Journal of Information Systems Engineering and Management
2025, 10(56s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Guideline for Converting Software Size to Cost Estimation from Sequence Diagram Using COSMIC Function Points

Renny Sari Dewi[1], Riyanarto Sarno[2], Endang Siti Astuti[3], Nisriina 'Aisy F. Sari[1], Yogantara Setya Dharmawan[4]
[1]*Department of Digital Business, Universitas Negeri Surabaya, Surabaya – Indonesia*
[2]*Department of Informatics, Institut Teknologi Sepuluh Nopember, Surabaya – Indonesia*
[3]*Department of Administration Sciences, Universitas Brawijaya, Malang – Indonesia*
[4]*Department of Information Systems, Institut Teknologi Sepuluh Nopember, Surabaya – Indonesia*
Corresponding author: rennydewi@unesa.ac.id

ARTICLE INFO

Received: 18 Dec 2024
Revised: 10 Feb 2025
Accepted: 28 Feb 2025

ABSTRACT

This study explores the application of the early COSMIC method—a functional size measurement approach designed for early-stage software development—by leveraging sequence diagrams as primary input artifacts. Unlike COSMIC as comprehensive size functional measurement, early COSMIC only indicates the software pre-size estimation when system specifications are incomplete, making it well-suited for iterative development. The method was implemented in a case study involving the development of pond applications using IoT sensors, which comprised a unified modelling language (UML) sequence diagram. Its functional size was estimated at 45 COSMIC Function Points (CFP), derived from the classification of data movements such as Entry, Read, Write, and Exit. Based on the calculated functional size and labor pay rates (refer to Inkindo billing rates), the projected development cost for the application was IDR 109,361,574 allocated then. These findings affirm that early COSMIC provides a systematic and practical contribution for software project manager in real-world project development budget estimation.

Keywords: software estimation, COSMIC, software cost, sequence diagram.

1. INTRODUCTION

Software cost estimation method is a critical aspect of project planning in software development [1]. Its primary objective is to forecast the financial resources required to complete a project within a predefined timeframe and scope. The accuracy of cost estimation significantly influences project success, as poor estimations may lead to budget overruns or project failure. Consequently, the use of systematic and reliable approaches is essential for conducting software effort estimation, particularly during the early phases of development [2].

One practical method that has gained considerable attention in recent studies is the early application of the COSMIC (Common Software Measurement International Consortium) functional size measurement model [3], [4]. This approach facilitates the quantification of software functional size in the early stages of software engineering, even when only initial design artifacts—such as sequence diagrams—are available. Sequence diagrams are frequently employed during the preliminary design phase to illustrate the interactions among system objects [5]. Accordingly, the early COSMIC approach leverages these artifacts to derive preliminary size estimations without the necessity of complete specification documents.

Despite the promising potential of the early COSMIC approach in estimating functional size at early development stages, very few studies have explored its integration into a comprehensive software cost estimation process. Most existing research limits its scope to functional size measurement, without extending the analysis to translate these measurements into tangible cost estimates [6], [7], [8], [9]. Establishing a direct linkage between functional size and cost models would be highly beneficial for project managers, enabling data-driven decision-making from the outset of the development lifecycle.

This research gap highlights the necessity for developing a methodological framework that bridges early functional size measurement—using the early COSMIC method—and comprehensive software cost estimation [10]. Such an

Journal of Information Systems Engineering and Management
2025, 10(56s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

integration would not only enhance the accuracy of estimations but also enable earlier and more informed planning decisions [11]. Therefore, further investigation is required to assess how early COSMIC can be systematically combined with cost estimation models to construct a holistic and practical framework.

The contributions of this study are both theoretical and practical. Theoretically, the research aims to extend the applicability of early COSMIC by incorporating it into the broader context of cost estimation and by providing a comprehensive framework that spans from size measurement to budget calculation. Practically, the proposed method is expected to assist software project managers in formulating reliable budget estimates from the early phases of development, thereby improving planning efficiency, enhancing estimation accuracy, and mitigating the risk of project failure due to budgetary miscalculations.

2. LITERATURE REVIEW

2.1. Early COSMIC Approach as Data Inputs

The early COSMIC (Common Software Measurement International Consortium) method is an approach designed to measure the functional size of software during the early stages of development, even in the absence of complete system documentation. In contrast to the standard COSMIC method, which typically requires detailed functional specifications, early COSMIC facilitates size estimation based on software requirement specifications (SRS) such as use case diagrams, sequence diagrams, or user requirement scenarios. The core principle of this approach adheres to the fundamental COSMIC measurement rules, which involve counting data movements—Entry, Exit, Read, and Write—that represent interactions (Figure 1) between actors, systems, databases or among components within the system [4], [7].

