Cost-Benefit Analysis for Modernizing a Large-Scale Industrial System

Kazuki Yokoi*, Eunjong Choi†, Norihiro Yoshida‡, Joji Okada* and Yoshiki Higo§
*NTT DATA Group Corporation, Japan, {Kazuki11.Yokoi, Joji.Okada}@nttdata.com
†Kyoto Institute of Technology, Japan, echoi@kit.ac.jp
‡Ritsumeikan University, Japan, norihiro@fc.ritsumei.ac.jp
§Osaka University, Japan, higo@ist.osaka-u.ac.jp

***Abstract*—Legacy systems pose significant challenges to companies. Software modernization approaches have been proposed to address this issue. However, a lack of standardization and reliance on ad hoc processes often lead to software modernization failures. Incremental modernization, a strategy that improves software systems in a step-by-step manner rather than attempting to simultaneously overhaul the entire system, aims to mitigate the risk of failure. However, this approach can increase costs owing to the complexity of integrating legacy and modernized products. In this paper, we present a case study that employs a cost-benefit estimation analysis in a large-scale industrial project that underwent incremental modernization in the past. We compare the actual and estimated cost-benefit values in the context of incremental modernization. As a result, we confirmed that the cost estimates were valid, but we could not judge whether the benefit estimates were valid.**

***Index Terms*—software maintenance, dependency analysis, cost estimation**

I. Introduction

Various software modernization approaches have been proposed to address critical challenges posed by legacy systems [1]. Many companies demonstrate a strong demand for software modernization, motivated by the necessity of rejuvenating legacy systems for agile responses to business changes and cost-effective maintenance. Frequently, the systems targeted for modernization are critical, allowing for minimal tolerance for failure. However, the lack of standardization and reliance on ad-hoc processes often leads to failures in software modernization. Incremental modernization, which is an approach that improves software systems in a step-by-step manner rather than attempting to overhaul the entire system simultaneously, aims to mitigate the risks of failure [2], [3].

Software modernization approaches can increase costs because of the complexity of integrating legacy and modernized projects [2]. Therefore, estimating the cost-benefit at the planning phase of modernization is crucial, especially in industry. However, it is challenging to estimate the cost-benefit of incremental modernization in the planning phase. For example, the lack of design documents hinders the accurate estimation of development costs and benefits. Furthermore, it is difficult to predict the main risk factors for incremental modernization in the planning phase; as a result, the actual costs may exceed the estimated costs. The first and fourth authors of this study frequently encountered such scenarios when they engaged in the modernization of real software projects in their cooperation.

Although several studies have been conducted to estimate the cost-benefit of refactoring [4]–[8], to the best of our knowledge, the cost-benefit estimation for incremental modernization has not yet been explored. To bridge this gap, we present a case study that conducts a cost-benefit estimation analysis on a large-scale industrial system, specifically focusing on incremental modernization. The goal of this study is to present a case study that estimates the cost-benefit of incremental modernization in a large-scale industrial system. To achieve this goal, in the case study, we apply the cost-benefit estimation approach, which employs dependency analysis, to a core system within a financial institution. Then, we compare the actual and estimated cost-benefit values in the context of incremental modernization. Consequently, we confirm the effectiveness of the cost-benefit estimation approach for estimating the cost-benefit associated with incremental modernization.

The main contributions of this study are as follows:

- We have preliminary estimated cost-effectiveness, which is essential in system modernization for a large-scale financial system.
- We developed a model for calculating a cost-benefit for incremental modernization for stable system modernization.
- We have analyzed the difference between the pre-estimated cost-benefit and the actual cost-benefit and discussed why the difference occurred.

The remainder of this paper is organized as follows. Section II presents the background of this study. Section III describes a system that underwent incremental modernization, which is the focus of this study. Section IV explains the estimation approach used in this study. Section V presents the analysis. Section VII introduces the related work. Finally, Section VIII concludes the paper and discusses future challenges.

II. Background

A. Modernization

Several modernization approaches have been proposed over the years, including the “big bang” approach [1] and incremental modernization [3]. Figure 1 shows the overview of “big bang” and incremental modernization. The left side of

