, etc., particularly massive ERP systems, for development and customization support. Such software once purchased, requires one or more support teams to ensure its availability for the client. The infrastructure teams usually maintain server support whereas the application maintenance teams provide customization and functionality support. In this research, we have developed a model that considers the fixed cost and the recurring costs associated with the software. The fixed cost is related to the cost of development whereas the recurring cost involves the cost associated with the cost of cloud/on-premise deployment and the cost associated with support teams. The contribution of this research is twofold. We have proposed a model that considers the largest set of parameters of cost-related estimation, aligned with both development and support, which is highly mapped to ERP-like software. To the best of our knowledge and belief, no existing research considers all these parameters. To make the analysis applicable to a number of case studies, we have fuzzified the parameters to make them align with linguistic hedges. The possible deviations in the cost computation are estimated using linear regression ML models. We have considered the supports and customization part in accordance with modern bug-tracking tools like JIRA. The analysis is done for the case of an educational ERP with LMS and compared the result with those available as open-source in the UCI repository.

Keywords*: Software Cost Estimation, Fuzzy Logic, Linear Regression Analysis, JIRA*

Introduction

Software is the driving force behind modern society. We have software for essentially everything from ICUs of Hospitals to Space Stations, from Humidity Sensors to driverless cars, and it's infinite. Associated with the usage, domain, technology, device, etc., we have several classification schemes, like system software, application software, embedded systems, real-time systems, management systems, standalone-web-based systems, etc. and this classification criterion is also endless. A given software can fall into one or more of these criteria. In this study we have investigated the type of software which is on Cloud /Web Based, developed by a Team of Developers for the management of workflow of some organization, and also has a development and one or more support teams to ensure its availability and maintenance to the end users. The term, software cost estimation is a task to determine the overall cost expense associated with the software product. This estimate must be conveyed to the customer at the time of the agreement. Also**, as**

1Department of Mathematics Sagar Institue of Science & Technology, Bhopal

2Department of Mathematics Sagar Institue of Science & Technology, Bhopal

3Department of Computer Science & Engineering Sagar Institue of Science & Technology, Bhopal

4Department of Computer Science & Engineering Chandigarh University, Mohali.

5Department of CSE, Symbiosis Institute of Technology, Symbiosis International Deemed University, Pune

Swatisaxena@sistec.ac.in, nargishgupta@sistec.ac.in, Shivkumar@sistec.ac.in, meenamlk@gmail.com, Ankur_gg5781@yahoo.co.in

it is an estimation, related to a sizeable entity, and also includes the recurring component, must be done very carefully, failing which can cause disaster to the customer business.

This research focuses upon the estimation of the software cost which includes both the costs of development and maintenance of the software. Thus it includes the one-time cost as well as recurring costs. This study is based purely on the agile practices which are now adopted by almost all the software development / support organization.

Problem Statement

Software development and support practices may have varying scenarios. Not every time it is the same organization that both develops and maintains the software. At times, there is one organization that develops the software which may be maintained by another organization, which specializes in IT support. This is applicable in those industries for which there is a popular software product, by one company, which is then taken care of by another organization for the client. Popular software architectures for this type of scenario consist of (at least) two layers; a core part and a customization layer specific to the client. The customization itself includes the development of new features specific to the client, as well as software support for the existing features. Moreover, the infrastructure support; server loads, memory-usage in servers, etc., need to be monitored or at times, need special scripts to automate this task. A popular tool for server monitoring, at the time of this writing, is Grafana, which has a built-in/available of customization "Alert Engine". All this comes blurred between software development and software support. As such, it seems worth researching, to develop a cost estimation model which is applicable in such realistic scenarios of the modern software industry.

Considering both these scenarios, the cost of software estimation must be adjusted correspondingly. We attempt to develop the software cost estimation model which considers all these possibilities. To the best of our knowledge, no existing study has included both these scenarios.

Literature Review

One of the most fundamental type of cost estimation models in software engineering is the COCOMO model. This model is based upon regression and takes up as input, either one of the two parameters, namely Lines of code (LOC) or Function Points (FP). As lines of code cannot be taken as a true parameter of measure (with the advent of on-the-shelf software components), only FP is relevant for almost all cost estimation practices.