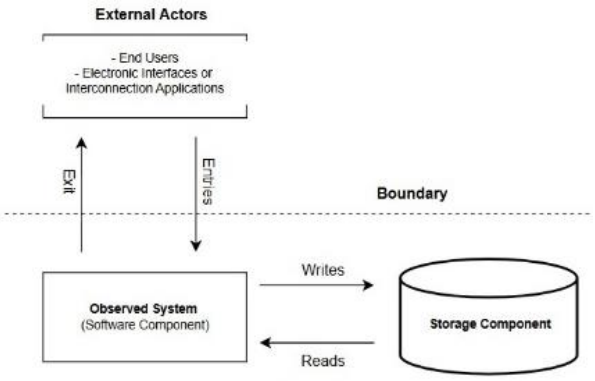

Figure 1. Type of Data Movement in Early COSMIC Approach

Based on Figure 1 early COSMIC adapts four data movements based on elements available refer to design artifacts, such sequence diagrams [12]. For instance, in the context of sequence diagrams, inter-object messages are utilized to identify relevant data flows, thereby enabling measurements to be conducted more efficiently without the need for complete documentation. The outcome of this measurement process is expressed in COSMIC function points, a standardized unit that can be employed for various purposes such as cost estimation, project planning, and cost-based development activities.

Due to its capability to produce raw stage estimations, the early COSMIC method proves particularly valuable in modern software development environments that are iterative and fast-paced, such as those employing agile methodologies [13]. Although this method simplifies certain aspects of the measurement process, it retains validity and consistency by adhering to the formal rules established in the COSMIC standard. Therefore, early COSMIC presents itself as an efficient and reliable alternative for supporting early decision-making throughout the software development lifecycle.

2.2. Resource Allocation-Based Software Cost Estimation

Software cost estimation refers to the process of forecasting the required resources—primarily time, labor, and financial expenditure—for the development of software applications. Various methodologies have been established

to support this task, including the Constructive Cost Model (COCOMO) [14], Function Point Analysis (FPA) [15], [16], and the COSMIC functional sizing method [17]. However, the majority of these approaches rely heavily on the availability of complete functional specification documents and do not directly incorporate visual design artifacts, such as sequence diagrams, as primary input for estimation. For instance, COCOMO depends on metrics such as lines of code (LoC) or user-based functional size, which are typically only obtainable in the later stages of design or even during initial implementation.

In contrast, alternative approaches and recent research efforts have begun to investigate the use of Unified Modeling Language (UML) artifacts—particularly sequence diagrams—for cost estimation in the early stages of development [18]. Sequence diagrams chronologically depict the interactions among system objects and thus can represent the intended functional scenarios envisioned by end-users. By extracting message interactions from these diagrams, some methods attempt to translate this information into functional size metrics, which can then be converted into cost estimations using empirical or statistical modeling techniques.

One of the practical advantages of early-stage cost estimation is its ability to facilitate early resource planning. In particular, labor cost—often the largest component in software development budgets—can be quickly estimated by allocating working hours or days to each identified functional unit, based on pay rates derived from standard industry benchmarks or development activities [19]. Since labor cost can be directly linked to estimated effort per functional unit, it becomes the most straightforward and rapid element to quantify within the overall cost estimation process. This enables development teams and project managers to make informed budgeting decisions even before full technical specifications are available.

The sequence diagram-based approach thus offers significant benefits in terms of early, adaptive, and context-aware estimation, particularly in modern iterative and agile development environments where timely decision-making is critical. While this approach is not yet as widely institutionalized as traditional models such as COCOMO, it shows considerable promise when combined with early COSMIC, both in terms of methodological rigor and practical utility. Nonetheless, the adoption of such methods still requires the development of standardized frameworks and further empirical validation to ensure reliability and consistency across diverse development contexts [20].

In conclusion, integrating sequence diagram analysis with functional sizing and labor-based cost allocation creates a streamlined estimation process that supports early and rational decision-making. This integrated approach enhances estimation flexibility while preserving accuracy, positioning it as a viable alternative for project managers aiming to conduct resource planning from the earliest phases of the software development lifecycle.

2.3. Cost Rates from Salary Guide

The hourly labor wage is derived by dividing the monthly salary by the number of working days in a month. However, according to the 2024 Inkindo billing rate, a regional index variable—based on the province weight—also influences the monthly salary, which is further determined by the worker's educational qualifications. This project is implemented in East Java, where the index, as stipulated by the Inkindo billing rate, is 0.971.

Table 1. Labor Payrates Assumption Based on 2024 Inkindo Billing Rate

Job Position	Qualification	Monthly Salary (IDR)*	Cost/days (IDR)	Payrate/hours (IDR)
Project Manager (PM)	Master's degree, with min. 3-yr experiences	33.014.000	1.500.636	166.737
System Analyst (SA)	Bachelor's degree with min. 3-yr experiences	25.246.000	1.147.545	127.505
Software Tester (ST)	Bachelor's degree with min. 3-yr experiences	25.246.000	1.147.545	127.505
Programmer (PR)	Bachelor's degree with min. 2-yr experiences	14.030.950	637.770	70.863

Source: https://www.inkindo.org/documents/ikdweb-billing-rate.pdf accessed on April 23, 2025

Journal of Information Systems Engineering and Management
2025, 10(56s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

3. METHODS

3.1. Proposed Method

The methodological framework for software cost estimation is based on the early COSMIC approach, utilizing sequence diagrams as the primary data inputs. This process is structured into three major phases: functional size measurement using early COSMIC, determination of labor payrates and resource allocation based on development activities, and computation of the total projected software cost (Figure 2).