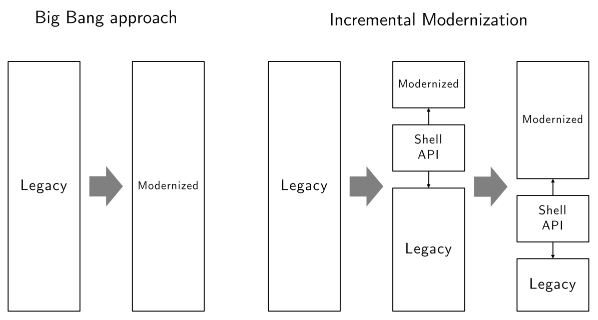


Fig. 1. "big bang" approach and incremental modernization

Figure 1 shows the "big bang" approach, which simultaneously reconstructs the entire system. However, this approach carries a significant risk of migration failure. To mitigate this risk, server approaches including incremental modernization [3] and the "Chicken-Little" strategy [2] have been proposed. These approaches involve dividing the system improvement or migration process into smaller and more manageable components to reduce the risk. The right side of Figure 1 illustrates the core idea of the "Chicken-Little" strategy to construct a composite system consisting of both legacy and modernized products. Communication between these legacy and modernized products occurs through a gateway. In this strategy, a segment of the legacy product is initially replaced by a modernized product, and the gateway directs requests to the modernized product. Over time, the proportion of the legacy product being replaced by the modernized product gradually increases. This process leads to a step-by-step expansion of the modernized product.

While the "Chicken-Little" strategy mitigates the risk of migration failure, it introduces increased risks due to the complexity of integrating legacy and modernized projects. These risks include:

1) Complex communication control between the legacy and modernized products.
2) Ensuring data consistency between the legacy and modernized products becomes challenging.

To mitigate these risks, the integration of legacy and modernized products requires a **shell API** [9]. Shell API refers to codes or scripts that enable interoperability between different software components or systems [9]. This enables the collaboration between independently developed components or systems. In the context of incremental modernization, integrating different technology stacks or platforms between legacy and modernized products requires communication control and data consistency management through a shell API. Consequently, incremental modernization incurs additional costs for developing shell API, resulting in an overall increase in the development costs for the entire system.

B. Cost Estimation

Cost estimation is an empirical process used to estimate the effort and time costs of any software project before its development [10]. In software development, these estimated costs are often inadequate, causing a discernible gap between the estimated and actual effort. This gap tends to widen, particularly during the planning phase of software development, owing to many ambiguous elements (e.g., requirements and design). However, cost estimation during the planning phase is crucial for many companies because it facilitates budget allocation and overall planning. Therefore, it is necessary to recognize inherent uncertainty embedded in the planning phase of cost estimation, communicate this to all stakeholders, and re-estimate the associated costs.

Several cost estimation approaches have been proposed, including analogy-based, bottom-up, and parametric. An analogy-based estimation was derived through comparison with similar past projects. Although this approach is straightforward and suitable for early-stage system development estimates, its application becomes challenging when the background, constraints, and features of past project performances are unclear.

Bottom-up estimation involves identifying the components of a target project, aggregating the effort estimates for each component, and calculating the estimated value for the entire project. While this approach tends to be more accurate, it requires breaking down a target project into smaller, more manageable tasks using a structure such as a Work Breakdown Structure (WBS) and pre-identifying components such as subsystems and components as part of the deliverables. Therefore, adopting this method in the initial stages of a project is challenging because of inherent ambiguity.

The parametric approach involves representing factors such as effort as the dependent variable and attributes such as size and factors as independent variables through mathematical functions. This approach primarily employs an equation effort in which effort and size exhibit direct proportionality ($E = \alpha \times Size$) or size is proportional through a power function ($E = \alpha \times Size^{\beta}$). The latter, in which the effort is proportional to the exponent of size, was also employed in the COCOMO method [11]. In this study, we adopted this approach for cost estimation.

III. Target Project

In this section, we introduce the target project for incremental modernization in this case study. The target project was the core project of a financial institution in Japan, and this system has undergone incremental modernization. The project was a COBOL core system, and the corporation where the first and fourth authors of this paper are currently employed has been responsible for maintaining and developing the project for many years. A summary of the system is presented in Table I. It was a substantial legacy system with inherent issues of low maintainability prior to modernization. The primary objective

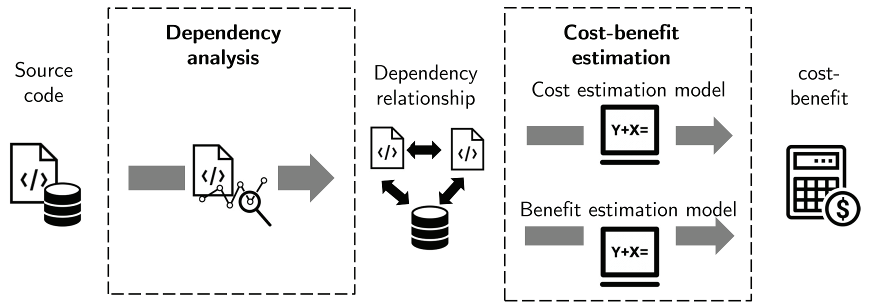


Fig. 2. Estimation approach overview

TABLE I
SUMMARY OF TARGET SYSTEM

System Overview	Core System of financial institutions
System Type	Batch and Online System
Language	Open COBOL
Size	About 4.6 MLOC (Only COBOL)

of the incremental modernization of the target system was to enhance its maintainability.

