INNOVARE JOURNAL OF ENGINEERING & TECHNOLOGY

Vol 11, 2023

ISSN - 2347-1573

Review Article

LEARNING THE EFFICIENT ESTIMATION TECHNIQUES FOR SUCCESSFUL SOFTWARE PROJECT MANAGEMENT

KHIN SHIN THANT[1], HLAING HTAKE KHAUNG TIN[2]*

[1]Department of Information Science, University of Computer Studies, Hinthada, Myanmar, [2]Department of Information Science, University of Information Technology, Yangon, Myanmar. Email: hlainghtakekhaungtin@gmail.com

Received: 20 May 2023, Revised and Accepted: 23 May 2023

ABSTRACT

The process of software project management involves planning and supervising the development of software projects to deliver a quality product within the customer's budget and schedule. This process begins with project planning, which includes estimating the work to be done, required resources, and project schedule. Once these activities are accomplished, a project schedule is established that defines software engineering tasks, identifies responsible parties, and specifies inter-task dependencies. The paper aims to explore project management activities and techniques for estimating project size. Overall, software project management involves managing, allocating, and timing resources to develop software that meets requirements and is delivered within budget and schedule. This paper highlights the significance of employing efficient estimation methods to achieve successful software project management. Estimation plays a critical role in the software development process, as it helps project managers to determine the resources and time required for completing the project. The paper starts by introducing the concept of software project management and highlighting its importance in delivering successful projects. The advantages and disadvantages of each technique are discussed, and guidelines for selecting the appropriate technique for a specific project are provided. The paper also explores the importance of accurate estimation in agile software development and the use of estimation tools to simplify the process. Finally, the paper concludes by summarizing the key takeaways from the discussion and emphasizing the significance of efficient estimation techniques in ensuring successful software project management.

Keywords: Software engineering, Lines of code, Function point, Project management activities, Estimation techniques.

 DOI: http://dx.doi.org/10.22159/ijet.2023v11i3.47605. Journal homepage: https://innovareacademics.in/journals/index.php/ijet

INTRODUCTION

Software project management is a discipline of planning and supervising software projects' development process activities to deliver a quality product within the customer's budget and schedule. Estimation is a crucial aspect of software project management that involves determining the resources and time required to complete the project successfully. Accurate estimation can help project managers to make informed decisions, identify potential risks, and avoid project delays and cost overruns. On the other hand, inaccurate estimation can lead to significant project failure and negative impacts on the project team, customer, and organization.

This paper aims to explore the efficient estimation techniques for successful software project management. It discusses the importance of software project management, the part of estimation in the software development process, and the consequences of inaccurate estimation. The paper provides an overview of various estimation techniques. It examines the advantages and disadvantages of each technique and provides guidelines for selecting the appropriate technique for a specific project.

Furthermore, the paper discusses the importance of accurate estimation in agile software development and the use of estimation tools to simplify the process. Finally, the paper concludes by summarizing the key takeaways from the discussion and emphasizing the significance of efficient estimation techniques in ensuring successful software project management.

LITERATURE REVIEW

Software project management [1] involves planning, monitoring, and controlling the development process of software projects to ensure that they are completed on time, within budget, and to the required quality standards. The estimation of project resources is an essential part of the project planning process, which involves estimating the effort, cost, and duration required to complete the project successfully. In recent years, several techniques have been proposed for efficient estimation of software projects. This literature review aims to explore these techniques and their effectiveness in software project management.

Efficient estimation of software projects is crucial for successful project management. Several techniques, including function point (FP) Analysis, COCOMO Model [2], Agile Estimation Techniques, and Machine Learning-Based Estimation Techniques, have been proposed for software estimation. These techniques have their strengths and weaknesses, and the choice of technique depends on the project's requirements and constraints. However, the use of these techniques can improve the accuracy of project resource estimation and help in successful software project management.

Project monitoring and control activities begin once development activities commence. They are focused on ensuring that the software development proceeds as planned. During this stage, the project manager may need to adjust the plan to address specific situations that arise. Effective project monitoring and control are essential to the success of a software project [3].

In this session, the software project management is discipline of planning and supervising software projects development process activities to deliver a quality product, keeping the cost within the customer's budget and deliver the project as per schedule. It involves a procedure of managing, allocating, and timing resources to develop software that fulfills requirements.

Software project management is essential to implement user requirements, along with budget and time. A project manager is responsible for managing a project who essentially works as the

administrative leader of the team and is an experience member of the project team. Software project management is much more complex than management of many other types of projects due to invisibility, changeability, complexity, uniqueness, exactness of the solution, and team-oriented and intellect-intensive work [1].