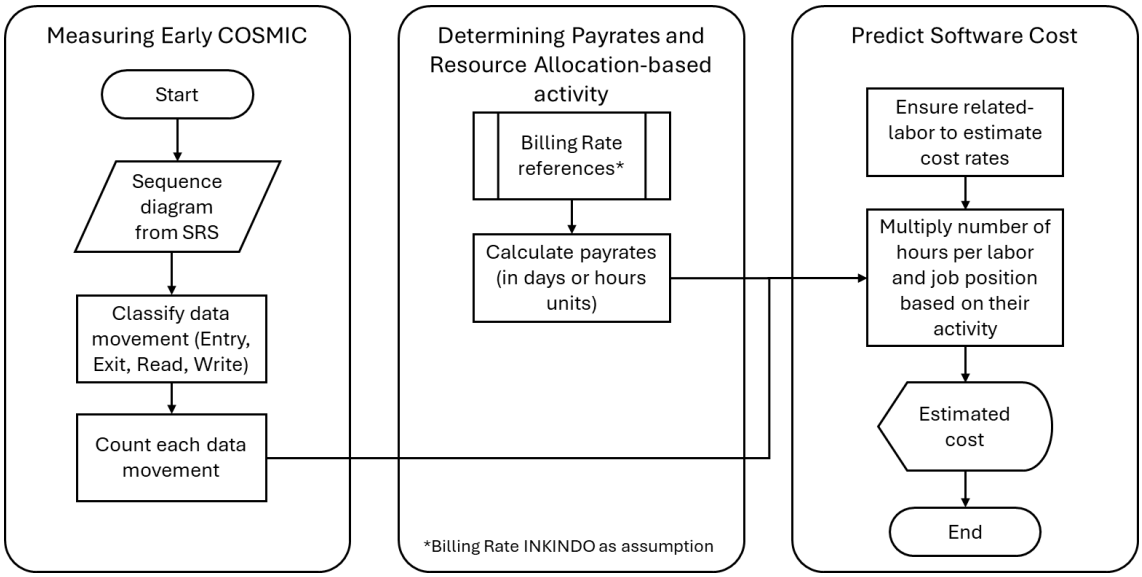

Figure 2. The Proposed Method

According to Figure 2, the initial phase begins with the analysis of a sequence diagram, which depicts the chronological interactions among objects within the software system. From this diagram, the types of data movement—namely Entry (user input), Exit (system output), Read (retrieving data from storage), and Write (storing data to storage)—are identified and classified. Each type of data movement is then counted to derive an initial estimate of the software's functional size.

The second phase focuses on establishing pay rates, which refers to the labor costs that serve as the basis for cost computation. These pay rates may be sourced from industry standards or prevailing market data and are typically expressed in time-based units, such as hourly or daily wages. The pay rates play a critical role, as they are used to determine the monetary cost associated with completing each unit of work, defined in terms of data movement. During this phase, the functional size is also converted into estimated work time, which involves approximating the duration required by development personnel to implement each type of data movement previously identified.

The final phase entails calculating the total software cost estimate. This is achieved by multiplying the number of data movements, the estimated time per data movement unit, and the labor pay rates. This approach enables cost estimation to be performed at the early stages of development, even when system specifications are not yet fully defined, making it especially beneficial in modern, iterative, and agile development environments.

3.2. Case Study to Test the Proposed Method

The sequence diagram of a web-based pond product application that is used to test the proposed method (Figure 3). It is drawn by Claude AI then needs to be adjusted by the software expert as project manager itself. Practically in software development, the sequence diagrams are part of software requirement specification (SRS) document, which is not only sequence diagram but also high-level requirement, domain object, use case diagrams, use case narratives, and raw conceptual data model.

Journal of Information Systems Engineering and Management
2025, 10(56s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