A review of most cost estimation techniques presented in the literature uncovered represents the primary remarkable dissimilarity among all the estimation models is for the LOC considered as the prime input models and which uses FPs. The state-of-the-art cost estimation models use LOC, which was selected early as a metric due to its quantifiability and seeming objectivity. This was the era when we had most three and four available choices of programming languages in the tech stack. By following it, a complete domain has developed in order to decide the best LOC counting scheme []. Gradually, with experts' objections for estimating LOC in advance for a project, the new developed models suggested not to use SLOC as the most important input []. The new model take different parameters to objectively estimate the cost of the software. Most such models use function-points. To determine well regarded models there is a need to consider two important factors: Firstly, the inventor of famous COCOMO model aka Barry Boehm has presented a detailed analysis of all the important models in his book entitled Software Engineering Economics [5]. The candidate generation was done with the help of list. Secondly, the review presented in various article in the Journal of Parametrics, contributes major portion of the articles for representing the software estimation. The resulting scenario thereby validate the Boehm work for most cited articles. Boehm evaluation and examination provide eight different models in his work as follow: SDC, COCOMO, Wolverton, Doty, SLIM, PRICE, Boeing , IBM- FSD. The detailed investigation for candidates list, an analysis of latest issues of the Journal of Parametric is very helpful, in order to express the popularity of all the models in other research work by demonstrating all the parameters.

The fundamental factors to define the quality of the software products, that are presented as a result of the Cocomo model are schedule and effort:

1. Effort: The number of people required for the task completion represents the effort required and the measurement unit is person per months.
2. Schedule: Represents the total time that is essential for the job completion, obviously that will be always

proportional to the effort placed. The measurement unit for time is represented in weeks, or months.

At different project levels, several Cocomo Models have already been offered in order to predict the cost estimation on the basis of correctness and accuracy. These models are applicable to a wide range and type of projects, the features will be determining factor for the constant value need to be applied in the following calculations. These features refer to different types of systems as mentioned below.

As per Boehm's description of projects for different types such as organic, semidetached, and embedded systems are defined as follows:

1. Organic – A software project falls under the category of organic type when the problem domain properly understood, it is previously solved already means the team is having a little exposure of the domain and the size of team is effectively small.
2. Semi-detached – A software project comes under this category when the dynamic features for example team-size, the work experience of team members, the expertise level in different programming environment falls in between to the dynamic features of organic type and Embedded type. The projects under this category are relatively challenging to develop in comparison to the organic type and demands additional experience and more expertise in order to deliver better creativity and guidance. Eg: All types of Compilers and Embedded Systems come under Semi-Detached type.
3. Embedded – A software project which involve high complexity and demands more expertise and creativity, falls in this category. This type involves a big team size as compared to the other two types and the developers must possess adequate experience and innovative level in order to develop such complex models.

Types of Models:

In COCOMO, the hierarchy contains three comprehensive and precise forms. For our requirements any one out of the three practices may be used as per the problem requirements. The three types of models are as below:

1. Basic COCOMO Model
2. Intermediate COCOMO Model
3. Detailed COCOMO Model

The Basic COCOMO may be applied for considerably rough and fast Software Costs calculations. But the accuracy of this model is restricted to some extent due to the lack of necessary factors for considerations. The second model i.e. Intermediate COCOMO considers Cost Drivers as important criteria and the third model that is Detailed COCOMO furthermore considers the individual project phases as an influence for deciding the actions, so in case of Detailed model, all cost drivers along with the calculations need to be performed phase-wise thereby resulting in a more clear and accurate result.

Model Development

Before starting the model equations which describe the estimation for the software cost, we enlist here the model behavior and also depict the corresponding flow in the form of a flow diagram. Throughout this writing, we have used the term Issue/JIRA interchangeably to indicate any flaws in the software working.

Fact#1: Agile software development follows a practice in which the software is delivered to the customer in iterations wherein each iteration enhances the desired functionality.

Corollary#1: In the case of large software, the core functionality is delivered to the customer at once and the other enhancements to be made are put in the form of "user-stories", which are prioritized.

Corollary#2 The enhancements are picked up by the development team, based on the recommendations of "system specialists" for the enhancements. New user stories are created, if there is a change required in the existing/desired functionality.