Projects are being carried out at all levels of the organization. Their time period ranges from a few weeks to more than 5 years. Projects can include a single unit of an organization or cross institutional boundaries, such as joint ventures and partnerships.

PROPOSED SYSTEM WITH TWO SIZE ESTIMATION TECHNIQUES

At present, project managers may utilize diverse methodologies and techniques for managing software projects across various organizations. However, there are five core activities [4] that are universally recognized as fundamental to software project management across all organizations.

Project planning

Project planning is the process of defining the objectives and goals of a project, determining the tasks needed to achieve those objectives, and estimating the resources required to complete those tasks. It involves creating a roadmap for the project that outlines the scope, timeline, budget, and deliverables. Effective project planning ensures that the project is well-organized, well-managed, and delivered on time and within budget. Project planning includes activities such as feasibility study, requirements analysis, specification, and project schedule development. It is a critical activity in software project management that sets the foundation for the project's success.

Risk management

Risk management is the process of identifying, assessing, and controlling risks that may affect the success of a software project. It involves identifying potential risks that may arise during the project's lifecycle and implementing measures to mitigate their impact. The goal of risk management is to minimize the likelihood and impact of negative events on the project's objectives and outcomes.

The risk management process involves four steps: risk identification, risk assessment, risk response planning, and risk monitoring and control. In the risk identification phase, the project team identifies potential risks, such as technical risks, operational risks, and external risks. In the risk assessment phase, the team analyzes and prioritizes the risks based on their potential impact and likelihood of occurrence. In the risk response planning phase, the team develops strategies to mitigate the risks, such as avoiding, transferring, mitigating, or accepting the risks. Finally, in the risk monitoring and control phase, the team continually monitors the project's risks, evaluates the effectiveness of the risk response strategies, and adjusts them as necessary. Effective risk management is critical to the success of a software project, as it helps project teams to proactively identify and manage risks, minimize project delays, and avoid cost overruns.

People management

People management is the process of managing the human resources involved in a software project. It involves managing the project team's recruitment, selection, training, motivation, and performance evaluation. Effective people management ensures that the project team has the necessary skills, knowledge, and resources to complete the project successfully.

People management [1] in software project management includes several activities, such as team formation, team building, communication management, conflict resolution, and stakeholder management. Team formation involves selecting and assembling a project team that has the necessary technical and interpersonal skills to complete the project. Team building involves developing a cohesive team culture, establishing team norms, and fostering collaboration and communication among team members.

Communication management involves establishing effective communication channels between project stakeholders, team members, and other project participants. Conflict resolution involves identifying and resolving conflicts that may arise during the project's lifecycle. Finally, stakeholder management involves identifying and managing project stakeholders, such as customers, end-users, sponsors, and other project participants. Effective people management is critical to the success of a software project, as it ensures that the project team is well-managed, motivated, and equipped to complete the project successfully. It also helps to foster a positive project culture and maintain good relationships with project stakeholders.

Reporting

Reporting in software project management involves the regular and systematic collection, analysis, and dissemination of project-related information to project stakeholders. The goal of reporting is to provide project stakeholders with timely, accurate, and relevant information about the project's progress, status, and performance. Effective reporting helps stakeholders to make informed decisions, manage project risks, and ensure that the project stays on track.

Reporting in software project management includes several activities, such as data collection, data analysis, report generation, and report dissemination. Data collection involves collecting project-related data from various sources, such as project team members, project documents, and project management tools. Data analysis involves analyzing the collected data to identify trends, patterns, and insights that can inform decision-making.

Proposal writing

Proposal writing in software project management involves the process of preparing and presenting a document that outlines the project's scope, objectives, requirements, timelines, and budget to potential customers or stakeholders. The goal of proposal writing is to convince the potential customers or stakeholders that the project team is capable of delivering a quality product that meets their needs and expectations.

Proposal writing in software project management includes several activities, such as researching and understanding the customer's requirements and needs, developing a project plan that meets those requirements, and presenting the plan in a clear, concise, and persuasive manner. Effective proposal writing also involves developing a budget that accurately reflects the resources needed to complete the project successfully.

METHODS

Effective project planning requires not only a comprehensive understanding of different estimation techniques but also past experience. It is important for both the project manager and customer to recognize that software requirements variability can lead to cost and schedule instability. Project estimates should aim to define best-case and worst-case scenarios to help establish boundaries for project outcomes [5]. It is essential to adapt and update the plan as the project progresses to ensure it remains relevant and effective.