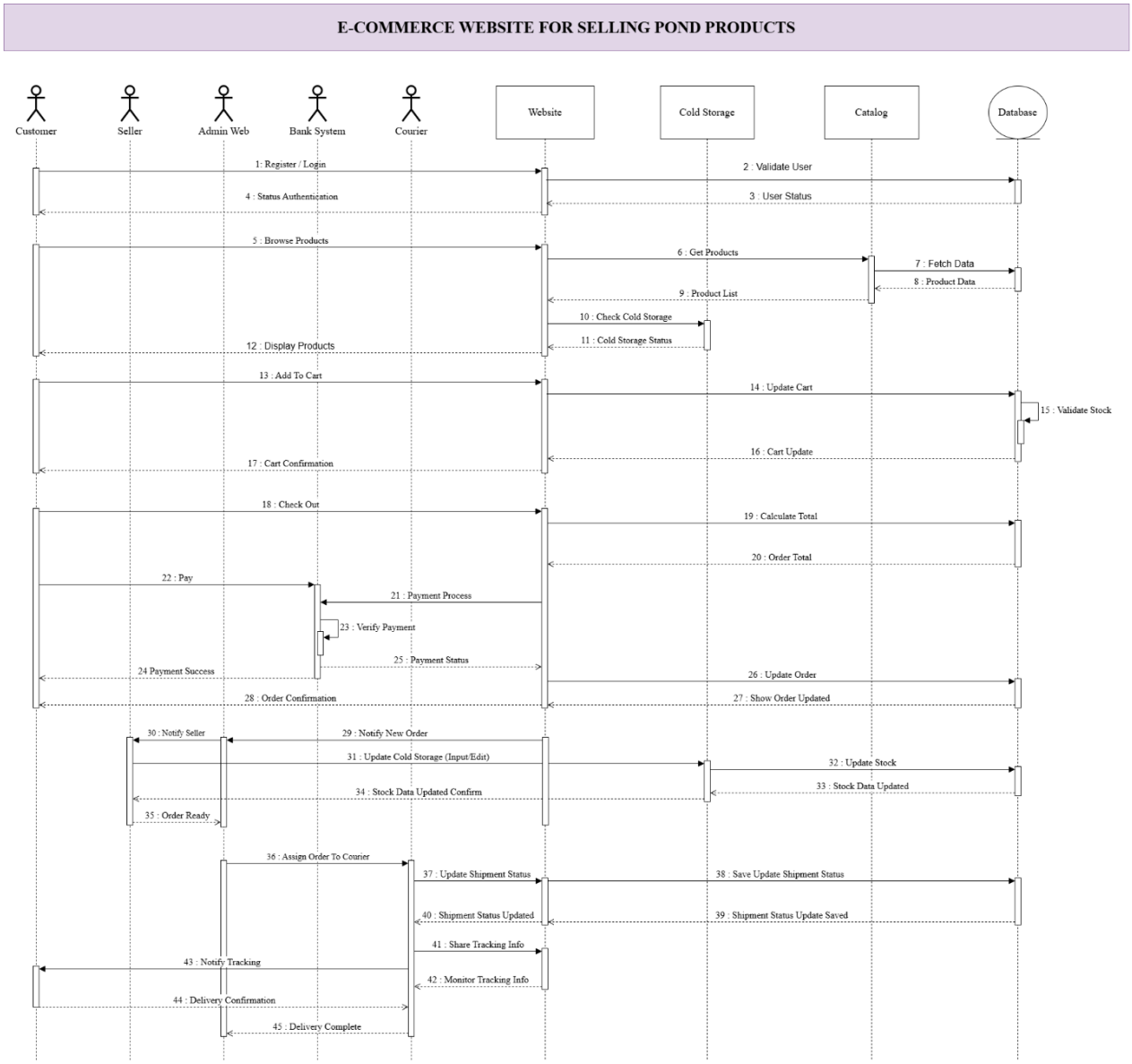

Figure 3. Sequence Diagram Illustration (by Claude AI) as Case Study

Figure 3 presents a detailed sequence diagram modelling the operational workflow of an e-commerce platform designed for selling pond-related products. The diagram captures the dynamic interactions between various actors—namely customer, seller, admin web, bank system, and courier. Also, system components including website, cold storage, catalogue, and database. Each interaction is represented sequentially from top to bottom of flow of activities that comprise the system's business process. There are 45 data movements such as 12 Entries, 7 Writes, 13 Read, and 13 Exits.

The process initiates with the user registration and login sequence, followed by authentication through the website and database. Once authenticated, the customer is able to browse products, which involves the retrieval of product data from the catalogue and cold storage subsystems. After displaying the product listings, the customer proceeds to add selected items to the cart. This action triggers a series of updates involving the cart, stock validation from the database, and total cost calculation.

Subsequent stages involve payment processing facilitated by the bank system, followed by order confirmation and seller notification. The seller updates the cold storage inventory and synchronizes stock data with the central database. Upon order readiness, the seller assigns the order to the courier, who updates shipment status and tracking

Journal of Information Systems Engineering and Management
2025, 10(56s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

information. This sequence concludes with delivery confirmation and completion, ensuring that all relevant stakeholders are updated accordingly. The use of this comprehensive sequence diagram enables a systematic and traceable approach to modelling complex e-commerce operations and facilitates functional size measurement when applying methods such as early COSMIC.

4. RESULT AND DISCUSSION

The result and discussion of this study are represented in this section. Overall, the findings successfully address the primary objective of this study, which is to estimate software development costs based solely on-pre-stage analysis utilizing sequence diagrams as the main data inputs.