Through the first incremental modernization, approximately 870 KLOC were extracted from the target system and restructured into Java programs. Unfortunately, the complexity and potential risks associated with incremental modernization were insufficiently considered during the planning phase of modernization. Consequently, the actual size of the target system exceeded the estimated size by 27.5%.

Although the primary objective of incremental modernization was to improve maintainability, a comparison of the development efforts before and after modernization revealed an uncertain enhancement in maintainability. Moreover, owing to the absence of defined metrics to measure this improvement, it remains unclear whether the project yielded any benefits after the modernization.

This experience indicates that the corporation, involving the first and fourth authors of this paper, encountered challenges in estimating the cost-benefit of incremental modernization during the planning phase. Moreover, apart from the project discussed here, the corporation is involved in several other projects in which incremental modernization is being considered. Therefore, this issue is very relevant for the corporation. However, even though several existing studies have estimated the cost-benefit of refactoring [4]–[8], cost-benefit estimates for incremental modernization have yet to be explored, to the best of our knowledge. To bridge this gap, we propose the following Research Questions (RQs):

RQ1 Is it possible to estimate more valid costs during the planning phase of incremental modernization?

RQ2 Is it possible to estimate more valid benefits during the planning phase of incremental modernization?

This study aims to present a case study estimating the cost-benefit of incremental modernization in large-scale industrial systems. To answer the above-mentioned RQs, we present an approach for estimating the cost-benefit in the planning phase of incremental modernization and assess the validity of the estimation approach by applying it to the target project.

IV. Estimation Approach

In this section, we introduce an approach for estimating the cost-benefits of incremental modernization. Figure 2 presents an overview of this estimation approach. As can be seen in the figure, the approach employs a dependency graph constructed through dependency analysis and then estimates the cost-benefit of incremental modernization based on this graph. In the following sections, we describe the dependency graph, followed by explanations of cost and benefit estimations.

A. Dependency Graph

First, we discuss the dependency graph used in this approach. A dependency graph is a graph structure that represents the relationships among software components. In this graph, the nodes depict code entities (e.g., methods or functions) and data entities (e.g., database tables or global variables). The relationships among these nodes are illustrated as directed edges. These edges represent various dependencies between the software components, including call dependencies, where one method or function calls another, or data dependencies, where a piece of code relies on a specific data entity. These dependencies are categorized as “relation types” [12]. These edges include various types of dependencies, such as call dependencies and data dependencies, which are referred to as ”relation types.” Figure 3 shows an example of a dependency graph. In this figure, the nodes represent functions and database tables, and the edges indicate call and data dependencies.

B. Cost Estimation

The cost estimation calculates the person-hours for incremental modernization. This involves determining the estimated work hours by multiplying the calculated person-hours by the monthly rate per person, resulting in a monetary value.

In the cost estimation, the size of the development for incremental modernization was calculated. In the case of modernization of the entire system, the effort is calculated

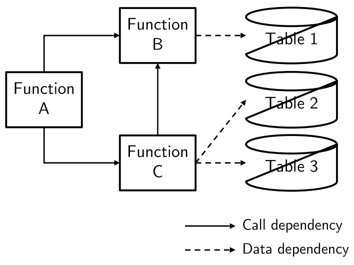


Fig. 3. Example of dependency graph

by multiplying the size of the entire system by productivity. During incremental modernization, part of the system was extracted and rebuilt. However, in some cases, by simply multiplying the size of the extracted program by productivity, the estimated effort may be less than the actual effort. This is because the estimate of the shell API effort was missing. Therefore, we propose an approach for estimating the development size of shell API to obtain the shell API development effort. Dependency graphs are used to estimate the development size of the shell API.

1) Procedure for Cost Estimation: The development effort E for the shell API can be determined as follows:

$$E = \alpha \times \sum_{d \in D} N_d L_d \tag{1}$$

Here, N_d denotes the number of dependencies d spanning the legacy and modernized products, L_d represents the shell API implementation line number per dependency d, and α signifies the productivity (person-months/size). These are summed up for each type of dependency d.

When a modernized product is extracted from a legacy product, the dependencies between the legacy and modernized products are disconnected. For example, dependencies connected by function calls before extraction change to communication via the network after extraction. Therefore, the development of components to convert communication systems requires considerable time and effort.

The number of lines of the shell API implemented for each type of dependency depends on the architecture of the legacy and modernized products. Therefore, estimates of these values are determined by analogy from past development results or by developing a shell API as a proof of concept and collecting data.

2) Example of Cost Estimate Calculation: As illustrated in Figure 4, if only function B is decomposed from the others, communication between functions A and B, functions B and C, and function B and table 1 occurs via the shell API. Therefore, for incremental modernization, the development of a new shell API is necessary. From Table II, the number of dependencies via the shell API is 20 for Call Dependency and 10 for Data Dependency. Thus, the size to implement one Call Dependency with the shell API is 40 LOC, and the size to implement one Data dependency with the shell API is 100 LOC, which are tentative values. When multiplied by the inverse of productivity shown in Table II, the shell API effort is estimated to be 2.10 person-months.