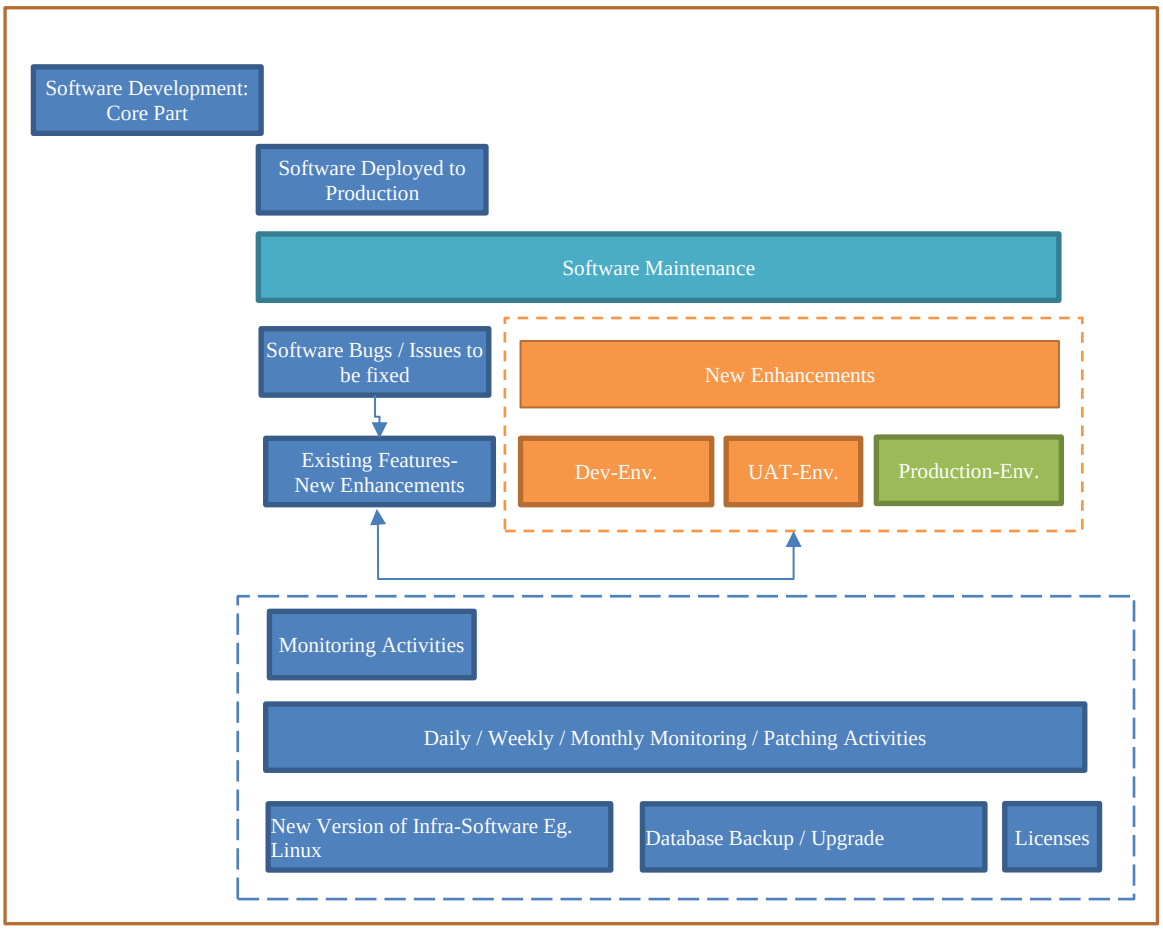
Fact#2: Once the core part of the software is tested and released to production, the new enhancements need to be tested thoroughly before these changes are merged into the core software.

Corollery#1: There is a need for thorough testing of the new feature before it can be merged into the main software which is running in a live environment. We assume, as in most cases, there are three environments that are deployed by the development/maintenance organizations; viz, Development Testing and Production Environment. For practical reasons, Test Env. Is usually called UAT (User Acceptance Testing) Environment.

Corollery#2: As suggested by the above corollary, the distinction between software development and

software support has become blurred. An issue faced by the customer in the production environment might have the only resolution in the form of a new feature to be developed, by the support team. Such issues are registered in the form of new user stories.

Corollery#3: An issue/incident reported by any user might end up in a support bug/ JIRA, which needs to be handled like any other JIRA issue in the backlog.



Typical Workflow of Software Development and Support: Agile Methodology

Model Pseudocode:

Variables for Initialization and Possible Values

The variable names and enumerated values are listed in Table 1.