4.1. Measuring Early COSMIC

To measure early COSMIC, firstly, ensure four types of data movements are mapped carefully. Each type must relate to the job position which labors are in charge. Then the labor shall connect to development activities where they involved to its job consequences. This relationship can be seen on Table 2.

Table 2. Relationship Between Early COSMIC, Job Position, and Development Activities

Data Movement	Relate to Job Position* (weight %)	Development Activities**
Entry	PR (100%)	Coding
Write	SA (30%) and PR (70%)	Design
Read	ST (40%) and SA (60%)	Unit Testing and Integration
Exit	ST (50%) and PM (50%)	Deployment and Acceptance

*Abbreviations: PR=Programmer, SA=System Analyst, ST=Software Tester, PM=Project Manager
**Refer to prior study [21]

Table 2 shows that each type of data movement has one or more job positions even though they do the same activity. Because of that, the expert assumed that each job position has the weight of role, and the total is 100%. This weight is important to move to the next stage.

This study uses several assumptions in Table 2. Each data movement category within the Early COSMIC functional size measurement framework is not merely a technical construct but is intricately tied to distinct roles and responsibilities within a software development team. This mapping between data movement types and human resource functions provides a critical foundation for interpreting the cost distribution shown in Table 4.

Starting with Entry, which is fully assigned to the Programmer role (PR: 100%) and associated with the activity of *coding*, we see a direct correlation with high developer effort. Entry functions typically involve the development of front-end input mechanisms and server-side validation logic, which are core programming tasks. Given that programmers are responsible for both functional implementation and handling user input securely and efficiently, the assignment of a 27-hour effort per Entry unit in Table 4 becomes justifiable. This labor-intensive work not only involves scripting but also client-server interactions, exception handling, and integration into broader system workflows—justifying both the time requirement and the programmer-specific rate used in cost computation.

Write operations are distributed between the System Analyst (SA: 30%) and Programmer (PR: 70%) roles, and are associated with the *design* phase. This hybrid allocation reflects the dual nature of Write functions, which combine logical data structure formulation (handled by analysts) with physical implementation (handled by programmers). The elevated hourly effort of 36 hours per Write unit in Table 4 is thus explained by the need for upfront architectural planning (by analysts), followed by extensive back-end logic development (by programmers). Moreover, writing data to persistent storage demands transaction management, consistency enforcement, and error recovery—all of which add complexity and necessitate senior-level expertise. Hence, both the resource mix and the high cost per unit are logically aligned with the real-world development scenario.

In the case of Read, the job role distribution is 60% System Analyst (SA) and 40% Software Tester (ST), with the primary development activity being *unit testing and integration*. This reflects the need for Read functions to be designed with an understanding of the database schema and application flow (by analysts), while also being subjected

Journal of Information Systems Engineering and Management
2025, 10(56s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

to rigorous verification to ensure data correctness, user access privileges, and query performance (by testers). The estimated 18 hours per Read unit in Table 4 is justified not only by the design of data retrieval mechanisms but also by the necessary quality assurance efforts prior to production release.

Exit operations are mapped equally to Software Testers (ST: 50%) and Project Managers (PM: 50%) and are associated with *deployment and acceptance* activities. Exit functions typically involve producing system outputs—such as reports, exports, or data transmissions to third-party systems—often in standardized formats. The role of the Project Manager is prominent here due to the need to coordinate deployment procedures, interface with stakeholders, and manage compliance or documentation requirements. Meanwhile, testers validate output correctness and ensure it meets acceptance criteria. As such, the 18-hour effort per Exit unit in Table 4 is appropriately reflective of the coordination and validation workload shared by PMs and STs.

4.2. Determining Cost Rates and Resource Allocation-based Activity

As described in Section 2.3, labor payrates were assumed from monthly salary (by 2024 Inkindo Billing Rate considered to East Java's wage index) then divided into 22 days and 9 hours per day. During the activity's running, one or more job positions need to be adjusted based on their percentages' activity weight (Table 3).

Table 3. Labor Payrate Adjustment Based on Development Activities

Development Activities	Job Position in charge	Adjusted Payrate/hour (IDR)
Coding	PR (100%)	70,863
Design	SA (30%) and PR (70%)	87,856
Unit Testing and Integration	ST (40%) and SA (60%)	127,505
Deployment and Acceptance	ST (50%) and PM (50%)	147,120

Table 3 above describes the adjusted payrate depends on job position's percentage. For example, "Deployment and Acceptance" activity mix two job position such as Software Tester or ST (50%) equals to IDR 63,753 and Project Manager or PM (50%) equals to IDR 83,368. So, the adjusted payrate for this activity is IDR 147,120 per hour.

4.3. Calculating Software Cost Estimation

After payrate being adjusted by percentage of job position, projected cost is gathered from multiplying number of data movement, cost rate per hour, and total hours of work duration. For example, "Exit" has 13 CFP then multiply with adjusted cost rate per hour is IDR 147,120. For another data movement calculation is shown in Table 4.