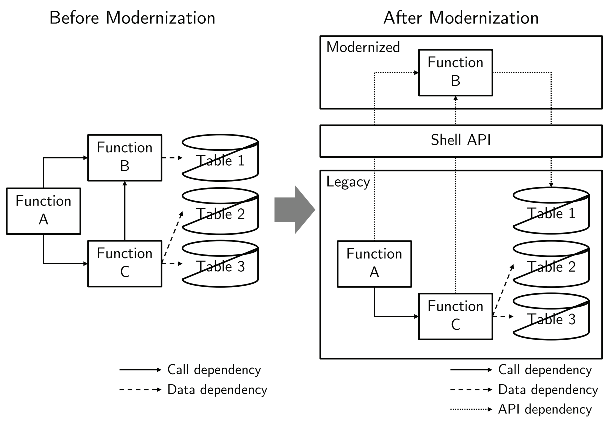


Fig. 4. Examples of functions and tables used for cost estimation

TABLE II
EXAMPLE OF EFFORT ESTIMATION FOR EACH DEPENDENCY

Dependency Type	Number of Dependency	LOC per Dependency	Productivity	Effort
Call	20	40	855	0.93
Data	10	100	855	1.17
		Sum of Effort		**2.10**

Note1: LOC per dependency and productivity are tentative value.
Note2: Productivity is measured in LOC/person-month, and effort is measured in person-month.

C. Benefit Estimation

Modernization has several benefits. In this study, however, we estimated the benefit of the reduction in maintenance effort. A reduction in maintenance effort results in shorter maintenance periods and lower maintenance costs. As a result, the maintenance period per project is shortened, and development agility is improved. Furthermore, reducing maintenance costs will promote investment in other areas, emphasizing the benefits of modernization.

In this study, we estimated the benefits of a reduction in maintenance effort per project after incremental modernization.

Previous research has shown that software maintenance requires significant effort to understand and test [13]–[15], and it is evident that this effort is significantly influenced by change impact [16], [17]. Therefore, the reduction rate of maintenance effort can be affected by the reduction rate of the change impact.

Impact scope refers to the range within which changes to a part of a program can affect other programs. A broader change impact means a more comprehensive range to understand and test, increasing the maintenance effort. By decomposing systems through incremental modernization, we estimated the reduction rate of this change impact and its effect on reducing the maintenance effort.

1) Procedure for Estimating Benefit: The reduction rate of maintenance effort, R, is determined by the following Equation:

$$R = 1 - \frac{\sum_{m \in M} P_m CI'_m}{\sum_{m \in M} P_m CI_m} \quad (2)$$

In this context, M represents the set of all modules in the entire system. P_m denotes the probability that module m (within the module set M) will be modified during a maintenance activity. CI_m and CI'_m denote the change impact for module m before and after modernization, respectively. The change impact for a module is measured by the lines of code to be tested when the module is modified.

Equation (2) calculates the expected change impact across the entire system by multiplying the change probability of a module with its change impact. The expected change impact was calculated because not all modules will be modified in a single maintenance project. For instance, modules that are frequently modified will have a higher effect when reducing their change impact, whereas less-modified modules will have a lesser effect. Therefore, Equation (2) considers the probability of a change for each module. This study determines the change probability P_m based on the proportion of the module m size to the entire system size. This is based on the assumption that larger modules have a higher likelihood of modification and smaller modules have a lower likelihood. Furthermore, the change impact CI_m is calculated by summing the lines of code for module m and all the modules that depend on module m. This is because the modules that depend on the changed module must be retested for change impact. These concepts are supported by [4].

The expected change impact after modernization is calculated by assuming that dependencies spanning legacy and modernized products will be eliminated. For instance, if a module had a dependency through a function call before decomposition, but communicates via the network after decomposition, the coupling between modules weakens, making testing more accessible. Consequently, the expected change impact is reduced.

Based on the above, we can estimate the maintenance effort reduction rate from the reduction ratio of the expected change impact between before and after modernization.

2) Example of Benefit Estimate Calculation: As shown in Figure 5, if a maintenance activity modifies module D, all code in modules A, B, C, and D must be retested. The probability of a maintenance event changing module D is the ratio of the code size to the total module size (40% from Table III). The contribution to the mean system change impact is the sum of the change impact of the module multiplied by the probability of occurrence. The contributions from all modules were then summed to provide the expected value of the change impact for the system.

When modules C and D are decomposed from modules A and B, communication between modules B and C occurs via the shell API. Therefore, the shell API prevents the expansion of the change impact. Tables III and IV show that the change impact of module C was reduced from 60 to 30 and that of module D from 100 to 70 before and after modernization. Consequently, the expected change impact was reduced from 65 to 44, resulting in a 32.3% reduction in the maintenance effort.