Table 1

S. No.	Variable Name	Enumerated Values
1	SOFTWARE ACTIVITY	DEV, SUPPORT, BOTH
2	SUPPORT ACTIVITY	L1, L2, L3, L1&L2, ALL
3	L1 Support	Control Panels, Call, Both
4	L2 Support	Adding/Modifying DB, Debug Operation, ALL
5	L3 Support	Debug Existing Features, Develop New Features, Both
6	New Prod Release	Weekly, Monthly, Quarterly
7	System Specialist	On Team, On Client

8	Infrastructure	On-Premise, On Cloud, Scattered
9	Third-Party Software Licensing	On Team, On Client
10	Incident Frequency	<2 Per Day, <5 Per Day, <10 Per Day, >10 Per Day
11	Severity of Incident	Often low Severity, Often High Severity, Both
12	Incident Resolution	Within 3 Hrs, Within 8 Hrs, Within 24 Hrs., Mixed of Incident Types

Workflow:

For a steady state, we assume that software is deployed and available to the users in the production environment. The rate of generation of events is defined in Table 2. These rate values mention the rate of generation of issues and the rate of resolution. With these rates, we can define the steady state equations which give the threshold values of the required parameters which are fundamental values for the estimation if the cost of the software.

Table 2

S. No.	Rate Parameter	Description
1	λ_h	Rate of issue reporting by the stakeholder (Sev#High)
2	λ_m	Rate of issue reporting by the stakeholder (Sev#Medium)
3	λ_l	Rate of issue reporting by the stakeholder (Sev#Low)
4	γ	(Variable) Rate of picking up issues from the backlog by the developer Team
5	$\boldsymbol{\mu}_h$	Rate of Fixing up of the issues reported with sev#high
6	$\boldsymbol{\mu}_m$	Rate of Fixing up of the issues reported with sev#Medium
7	$\boldsymbol{\mu}_l$	Rate of Fixing up of the issues reported with sev#Low
8	ξ	Rate of issues getting fixed, the issues which are picked from the backlog

We these rates we can find the steady-state probabilities of the size of the development team. To keep the calculation of cost estimation simple, we can safely assume that all the developers in the development team have the same expertise related to the issue.

We formulate the recurrence equations, describing the model on the basis of queueing theory. For the following equations, we consider the following formulation:

$$\begin{aligned} &Probability\ of\ n\ customers\ in\ queue\ at\ time\ t \\ &= \left(Probability\ of\ n-1\ customers\ in\ the\ queue\ at\ time\ (t-1)\right) \\ &* Probability\ of\ ONE\ arrival\ in\ time\ \Delta t * Prbability\ of\ NO\ departure\ in\ time\ \Delta t \\ &+ \left(Probability\ of\ n\ customers\ in\ the\ queue\ at\ time\ (t-1)\right) \\ &* Probability\ of\ NO\ arrival\ in\ time\ \Delta t * Prbability\ of\ NO\ departure\ in\ time\ \Delta t \\ &+ \left(Probability\ of\ n+1\ customers\ in\ the\ queue\ at\ time\ (t-1)\right) \\ &* Probability\ of\ NO\ arrival\ in\ time\ \Delta t * Prbability\ of\ ONE\ departure\ in\ time\ \Delta t \end{aligned}$$

We consider the following equation for all three types of incidents as mentioned in Table 2.

The Probability of n incidents (Priority High) waiting in the queue to get fixed, at any given time, is given by the following recurrence relations.

$$P_{nh} = (P_{n-1} * \lambda_h * (1 - \mu_h)) + P_n * (1 - \lambda_h) * (1 - \mu_h) + P_{n+1} * \mu_h * (1 - \lambda_h)$$

The Probability of n incidents (Priority Medium) waiting in the queue to get fixed, at any given time, is given by the following recurrence relations.

$$P_{nm} = (P_{n-1} * \lambda_m * (1 - \mu_m)) + P_n * (1 - \lambda_m) * (1 - \mu_m) + P_{n+1} * \mu_m * (1 - \lambda_m)$$

The Probability of n incidents (Priority Low) waiting in the queue to get fixed, at any given time, is given by the following recurrence relations.

$$P_{nm} = (P_{n-1} * \lambda_m * (1 - \mu_l)) + P_n * (1 - \lambda_l) * (1 - \mu_l) + P_{n+1} * \mu_l * (1 - \lambda_l)$$

The following are the constraints for steady state:

$$\lambda_h < \mu_h$$

$$\lambda_h + \lambda_m + \lambda_l < \mu_h + \mu_m + \mu_l$$

To ensure the fixing in the backlog, we can state the above equation like:

$$\lambda_h + \lambda_m + \lambda_l + \gamma = \mu_h + \mu_m + \mu_l + \xi$$

To ensure the "rate of enhancements" in the software to keep above a threshold.