Table 4. Software Development Cost Estimation

Data Movement	Early COSMIC	Adjusted Cost Rate/hour (IDR)	Work Duration (hours)	Cost Estimated (IDR)
Entry	12	70,863	27	22,959,612
Write	7	87,856	36	22,139,712
Read	13	127,505	18	29,836,170
Exit	13	147,120	18	34,426,080
Total				109,361,574

Based on Table 4, the cost estimation for the development of a web-based point-of-sale application is carried out using the *Early COSMIC* method, which focuses on measuring the functional size of software through four categories of data movements: Entry, Write, Read, and Exit. In this approach, each unit of data movement is assumed to require a fixed implementation effort, and the total development time is calculated by multiplying the number of *COSMIC Function Points* (CFP) by the estimated effort per unit. Specifically, if the estimated effort for an Entry is 27 hours, this means that the development of each unit Entry function—comprising analysis, design, implementation, testing,

Journal of Information Systems Engineering and Management
2025, 10(56s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

and integration—requires 27 working hours. Therefore, with 12 Entry CFPs, the total time required amounts to 12 × 27 = 324 hours. Similarly, the Write operations are estimated at 7 CFP × 36 hours = 252 hours; Read and Exit each at 13 CFP × 18 hours = 234 hours.

These differences in effort per data movement unit are not arbitrary. Rather, they reflect the inherent technical complexity and implementation challenges associated with each type of function. The "Entry" operations, while seemingly straightforward as user input mechanisms, are in practice among the most complex to implement. They involve not only designing user-friendly input forms but also developing robust client- and server-side validation logic, handling erroneous input scenarios, implementing dynamic form behavior (e.g., dependent dropdowns, autocomplete), and ensuring secure and efficient communication with backend services. Additionally, Entry functions require extensive testing to verify system reliability against a broad range of user interactions. These characteristics justify the relatively high estimated effort of 27 hours per Entry function.

"Write" operations are even more complex, with an estimated duration of 36 hours per unit, the highest among all data movements. Writing data to the system entails more than executing a database insert or update; it includes ensuring data integrity through proper relational schema design, enforcing constraints, managing transactional behavior (ACID compliance), and implementing rollback mechanisms in the event of failure. Furthermore, "Write" functions often require implementing concurrency control, audit logging, and input sanitization to prevent vulnerabilities such as SQL injection. The critical impact of "Write" operations on the persistent state of the system necessitates rigorous development and testing processes, thus warranting the highest time allocation.

In contrast, "Read" operations are assigned a lower per-unit duration of 18 hours, as they typically involve retrieving and displaying data from the database to the user interface. However, this task still requires the design of efficient database queries, the implementation of filtering, sorting, pagination mechanisms, and compliance with user-specific access control policies. Even though "Read" is functionally less invasive than "Write", ensuring performance, data accuracy, and security still demands a moderate development effort.

Similarly, "Exit" operations are estimated at 18 hours per unit. "Exit" refers to data moving out of the system, such as generating reports, exporting files, or transmitting data to external systems. These operations require formatting and structuring the output data—often in specific formats such as PDF, Excel, or JSON—as well as potentially managing integrations with external APIs. When exporting sensitive data, "Exit" functions may also include digital signatures, encryption, or additional compliance mechanisms. The formatting, validation, and testing of the output, along with the potential for integration challenges, contribute to the comparable development effort to that of Read functions.

In summary, the variation in per-unit time estimates for each type of data movement reflects the nuanced realities of software development effort. The values—27 hours for Entry, 36 hours for Write, and 18 hours each for Read and Exit—are consistent with a comprehensive effort estimation methodology that accounts for functional complexity, technical risk, validation requirements, and integration needs. These estimations are aligned with the principles outlined in the ISO/IEC 19761 standard for functional size measurement and supported by leading software engineering literature such as Abran & Nguyen (2009), Pressman (2010), and Sommerville (2011), all of which underscore the importance of correlating functional size with context-specific implementation effort to derive accurate and realistic software project estimates.

In addition to the variation in effort duration for each type of data movement, the application of differentiated hourly rates for Entry, Write, Read, and Exit functions further refines the cost estimation model. This differentiation in cost per hour for each category is based on the underlying assumption that not all development tasks require the same level of technical expertise, cognitive load, or implementation risk. Consequently, assigning a uniform rate across all data movement types would fail to accurately reflect the real-world distribution of labor intensity and skill specialization in software development.

From an economic and engineering standpoint, higher hourly rates for Write and Entry functions are justifiable due to their elevated technical complexity and the potential system-wide impact of errors in their implementation. Write operations, in particular, are associated with data persistence, business rule enforcement, and database integrity, all of which require advanced knowledge in backend architecture, transaction management, and secure coding practices. Developers working on Write functions must anticipate concurrency issues, ensure atomicity and consistency, and

Journal of Information Systems Engineering and Management
2025, 10(56s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

often implement rollback and recovery mechanisms. These tasks typically demand more senior-level expertise and therefore command a higher billable rate.