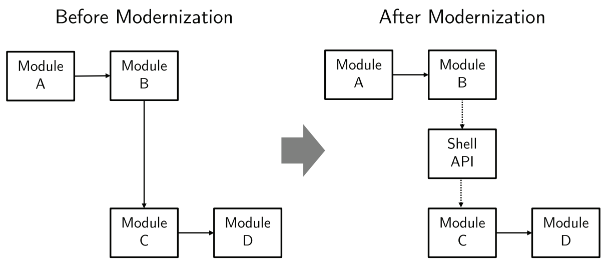


Fig. 5. Examples of modules used for benefit estimation

TABLE III
EXAMPLE OF EXPECTED CHANGE IMPACT BEFORE MODERNIZATION

Module	LOC	P_m	CI_m	Contrib. to Expected
A	10	10%	10	1
B	20	20%	30	6
C	30	30%	60	18
D	40	40%	100	40
		Expected Change Impact		65

TABLE IV
EXAMPLE OF EXPECTED CHANGE IMPACT AFTER MODERNIZATION

Module	LOC	P_m	CI_m	Contrib. to Expected
A	10	10%	10	1
B	20	20%	30	6
C	30	30%	**30**	**9**
D	40	40%	**70**	**28**
		Expected Change Impact		**44**

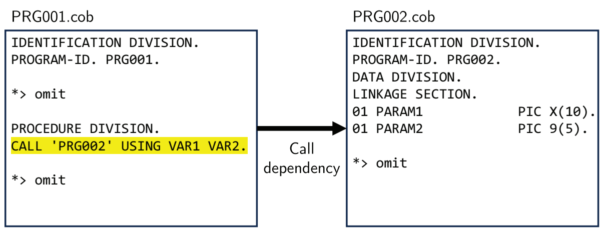
V. Analysis

This analysis aimed to confirm the effectiveness of the proposed estimation approach. In the following, we refer to the estimation approaches described in Section IV as proposed approaches.

A. RQ1

To address RQ1, we applied the cost estimation approach to the target system described in Section III. We estimated the development size and analyzed the gap from the actual value.

First, we analyzed the dependencies of the target system and created a dependency graph. The vertices of the graph represent the source files, and the edges represent the call relationships. In general, COBOL programs are compiled file-by-file, and other programs are called using the CALL statement. Figure 6 shows an example of the COBOL call dependency between two COBOL programs PRG001.cob



```
IDENTIFICATION DIVISION.
PROGRAM-ID. PRG001.

*> omit

PROCEDURE DIVISION.
CALL 'PRG002' USING VAR1 VAR2.

*> omit
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PRG002.
DATA DIVISION.
LINKAGE SECTION.
01 PARAM1             PIC X(10).
01 PARAM2             PIC 9(5).

*> omit
```

Fig. 6. Example of COBOL call dependency

and PRG002.cob. By performing dependency analysis, practitioners can determine which COBOL programs are called from other programs.

After creating a dependency graph, the programs were categorized into those to be extracted and rebuilt as modernized products, and those to be left as legacy products. Subsequently, the number of dependencies between programs rebuilt for the modernized product and those remaining in the legacy product was tallied.

Finally, the overall development size of the system was determined by summing the development sizes of the modernized product, legacy product, and shell API (see Section II-A). The development sizes of legacy and modernized products reuse estimated results from the planning phase of the target project. During the planning phase, developers were unaware of the need to develop shell API in incremental modernization. Consequently, the estimated size of the planning phase was reused in this analysis. The development size for the shell API was estimated using the formula described in Equation (1) in Section IV-B. Note that productivity α is excluded from the calculation in this analysis because we evaluated the size estimation instead of the effort estimation.

The accuracy was evaluated by examining the deviation of the cost model results from the actual development scale of 3,691 KLOC. Compared to the 2,675 KLOC estimated at the planning stage of the target system, we evaluated which method had a lower deviation rate from the actual values, the proposed approach, or the actual values. This gap was calculated using the Magnitude of Relative Error (MRE), as shown in the following Equation:

$$MRE = \frac{|(\text{actual}) - (\text{estimated})|}{(\text{actual})} \tag{3}$$

B. RQ2

To answer RQ2, we applied the cost estimation approach to the target system described in Section III. We estimated the reduction in LOC to be tested and then analyzed its gap from the actual values. Because the cost of the testing phase is relatively high during the development process, reducing the effort of the testing phase leads to a considerable reduction in the overall effort. We estimated the reduction in the test scale based on the assumption that the number of person-hours in the testing process increases or decreases proportionally with the test scale [16], [17].

First, a dependency analysis was performed on the target system, a dependency graph was created, and programs were classified into those extracted and reconstructed as modernized products and those left as legacy products. This process was the same as that described in Section V-A. We estimate the expected change impact before and after modernization using the formula denoted Equation (2).