During project planning, a project manager is responsible for various activities such as cost, duration and effort estimation, scheduling, staffing, risk management, and miscellaneous planning [6]. The first activity undertaken by the project manager is size estimation, as size is the most fundamental attribute that affects the accuracy and effectiveness of estimates and project planning. The project size is critical because as it increases, the interdependency among various software elements grows rapidly.

The size of a software project [1] can be measured by the amount of effort and time necessary to develop the product, and it can be estimated by taking into account factors such as the project type, application domain, functionality delivered, and the number of components. There are various estimation techniques available, but two of the most commonly used metrics for estimating project size are mentioned.

Lines of code (LOC)

LOC is a software metric used to measure the size or complexity of a software program. It counts the number of lines of source code in a program [7]. It is a simple and widely used metric that can give an indication of the scale of a project or the effort required to develop it. However, it has some limitations as it does not account for differences in programming languages, coding styles, or the quality of the code. In addition, as software development increasingly involves the use of third-party libraries and frameworks, the use of LOC as a metric may not accurately reflect the true size or complexity of a project.

Despite these limitations, LOC can still be a useful metric when used appropriately and in combination with other metrics and estimation techniques. When used to measure problem size, LOC has several disadvantages Table 1.

Overall, while LOC can be a useful metric for measuring code size, it should be used in conjunction with other metrics to provide a more comprehensive picture of the size and complexity of a software project.

Based on the given information, the estimated LOC Table 2 for the project is 33,200. The average productivity of the organization for a system is 620 LOC per person-month, which means that it will take approximately 54 person-months (33,200/620) to complete the project. The labor rate of the organization is $8,000 per month, so the total cost of the project can be estimated by multiplying the labor rate by the estimated effort in person-months, which is: 54 person-months * $8,000 per month = $432,000. The estimated cost per line of code is around $13, so the total cost of the project can also be estimated by multiplying the number of LOC by the estimated cost per line of code, which is: 33,200 LOC * $13 per line of code = $431,600.

These two estimates are quite close and suggest that the projected total cost of the project is around $431,000 to $432,000 with an estimated effort of 54 person-months.

FP

FP is a software measurement technique used to measure the functional size of a software application. It is based on the concept that software functionality can be quantified by the business transactions that the software performs. FPs measure the functionality of software by quantifying the inputs, outputs, inquiries, files, and interfaces required to support the business processes.

This method is independent of the technology used to implement the software and can be used to measure software of different types and sizes. The use of FPs enables more accurate estimates of project size, effort, and cost and provides a means to measure and compare the productivity of different development teams. FP computation Table 3 typically involves three steps.

Step 1: Compute the unadjusted FP (UFP) using a heuristic expression that considers the five components of functionality and their corresponding complexity weights.

Step 2: Refine the UFP. This involves adjusting the complexity weights of each FP component based on specific factors, such as data processing requirements, user interface complexity, and performance constraints.

Step 3: Compute the final FP count by additional refining. This involves applying adjustment factors for the general system characteristics, such as distributed processing, transaction rate, and operational ease. The result is a more precise estimate of the effort, cost, and schedule necessary to develop the software system.

FINDINGS AND DISCUSSION

Software project management involves various estimation techniques to ensure the success of software projects. One of the commonly used techniques is the FP analysis, which measures the size of a software system in terms of its functionalities. FP analysis has three main steps: computing UFP, refining UFP based on the complexity of parameters, and further refining UFP based on the project's specific characteristics. The technical complexity factor (TCF) is then computed to adjust the estimated effort and cost of the project based on the degree of influence (DI) of various project parameters. However, FP analysis has some disadvantages, such as not considering the algorithmic complexity of a function. Another technique used in software project management is LOC, which measures the size of a software project based on the number of LOC. Overall, choosing the most suitable estimation technique for software project management depends on various factors such as project requirements, available data, and expertise.

Step 1. UFP Computation

The computation of UFP involves using a heuristic expression that takes into account five different parameters of the software application. The equation for UFP computation is as follows:

UFP = (Count of External Inputs × Weight for External Inputs)+(Count of External Outputs x Weight for External Outputs)+(Count of External

Table 1. Disadvantages of LOC

1	Different programming languages have different levels of abstraction, which can lead to different LOC counts for the same functionality, making it difficult to compare projects written in different languages.
2	LOC measures the size of the code, but not the complexity of the code or the functionality it provides. Therefore, two programs with the same number of LOC may have different levels of functionality or complexity.
3	LOC does not take into account the quality of the code, such as readability, maintainability, or performance.
4	LOC can be easily manipulated by using code generators or copy-pasting code, which can result in an inaccurate measurement of the actual problem size.
5	LOC does not consider the reuse of code or the use of external libraries or components, which can significantly impact the size and complexity of a project.