Similarly, Entry operations may appear to involve only front-end interface design, but they often require deep understanding of both front-end frameworks and backend validation logic. High-quality Entry design affects not only usability but also security, as poorly designed input handling is a common vector for injection attacks and data corruption. Implementing user-centric validation, responsive design, internationalization support, and real-time feedback mechanisms calls for a multidisciplinary skill set that spans UX design, client-side scripting, and server-side logic. As such, the rate applied to Entry functions reflects the integration of diverse and often high-cost skill areas.

In contrast, Read and Exit functions, while still requiring rigorous implementation, are generally more predictable and automatable in nature. Read operations usually consist of composing database queries, applying business filters, and rendering data views. Although they may involve performance optimization and access control enforcement, these tasks are more routine and less error-prone compared to Write and Entry operations. Exit functions, particularly those involving report generation or data export, can often leverage existing libraries or reporting tools, which reduces development time and the need for bespoke solutions. Therefore, these functions may be assigned a relatively lower hourly rate due to their lower risk and the reduced need for advanced problem-solving skills.

Applying differentiated rates in this manner aligns with cost estimation practices in professional software engineering and project management. According to the *Constructive Cost Model* (COCOMO) and activity-based costing models, different software components contribute unevenly to overall project effort and risk, and should therefore be priced accordingly. Moreover, this rate-based differentiation supports a more granular budgeting process and enables project managers to allocate resources in a manner that optimizes both cost-efficiency and project quality.

In conclusion, the use of distinct hourly rates for Entry, Write, Read, and Exit data movements is not only logical but necessary to ensure accuracy in effort-based cost estimation. This approach reflects the real-world disparity in technical difficulty, required expertise, and associated implementation risks for each function type. When combined with functional size measurement such as COSMIC, this cost model enables a more precise and context-sensitive estimation framework—one that adheres to best practices in software engineering economics and enhances the reliability of project planning and budgeting outcomes.

Overall to build a web-based point-of-sale pond harvest application should invest about IDR 109,361,574 based on early COSMIC approach.

5. CONCLUSION

This research concludes that the proposed software cost estimation approach effectively quantifies the development effort required for a web-based pond product sales application. By conducting early COSMIC functional size measurement derived from the sequence diagram and incorporating regionally adjusted hourly labor payrates, the total estimated development cost amounts to IDR 109,361,574. The cost components are distributed across different types of data movements, with the Exit and Read categories contributing the highest cost due to their relatively high adjusted hourly rates. These findings demonstrate the applicability and practicality of early COSMIC in enabling early-stage cost estimation, especially when detailed specifications are not yet available. Furthermore, the results validate the integration of functional sizing and cost modeling as a reliable methodology for early project budgeting in iterative software development environments.

While this study demonstrates the feasibility of using early COSMIC in conjunction with sequence diagrams for early-stage cost estimation, its implementation has thus far been validated on a single web-based application for aquaculture product sales. This limited scope presents an opportunity for further empirical validation across a broader spectrum of software domains and architectures. Future research should explore the applicability and scalability of this method on mobile, desktop, and embedded systems, as well as on projects with varying levels of complexity and stakeholder requirements. In addition, a comparative study involving other cost estimation techniques—such as traditional COCOMO or Function Point Analysis—may help evaluate the relative accuracy and efficiency of the early COSMIC-based approach. Expanding the dataset with multiple case studies would also enable

the use of more advanced statistical modeling, potentially leading to the development of predictive cost estimation tools with higher generalizability and practical relevance in industrial settings.

REFRENCES

[1] R. S. Dewi, G. F. Prassida, Sholiq, and A. P. Subriadi, "UCPabc as an integration model for software cost estimation," in *Proceeding - 2016 2nd International Conference on Science in Information Technology, ICSITech 2016: Information Science for Green Society and Environment*, 2017. doi: 10.1109/ICSITech.2016.7852631.

[2] P. Brar and D. Nandal, "A Systematic Literature Review of Machine Learning Techniques for Software Effort Estimation Models," in *Proceedings - 2022 5th International Conference on Computational Intelligence and Communication Technologies, CCICT 2022*, 2022, pp. 494–499. doi: 10.1109/CCiCT56684.2022.00093.

[3] COSMIC, "COSMIC Measurement Manual for ISO 19761 Part 1 : Principles, Definitions & Rules," in *COSMIC Measurement Manual for ISO 19761*, 5.0., 2020, ch. 1, pp. 1–17.

[4] COSMIC, "COSMIC Measurement Manual for ISO 19761 Part 2 : Guidelines," in *COSMIC Measurement Manual for ISO 19761*, 5.0., 2020, ch. 2, pp. 1–17. [Online]. Available: https://cosmic-sizing.org/publications/measurement-manual-v5-0-part-1-principles-definitions-rules

[5] T. M. Fehlmann and E. Kranich, "COSMIC Functional Sizing based on UML Sequence Diagrams," in *MetriKon 2011*, 2011, p. 16.