The steady state queue length is given by the equation:

$$L = \frac{\lambda}{\mu - \lambda}$$

If (**μ**-λ) is kept as a random variable between 0 and α, then the length of the queue is directly proportional to the arrival rate as shown in the figure 1 given below:

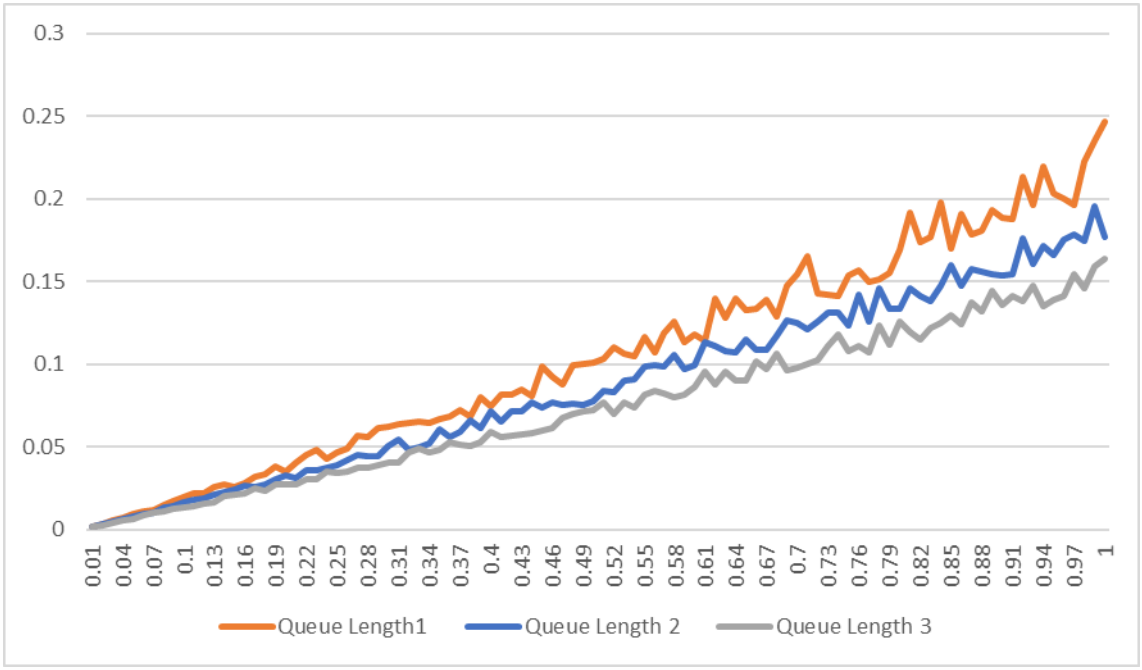


Fig 1: Stationary Values of Queue Length for Varying values of Service Rate

The above values indicate that queue length increases with the increase of arrival rate of incoming service request for support bugs and issues. The values given here are normalized between 0 and 1. However these give a count of the efforts required with respect to incoming requests.

The above graph presumes the steady state conditions that are necessary and sufficient condition stated in above equations, which indicates that the incoming rate of issues/bugs/enhancements are less than the rates at which these are fixed by the team.

For drawing a more concerte interpretation of the above stated equations, we need to consider the case study of a software project managed by a Team of Technical staff. To keep the model close to the practical scenario, we consider the case study of a Hospital Management System which is developed and maintained by a set of Teams each consisting of specific number of individuals. We conducted study of number of hospitals management system available from popular vendors to make a set of features applicable to out hypothetical model to be used in this case study. These are mentioned below in tabular format.

Table 2

Features Of The Software: Hospital Management System

S. No.	Feature Specification
1	Patient Registration
2	Appointment & Scheduling
3	Outpatient Management
4	Inpatient Management
5	Billing
6	Discharge Summary
7	Laboratory Management
8	Radiology Management
9	Pharmacy Management
10	Consultant Management
11	Inventory Management
12	Security Management
13	Health Records Data
14	Reception Management
15	Web Portal
16	MIS Reports
17	Analytics & Dashboards
18	Accounts Management
19	Mobile Apps
20	Biometric Device Interface
21	RFID Interface
22	Email Integration / SMS Integration
23	Web Camera Interface
24	Voice-To-Text Integration
25	Ambulance Management
26	Blood Bank Management

27	Canteen Management
28	Attendance Management
29	Feedback Management
30	Nurse Station
31	Operation Theatre Management
32	Equipment Maintenance Management

The above-mentioned software has 32 different modules. This typically represents a large class of software deployed for hospitals. This also represents, to a large extent, software used in Academic Institutions, Railways, Banks, etc.