Table 2. Estimation table for the LOC methods

Function	Estimated LOC
User interface and control facilities	2300
Two-dimensional geometric analysis	5300
Three-dimensional geometric analysis	6800
Database management	3350
Computer graphics display facilities	4950
Peripheral control function	2100
Design analysis modules	8400
Estimated lines of code	33200

Table 3. Function Point Computation Steps

No	Function Point Computation Steps
1	Identify the functions that the software system will perform, based on the requirements.
2	Categorize these functions into different types, based on their complexity and how they will be implemented in the system. These categories include inputs, outputs, inquiries, internal files, and external interfaces.
3	Assign a weight to each category, based on the complexity of the functions in that category, and then compute the total function points by adding up the weighted scores for each category.

Table 4. Five Categories of Parameters

Parameters	Description	Complexity
External Inputs (EI)	These are user interactions that result in the system receiving data from an external source. Examples include user input screens, sensors, and data feeds.	The complexity of an EI is determined by the number of data elements that are maintained an the number of different data element types.
External Outputs (EO)	These are user interactions that results in the system producing data for an external source. Examples include reports, screen displays, and alarms.	The complexity of an EO is determined by the number of data elements that are produced and the number of different data element types.
External Inquiries (EQ)	These are user interactions that result in the system producing data from internal sources. Examples include inquires, search functions, and data retrieval.	The complexity of an EQ is determined by the number of files referenced and the number of data elements retrieved.
Internal Logical Files (ILF)	There are logical groupings of data that are maintained within the system. Examples include tables, databases, and files.	The complexity of an ILF is determined by the number of data elements and the number of different data element types.
External Interface Files (EIF)	These are logical groupings of data that are used by the system but are maintained by external sources. Examples include message files and data exchange files.	The complexity of an EIF is determined by the number of data elements and the number of different data element types.

Table 5. Example of Complexity Adjustment Factors

Type	Simple	Average	Complex
Input (I)	3	4	6
Output (O)	4	5	7
Inquiry (E)	3	4	6
Number of Files (F)	7	10	15
Number of Interface	5	7	10

Table 6. Function Point Relative Complexity Adjustment Factors

14 Parameters	Value 0 (not present/no influence) to 6 (strong influence)
Requirement for reliable backup and recovery	4
Requirement for data communication	3
Extent of distributed processing	2
Performance requirements	5
Expected operational environment	4
Extent of online data entries	5
Extent of multi-screen or multi-operation online data input	4
Extent of online updating of master files	4
Extent of complex inputs, outputs, online queries and files	4
Extent of complex data processing	4
Extent that currently developed code can be designed for reuse	4
Extent of conversion and installation included in the design	4
Extent of multiple installations in an organization and variety of customer organizations	3
Extent of change and focus on ease of use	5

Inquiries×Weight for External Inquiries)+(Count of Internal Logical Files x Weight for Internal Logical Files)+(Count of External Interface Files×Weight for External Interface Files)

where:

Count of External Inputs: the number of inputs to the system from external entities

Weight for External Inputs: a weighting factor that reflects the complexity of processing external inputs

Count of External Outputs: the number of outputs from the system to external entities

Weight for External Outputs: a weighting factor that reflects the complexity of producing external outputs

Count of External Inquiries: the number of queries to the system from external entities

Weight for External Inquiries: a weighting factor that reflects the complexity of processing external inquiries

Count of Internal Logical Files: the number of logical files maintained by the system

Weight for Internal Logical Files: a weighting factor that reflects the complexity of maintaining internal logical files

Count of External Interface Files: the number of interface files used by the system

Weight for External Interface Files: a weighting factor that reflects the complexity of using external interface files

The weights used in the heuristic expression are typically based on industry averages and can vary depending on the complexity of the application being developed. The parameters in Table 4 are classified into five categories:

Each of these parameters is assigned a weight factor that reflects the complexity of the parameter. The weight factors are determined by a set of rules defined by the international FP users group. The weight factors are used to adjust the UFP.