This study presents the development performance statistics before and after incremental modernization in Table V. We focused on 21 cases of maintenance activity before modernization and 10 cases after modernization. We excluded cases where the production size, test size, and development productivity were incorrectly recorded.

For development performance, we aggregated both the mean and median for the following three items:

- Code to be tested in IT1:
  Lines of code to be tested in integration test 1 (IT1) phase. IT1 is closer to a unit test as it is on a smaller level than IT2.
- Code to be tested in IT2:
  Lines of code to be tested in integration test 2 (IT2) phase. IT2 is closer to a system test as it is on a larger level than IT1.
- Productivity:
  Development person-hours per production size. The unit is KLOC/person-month. Development hours are the sum of the hours from basic design to system testing.

A smaller line of code to be tested indicates a smaller change impact and suggests improved maintainability. A more significant value for the development productivity indicates that the number of implementation lines per person-month is more significant, suggesting improved maintainability. In addition, the development requirements before and after modernization did not match. Therefore, in this study, we compare the development productivity by dividing the development hours by the production size.

With the median of the IT2 Size showing a positive change rate and both the mean and median of Productivity showing negative change rates, we were unable to confirm that maintainability necessarily improved due to incremental modernization.

We analyzed the gap estimation from the development performance shown in Table V. We estimated the performance after modernization by multiplying the reduction rate determined by Equation (2). We then compared the actual development performance after modernization with our estimates to assess the degree of the gap. This gap was calculated using the same formula for the cost estimation MRE.

VI. Results and Lesson Learned

A. RQ1

The number of call dependencies from the modernized product to the legacy product was 3,431, and from the legacy to the modernized was 80, totaling 3,511 inter-system call relationships. The LOC size per program was 343.3 LOC.

TABLE V
DEVELOPMENT PERFORMANCE STATISTICS BEFORE AND AFTER INCREMENTAL MODERNIZATION

		Before Modernization	After Modernization	Change Rate
Number of Cases		21	10	-
Mean	Code to be tested in IT1 [KLOC]	22.75	21.24	-7%
	Code to be tested in IT2 [KLOC]	24.04	23.28	-3%
	Productivity [KLOC/person-month]	0.27	0.19	-30%
Median	Code to be tested in IT1 [KLOC]	14.78	13.53	-8%
	Code to be tested in IT2 [KLOC]	17.77	21.49	+21%
	Productivity [KLOC/person-month]	0.24	0.19	-21%

TABLE VI
COST ESTIMATION: ACTUAL AND ESTIMATED DEVELOPMENT SIZE

	Development Size	MRE
Estimation in this analysis	3,880 KLOC	5.1%
Estimation at the planning phase	2,675 KLOC	27.5%
Actual Value	3,691 KLOC	-

Therefore, the development size of the shell API was estimated to be 1,205 LOC. By adding the estimated size of the shell API to the previously estimated size in the planning phase (excluding the shell API) of 2,675 KLOC, the estimated total development size for the entire system was 3,880 KLOC.

A comparison between the estimated and actual values is shown in Table VI. The development size estimated using the proposed approach was 3,880 KLOC. Compared with the actual development size of 3,691 KLOC, the MRE was 5.1%. The estimation in the planning phase had a gap of 27% from the actual value, suggesting that the proposed approach yielded a closer estimation of the actual results. In the planning phase, the factors contributing to the increased costs of incremental modernization were unrecognized, and the development size of the shell API was excluded from the estimation. Therefore, by adding the development size of the shell API to the estimation during the planning phase, we obtained an estimation closer to the actual value.

Answer to RQ1

Cost estimation can provide more valid estimates because they have fewer errors with actual values than estimates made at the planning stage.

The lesson learned from these results is that incremental modernization, including shell API development size in the estimate using dependency analysis, produces an estimated result close to the actual development size. The development size of the shell API accounts for a large percentage of the entire system development size. Therefore, if the development size of the shell API is excluded from the estimate, the effort required exceeds the estimate. Before the analysis, we were concerned that MRE would be little changed or be worse than that in the planning phase. However, it improved by 22.4%, that is, from 27.5% to 5.1%. This indicates that the proposed approach is reasonable.

B. RQ2

In the target project, the expected change impact before incremental modernization was 7.99 KLOC, whereas, after incremental modernization, it became 6.99 KLOC. From Equation (2), it is estimated that the maintenance effort could be reduced by 13%.

Table VII compares the actual value after incremental modernization; the estimated value reduced by 13% from the actual value before incremental modernization. From Table VII, the estimated value displayed MRE ranging from 5% to 68% compared to the actual values after incremental modernization. Although the degree of MRE varied, the actual values were more significant than the estimated values for the code tested in IT1 and IT2. Furthermore, productivity had smaller actual values. This indicates that no development performance was as effective as estimated.

Answer to RQ2

It is impossible to determine whether the benefit estimation is valid because the maintenance effort has remained the same or increased since modernization.