Although, each module has its own specific size and further granularity in features, for the sake of simplicity, we can assume that on an average, there are N number of features in each module.

We presume a scenario wherein the core product is delivered to the client and the enhancement features are prioritized in the backlog, to be delivered to the client along-with the maintenance of the software.

We tabulate below, the parameters which represents the typical values, averaged, for such size of software.

Table 3

Technical Team Description

S. No.	Team Name	Team Size
1	Software Development Team	S_{sd} ~15
2	Database Team	S_{db}~5
3	IT/Software Support Team (Level 1/ Level 2)	S_{it}~12 (4 per shift of 8 Hrs)

Out of the 32 modules listed above, each, on an average having N features, we have a total of 32N features in the software. To keep the calculations further simplified, we assume that, on an average, from N features of each module, n features form the core part and N-n are to be delivered to the client in the form of enhancements.

If each feature takes “m” person-hours, then the total efforts involved in the development of the core part of the software requires a total effort of 32XnXm. The enhancements are to be prioritized by the system specialists, along with the regular maintenance activities.

We consider the following typical (realistic) values of the parameters:

Table 4

Model Values

Parameter	Values
Module Count	32
Sub-Feature Avg. Value	20
Core Features	8
Enhancement Features	12
Each Feature Dev. Time	12 Person - hrs
Each Feature Testing Time	4 Person-Hrs

Core Part Architecture Design Time	3 Months
Time for Development	3072
Development Team Size	12 Developers
Testing Team Size	3 Testers
Time for Development	1.6 Months (Considering 40 hrs per week effort by individuals)
System Testing (Feature-wise)	4 hrs/Feature
Testing Time	2.133333
Total Delivery Time	3.7

With these features, the time for development will be 1.6 months and the time for testing will be 2.13 months. As the Core Part Architecture design time is approx. 3 months, the delivery time for the core part of the module comes out to be 7 months. Neither of these activities can be done in parallel for the core part of the development.

A variation of the number of features from 20-32, with a development team size varying from 6-12, along with 2-4 testers gives a delivery time from 7 months to 1.5 years for the core part of the software.

The maintenance of the software is an ongoing activity which can be taken care by the same or different team. The features for enhancements can be kept in backlog to be picked by the team maintaining the software.

With the above values of the model parameters, we can compute the rate of enhancements to be added to the software, provided we have the rate of issues/bug fixes as demanded by the users of the software.

The figure 2 gives an estimate of time-to-deliever core software on the basis of the above parameters. In this figure, the horizontal scale gives the number of modules in the software, with each module consisting of approx. 12 core features.

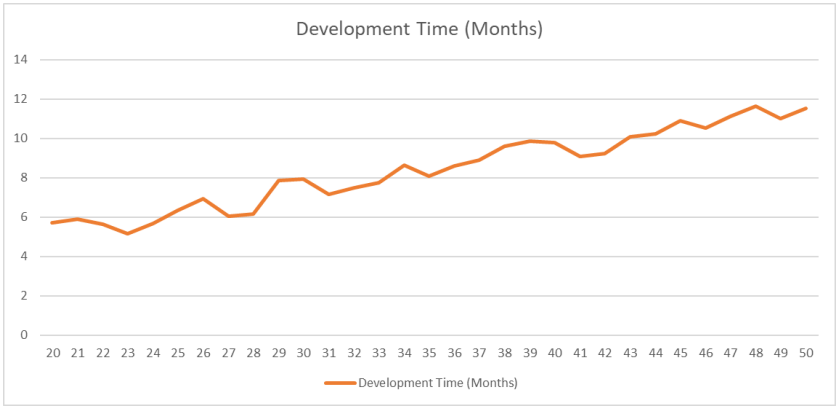


Fig 2. Variation of time-to-deliver (in-months) the core part with the count of modules in the software.

Figure 3 shows the variation of time to delivery of the core software module with respect to the count of features in each module which comprises the core part of software.