Step 2: Refine the UFP

FP relative complexity adjustment factors are used to refine the UFP count to reflect the complexities of the different parameters used in UFP computation. The following Table 5 shows an example of complexity adjustment factors for different types of functions:

For example, if a project has ten simple inputs, five average outputs, eight complex inquiries, 12 average number of files, and seven simple number of interfaces, then the total complexity adjustment factor would be calculated as follows:

$(10 \times 3) + (5 \times 5) + (8 \times 6) + (12 \times 10) + (7 \times 3) = 209$

This complexity adjustment factor of 209 would be multiplied by the UFP count to obtain the final FP count.

Step 3: Refined UFP based on complexity of the overall project

After computing the UFP in step 1, the next step Table 6 is to refine it to reflect the complexity of the overall project. This involves assigning

weights to the UFP based on various factors such as the complexity of the inputs, outputs, and user interactions. The weights are typically on a scale of 0 to 5, with 0 indicating no complexity and 5 representing maximum complexity.

For example, if a project has a high number of inputs, it would be assigned a weight of 4 or 5, whereas if it has only a few inputs, it would be assigned a weight of 1 or 2. Similarly, if a project has complex outputs, it would be assigned a weight of 4 or 5, whereas if the outputs are simple, it would be assigned a weight of 1 or 2.

Once the weights have been assigned to each of the components, they are multiplied by the corresponding UFP value to obtain a refined UFP. The refined UFP reflects the actual complexity of the project and is used in the final step to compute the FP.

The size of a project can be influenced by various project parameters such as high truncation rates, response time requirements, scope for use, and others. Hence, it is necessary to consider the overall project size while refining the UFP computed in step 2.

Based on the provided values for the 14 parameters, the TCF can be calculated as follows:

TCF = (0.65 + (0.01 * DI))

DI = (4 + 3 + 2 + 5 + 4 + 5 + 4 + 4 + 4 + 4 + 4 + 4 + 4 + 3 + 5) = 55

TCF = (0.65 + (0.01 * 55)) = 1.2

Therefore, the TCF is 1.2.

where,

TCF = a technical complexity factor,

DI = Total degree of influence.

DI can vary from 0 to 84 and TCF can vary from 0.65 to 1.49.

The TCF is calculated using the Total DI, which can vary from 0 to 84, and TCF can vary from 0.65 to 1.49. However, FPA has some limitations, such as not considering the algorithmic complexity of a function.

One disadvantage of FP analysis is that two functions with the same number of inputs and outputs may be given the same FP value, even if one of the functions requires much more complex processing than the other. In addition, the process of determining the appropriate weights for each input and output can be subjective and may vary depending on the person performing the analysis. Finally, it may not be suitable for certain types of projects or systems, such as those that involve significant hardware or infrastructure components.

CONCLUSION

Software project management is a challenging task that requires a lot of planning, coordination, and efficient estimation techniques. The success of a software project is dependent on the ability of the project manager to effectively estimate the resources, cost, and schedule required for the project. This paper has discussed the fundamental activities of software project management, including project planning, risk management, people management, reporting, and proposal writing. In particular, the focus has been on project planning, which involves estimating project characteristics and planning project activities based on these estimates. Effective project planning is essential for successful software project management. The paper has also discussed the importance of project size estimation and the different techniques available to estimate project size. It is essential for software project managers to have a systematic knowledge of the different estimation techniques and past knowledge to obtain effective project planning. Finally, the paper emphasized that project plans must be adapted and updated as the project progresses, and variability in software requirements means instability in cost and schedule.

AUTHOR CONTRIBUTIONS

Khin Shin Thant and Hlaing Htake Khaung Tin contributed equally to the conception and design of the study, data collection and analysis, writing of the manuscript, data interpretation, and critically revised the manuscript for important intellectual content. Both authors read and approved the final manuscript.

CONFLICTS OF INTEREST

The authors declare no funding sources or conflicts of interests related to this research paper.

AUTHORS FUNDING

The authors of a research paper have not received any funding related to this research or publication of the paper.

REFERENCES

1. Reifer DJ. Software Management. 6th ed. New Jersey: Wiley; 2002.
2. BarryBoehm B, Chris A. Software Cost Estimation with COCOMO II. New Jersey: Prentice Hall; 2000.
3. Sommerville I. Software Engineering. Global Edition. 10th ed. London: Pearson Education; 2015.
4. Available from: https://www.onlineengineeringprograms.com/
5. Tin HH. Analysis the learning outcomes survey of the software project management. Int J Futur Trends Eng Sci 2019;2:13-17.
6. Pressman RS. Software Engineering, a Practitioner's Approach. 9th ed. R.S. Pressman and Associates, Inc.; 2015.
7. Mall R. Fundamentals of Software Engineering. 4th ed. Delhi: Prentice Hall India Learning Private Limited; 2014.