The lesson learned from these results is that incremental modernization only occasionally improves maintainability. The frequency of change requests, bugs, and future modifications should be considered when extracting programs. Although the benefits of performing incremental modernization have been reported, these benefits are only fully realized if the appropriate functionality is extracted. Therefore, an approach to quantitatively estimate the effect of improved maintainability in modernization is still needed in the future. In addition, using the proposed estimation formula, the estimated result of maintenance effort after restructuring was always smaller than the maintenance effort before. Therefore, it is necessary to study a model to estimate the deterioration in maintainability.

C. Threats to Validity

In the cost-benefit calculation model, this analysis estimated the lines of code to implement per dependency from the average number of lines per program. By conducting proof of concept (PoC), it is possible to determine the lines of code to implement each dependency more precisely. However, conducting PoC during the planning phase is challenging. As shown in Table VI, even with the estimation approach used in this analysis, the MRE is smaller than the estimates at the

TABLE VII
BENEFIT ESTIMATION: ACTUAL AND ESTIMATED DEVELOPMENT PERFORMANCE

		Actual Value (After Modernization)	Estimated Value	MRE
Mean	Code to be tested IT1 [KLOC]	21.24	19.79	7%
	Code to be tested IT2 [KLOC]	23.28	20.91	10%
	Productivity [KLOC/person-month]	0.19	0.32	68%
Median	Code to be tested IT1 [KLOC]	13.53	12.86	5%
	Code to be tested IT2 [KLOC]	21.49	15.46	28%
	Productivity [KLOC/person-month]	0.19	0.27	42%

planning phase, indicating that the estimation accuracy using the average size per program is acceptable.

The benefit estimation has no baseline for setting a threshold for the MRE. Therefore, no upper or lower limit of the MRE can be considered to indicate the accuracy of the effect estimation model. Therefore, determining whether the benefit estimation is accurate is challenging.

Moreover, other metrics exist for evaluating the accuracy of the estimates apart from the MRE. In this study, we answered the RQ based on the MRE from the actual values. However, different results could be obtained if other metrics were adopted.

In this analysis, we analyzed just a single case. All that became established was that the estimated cost of this project was close to the actual cost. From this, we determined that the cost estimate was valid. However, increasing the number of systems under analysis and conducting cross-validation to enhance the reliability of the analysis is necessary.

Additionally, as seen in Table V, the change rate for the code to be tested in IT2 (median) is positive, and productivity (both mean and median values) is negative. This implies that the situation worsened after incremental modernization. Hence, despite the expected reduction in test scope and maintenance effort, we are concerned that expectations will fail after the incremental modernization. This suggests the possibility of validating from the premises.

VII. RELATED WORK

Leitch et al. proposed a method for estimating the cost-benefit of refactoring using dependency analysis [4]. Their approach calculates the possible maintenance cost savings from refactoring based on control and data dependencies and uses COCOMO II [11] to estimate the implementation cost of refactoring. However, Leitch et al. focused only on evaluating the cost-benefit of refactoring within a system, differentiating it from incremental modernization.

Cai et al. developed a framework for evaluating the cost-benefit of refactoring based on the source code, architectural information, and version histories [7]. Although their framework shares similarities with the proposed approach, they did not detail the exact estimation techniques. Their effect estimation also diverges from the proposed approach because it does not focus on the change impact and relies on correlations with past refactoring tasks.

Cui et al. investigated the correlation between methods of modifying dependencies and the costs of these modifications [5]. Their findings enabled estimations of modification work costs for each type of dependency. However, their method did not consider system extraction, which distinguishes it from the cost estimation in this study.

Xiao et al. investigated the correlation between architectural technical debt and maintenance development costs [8]. Their findings enable the estimation of increased maintenance costs owing to existing technical debt. Nevertheless, they did not estimate the reduction in maintenance effort owing to system extractions, setting their method apart from the cost estimation in this study.

Rebêl et al. explored the relationship between design stability and maintenance tasks by contrasting aspect-oriented extractions with object-oriented designs [6]. Their work allowed for the estimation of maintenance cost reductions owing to aspect-oriented extractions. However, their focus on aspect-oriented extractions instead of incremental modernization distinguishes their benefit estimation from the benefit estimation in this study.

Kobayashi et al. proposed a metric to quantify the propagation model of changes to improve system failure prediction accuracy [12]. They used dependency graphs to judge the change impact, similar to the proposed method. However, their primary focus on enhancing failure prediction accuracy, rather than estimating the reduction in development effort, distinguishes their approach from the benefit estimation in this study.

VIII. CONCLUSION

Adopting an incremental modernization strategy can reduce the risk of software system modernization failures. However, modernization costs tend to increase compared to the estimated costs in the planning phase. Furthermore, quantifying the benefits of incremental modernization is challenging, leading to delayed decisions to invest in modernization.