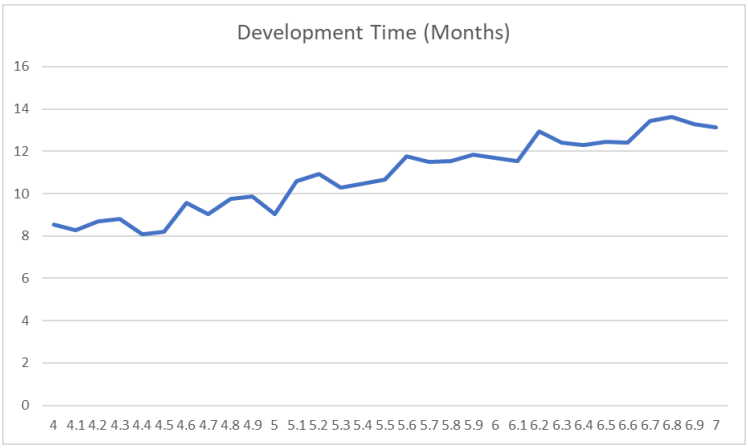


Fig 3. Variation of time-to-deliver (in months) the core part with the count of features in the modules in the software, assuming average of 25 modules in the software.

Conclusion And Future Scope

The given literature gives a practical approach towards estimation of the efforts required in any development and support project managed by a team of developers in which the team handles the support issues and new enhancements of the software. This study matches best in its class by inculcating all the workflow and practices handled by modern software development teams in current scenarios.

As a future scope of the current work, we will be modelling fuzzy logic based model equations for finding the steady state values of queue length the required count of manpower for managing the software project.

References

[1] P. Rodríguez et al., Continuous deployment of software intensive products and services: a systematic mapping study J Syst Softwe (2017)

[2] M. Jørgensen, Top-down and bottom-up expert estimation of software development effort, Inform Softw Technol (2004)

[3] S.A. Butt et al., A software-based cost estimation technique in scrum using a developer's expertise, Adv Eng Softw (2022)

[4] M. Kuhrmann et al., What makes agile software development agile

[5] IEEE Trans Software Eng, (2021)

[6] A. Khalid et al. Agile scrum issues at large-scale distributed projects: scrum project development at large

[7] Int J Softw Innov (IJSI), (2020)

[8] R. Kaim et al. , Benefits of agile project management in an environment of increasing complexity—a transaction cost analysis Intelligent decision technologies 2019, (2019)

[9] Y. Khmelevsky et al., Software development using agile and scrum in distributed teams

[10] A. Rasheed et al., Requirement engineering challenges in agile software development, Math Probl Eng, (2021)

[11] M.A. Ramessur et al., A predictive model to estimate effort in a sprint using machine learning techniques, Int J Inform Technol, (2021)

[12] N.A. Obilor et al., Constructive cost model II metrics for estimating cost of indigenous software, Int J Adv Eng Res Sci, (2021)

[13] S.A. Butt et al., Frequent change request from user to handle cost on project in agile model, Proc Asia Pacific J Multidiscipl Res, (2017)

[14] S.M.R. Chirra et al., A survey on software cost estimation techniques, J Softw Eng Applic, (2019)

[15] J. Shah et al., Extending function point analysis effort estimation method for software development phase

[16] V. Venkatesh et al., How agile software development methods reduce work exhaustion: Insights on role perceptions and organizational skills Inform Syst J (2020)

[17] I. Kaur et al. Neuro fuzzy—COCOMO II model for software cost estimation Int J Inform Technol (2018)

[18] M. Jorgensen Relationships between project size, agile practices, and successful software development: results and analysis IEEE Softw (2019)

[19] N. Rankovic et al. A new approach to software effort estimation using different artificial neural network architectures and Taguchi orthogonal arrays IEEE Access (2021)

[20] L. Villalobos-Arias et al. Evaluating hyper-parameter tuning using random search in support vector machines for software effort estimation

[21] H. Rygge et al. Threat poker: Solving security and privacy threats in agile software development H.L.T.K.

[22] Nhung et al., A review of use case-based development effort estimation methods in the system development context

[23] S. Shekhar et al. Review of various software cost estimation techniques, Int J Comput Applic (2016)

[24] S. Dalal et al. Efficient tuning of COCOMO model cost drivers through generalized reduced gradient (GRG) nonlinear optimization with best-fit analysis Progress in Advanced Computing and Intelligent Engineering (2018)