[6] R. S. Dewi *et al.*, "Involving Data Complexity in Software Size Estimation Based on Conceptual Data Model (Case Study: Catering Management Information System)," in *ICECOS 2024 - 4th International Conference on Electrical Engineering and Computer Science, Proceeding*, Institute of Electrical and Electronics Engineers Inc., 2024, pp. 376–379. doi: 10.1109/ICECOS63900.2024.10791219.

[7] R. S. Dewi *et al.*, "Improving Software Size Estimation Using Data Complexity (Case Study: Research and Community Service Monitoring Apps)," in *International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, Institute of Electrical and Electronics Engineers Inc., 2024, pp. 315–319. doi: 10.1109/EECSI63442.2024.10776530.

[8] R. S. Dewi *et al.*, "Calculating Software Size Based on Conceptual Data Model Using Data Complexities of Fishermen Information System," in *ICECOS 2024 - 4th International Conference on Electrical Engineering and Computer Science, Proceeding*, Institute of Electrical and Electronics Engineers Inc., 2024, pp. 304–307. doi: 10.1109/ICECOS63900.2024.10791268.

[9] R. S. Dewi *et al.*, "Software Size Measurement Using Data Complexities (Case Study: Marketing Kit Monitoring System)," in *International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, Institute of Electrical and Electronics Engineers Inc., 2024, pp. 331–335. doi: 10.1109/EECSI63442.2024.10776081.

[10] G. Kumar, P. K. Bhatia, and others, "A detailed analysis of software cost estimation using COSMIC-FFP," *PAK Publishing Group J. Rev Comput. Eng. Res*, vol. 2, no. 2, pp. 39–46, 2015.

[11] N. Mittas, E. Papatheocharous, L. Angelis, and A. S. Andreou, "Integrating non-parametric models with linear components for producing software cost estimations," *Journal of Systems and Software*, vol. 99, pp. 120–134, 2015, doi: 10.1016/j.jss.2014.09.025.

[12] R. S. Dewi and R. Sarno, "Software effort estimation using early COSMIC to substitute use case weight," in *Proceedings - 2020 International Seminar on Application for Technology of Information and Communication: IT Challenges for Sustainability, Scalability, and Security in the Age of Digital Disruption, iSemantic 2020*, 2020. doi: 10.1109/iSemantic50169.2020.9234227.

[13] T. Hacaloglu and O. Demirors, "Measureability of Functional Size in Agile Software Projects: Multiple Case Studies with COSMIC FSM," *Proceedings - 45th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2019*, pp. 204–211, 2019, doi: 10.1109/SEAA.2019.00041.

[14] S. Sholiq, R. Sarno, E. S. Astuti, and M. A. Yaqin, "Implementation of COSMIC Function Points (CFP) as Primary Input to COCOMO II: Study of Conversion to Line of Code Using Regression and Support Vector Regression Models," *International Journal of Intelligent Engineering and Systems*, vol. 16, no. 5, pp. 92–103, 2023, doi: 10.22266/ijies2023.1031.09.

[15] A. Yhurinda, P. Putri, and A. P. Subriadi, "Software Cost Estimation Using Function Point Analysis," *IPTEK Journal of Proceedings Series*, vol. 0, no. 1, pp. 79–83, 2019, doi: 10.12962/j23546026.y2019i1.5115.

[16] R. S. Dewi, A. P. Subriadi, and Sholiq, "A Modification Complexity Factor in Function Points Method for Software Cost Estimation Towards Public Service Application," *Procedia Comput Sci*, vol. 124, pp. 415–422, 2017, doi: 10.1016/j.procs.2017.12.172.

[17] G. De Vito, F. Ferrucci, and C. Gravino, "Design and automation of a COSMIC measurement procedure based on UML models," *Softw Syst Model*, vol. 19, no. 1, pp. 171–198, Jan. 2020, doi: 10.1007/s10270-019-00731-2.

[18] L. Lavazza and S. Morasca, "An Empirical Evaluation of Two COSMIC Early Estimation Methods," in *Proceedings - 26th International Workshop on Software Measurement, IWSM 2016 and the 11th International Conference on Software Process and Product Measurement, Mensura 2016*, IEEE, 2016, pp. 65–74. doi: 10.1109/IWSM-Mensura.2016.020.

[19] K. Saleh, "Guidelines for effort and cost allocation in medium to large software development projects," *International Conference on Applied Computer Science - Proceedings*, no. October, pp. 33–34, 2010.

[20] T. Rique, E. Dantas, M. Perkusich, K. Gorgonio, H. Almeida, and A. Perkusich, "Empirically Derived Use Cases for Software Analytics," in *2022 30th International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2022*, 2022. doi: 10.23919/SoftCOM55329.2022.9911514.

[21] R. S. Dewi and Y. S. Dharmawan, "A Proposed Model for Embedding Risk Proportion in Software Development Effort Estimation," *Procedia Comput Sci*, vol. 234, pp. 1777–1784, 2024, doi: https://doi.org/10.1016/j.procs.2024.03.185.