In this study, for systems in which incremental modernization was conducted, we applied a cost-benefit estimation approach using dependency analysis and compared the gap with the actual value. As a result of this analysis, we confirmed that the cost estimates are valid estimates, but we cannot judge whether the benefit estimates are valid.

Future works include the following three points:

- **Improving the reliability of analysis results:** In this analysis, the analysis has been conducted on just a single case. Increasing the number of systems under analysis and conducting cross-validation to enhance the reliability of the analysis is future work.

- **Expansion of dependency relationships:** In this analysis, we analyzed only the call dependencies, but other dependencies exist (e.g., access to databases). Expanding the dependency types and analyzing whether the accuracy of the estimated improves is future work.
- **Application to other languages:** In this analysis, estimations were conducted targeting the COBOL language. In this analysis, we performed our estimation on COBOL programs. We are applying the proposed approach to languages other than COBOL and analyzing whether accurate estimation results can be obtained in future work.

Acknowledgment

This study benefited greatly from discussions with Katsuro Inoue (Nanzan University, Japan), Makoto Matsushita (Osaka University, Japan), and Eiju Ueda (NTT DATA Group, Japan).

References

[1] J. Bisbal, D. Lawless, R. Richardson, D. O'Sullivan, B. Wu, J. B. Grimson, and V. P. Wade, "A survey of research into legacy system migration," in *Proc. of the International Conference on Informatics and Analytics*, 2007.

[2] M. L. Brodie and M. Stonebraker, "Darwin: On the incremental migration of legacy information systems," GTE Laboratories, Inc, Tech. Rep., 1993.

[3] S. Comella-Dorda, G. Lewis, P. Place, D. Plakosh, and R. Seacord, "Incremental modernization of legacy systems," Carnegie Mellon University, Software Engineering Institute's Digital Library, Tech. Rep., 2001.

[4] R. Leitch and E. Stroulia, "Assessing the maintainability benefits of design restructuring using dependency analysis," in *Proc. of METRICS*, 2003, p. 309.

[5] D. Cui, L. Fan, S. Chen, Y. Cai, Q. Zheng, Y. Liu, and T. Liu, "Towards characterizing bug fixes through dependency-level changes in apache java open source projects," *Science China Information Sciences*, vol. 65, 2022.

[6] H. Rebêl, R. M. F. Lima, U. Kulesza, M. Ribeiro, Y. Cai, R. Coelho, C. Sant'Anna, and A. Mota, "Quantifying the effects of aspectual decompositions on design by contract modularization: a maintenance study," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 23, no. 7, pp. 913–942, 2013.

[7] Y. Cai, R. Kazman, C. V. Silva, L. Xiao, and H.-M. Chen, "Chapter 6 - a decision-support system approach to economics-driven modularity evaluation," in *Economics-Driven Software Architecture*. Morgan Kaufmann, 2014, pp. 105–128.

[8] L. Xiao, Y. Cai, R. Kazman, R. Mo, and Q. Feng, "Detecting the locations and predicting the maintenance costs of compound architectural debts," *IEEE Trans. Softw. Eng.*, vol. 48, no. 9, pp. 3686–3715, 2022.

[9] H. Michael Ayas, P. Leitner, and R. Hebig, "An empirical study of the systemic and technical migration towards microservices," *Empir. Softw. Eng.*, vol. 28, no. 85, 2023.

[10] A. Arifoglu, "A methodology for software cost estimation," *SIGSOFT Softw. Eng. Notes*, vol. 18, no. 2, p. 96–105, 1993.

[11] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece, *Software Cost Estimation with COCOMO II*. Prentice Hall, 2000.

[12] K. Kobayashi, A. Matsuo, K. Inoue, Y. Hayase, M. Kamimura, and T. Yoshino, "Impactscale: Quantifying change impact to predict faults in large software systems," in *Proc. of ICSM*, 2011, pp. 43–52.

[13] T. A. Corbi, "Program understanding: Challenge for the 1990s," *IBM Syst.J.*, vol. 28, no. 2, pp. 294–306, 1989.

[14] C. L. McClure, *The three Rs of software automation : re-engineering, repository, reusability*. Prentice Hall, 1992.

[15] M. J. Harrold, J. A. Jones, T. Li, D. Liang, A. Orso, M. Pennings, S. Sinha, S. A. Spoon, and A. Gujarathi, "Regression test selection for java software," in *Proc. of OOPSLA*. Association for Computing Machinery, 2001, p. 312–326.

[16] A. Ko, H. H. Aung, and B. Myers, "Eliciting design requirements for maintenance-oriented ides: a detailed study of corrective and perfective maintenance tasks," in *Proc. of ICSE*, 2005, pp. 126–135.

[17] L. Briand, Y. Labiche, and G. Soccar, "Automating impact analysis and regression test selection based on uml designs," in *Proc. of ICSM*, 2002, pp. 252–261.