段階的再構築における依存関係分析を用いた費用対効果の試算

横井 一輝　崔 恩瀞　吉田 則裕　岡田 譲二　肥後 芳樹

数十年稼働してきたソフトウェアシステムを刷新する，モダナイゼーションの需要は大きい．モダナイゼーションの失敗リスクを低減する戦略として，システムの一部を切り出す段階的再構築が提案されている．しかし段階的再構築では，旧システムと新システムを組み合わさることによる複雑性が生じるため，再構築コストが増大する傾向がある．さらに，企業におけるモダナイゼーションでは，事前に再構築の効果を顧客に説明する必要があるため，計画段階で段階的再構築の費用対効果を見積もることが重要である．そこで，本稿ではシステムの依存関係分析を通じて段階的再構築における費用対効果を事前に試算する手法を提案する．提案手法は，システムの規模と依存関係の情報を考慮し，費用として再構築に必要な工数を，効果として削減可能な保守開発工数を試算する．評価実験では，過去に段階的再構築を実施した事例に提案手法を適用し，実際の工数と比較した．

1　はじめに

数十年稼働してきたソフトウェアシステムを刷新し，ビジネス上の変化への迅速な対応や保守コストの削減を目的とするモダナイゼーションの需要は大きい．対象となるシステムは，ビジネス上重要である場合が多く，失敗が許されない場合が多い．モダナイゼーションの失敗リスクを低減する戦略として，システムの一部を切り出す段階的再構築が提案されている [2]．段階的再構築では，徐々に旧システムを新システムに置き換えることで，移行失敗のリスクを抑えながらモダナイゼーションを完了できる．

段階的再構築は移行失敗のリスクを低減する一方で，旧システムと新システムが組み合わさることによる複雑性が生じるため，再構築コストが増大する傾向がある．さらに，企業におけるモダナイゼーションでは，事前に再構築の効果を顧客に説明する必要があるため，計画段階で段階的再構築の費用対効果を見積もることが重要である．

しかし，段階的再構築の費用対効果を計画段階で見積もることは困難である．なぜなら，開発前の計画段階では設計書がまだ存在しないため，開発に必要な費用や得られる効果を正確に把握できないためである．また，段階的再構築の困難さを早期に定量的に把握することが難しく，再構築コストが計画段階よりも大きく超過してしまう．著者らが所属する企業における再構築プロジェクトにおいても同様の問題により，計画段階より再構築コストが大きく超過した事例が存在する．

そこで，本稿ではシステムの依存関係分析を通じて段階的再構築における費用対効果を事前に試算する手法を評価する．過去に段階的再構築を実施した金融系基幹システムを対象に，費用と効果を試算することで，依存関係分析を用いた見積手法の実績値との乖離率を検証する．

以降，2 章では，本研究の背景について述べる．3

---

* Estimating Cost-Effectiveness Using Dependency Analysis in Incremental Modernization

Kazuki Yokoi, 株式会社 NTT データグループ, NTT DATA Group Corporation.
Eunjong Choi, 京都工芸繊維大学, Kyoto Institute of Technology.
Norihiro Yoshida, 立命館大学, Ritsumeikan University.
Jouji Okada, 株式会社 NTT データグループ, NTT DATA Group Corporation.
Yoshiki Higo, 大阪大学, Osaka University.

章では，本研究で評価対象とした段階的再構築を実施したシステムについて述べる．4 章では，本研究における見積手法について述べる．5 章では，評価実験について述べる．6 章では，関連研究について述べる．最後に，7 章でまとめと今後の課題について述べる．

2 背景

2.1 段階的な再構築

システム全体を一度で再構築する戦略であるビッグバンアプローチは，移行失敗のリスクが高い．ビッグバンアプローチの移行失敗のリスクを回避するため，Brodie らは Chicken-Little 戦略という段階的な移行戦略を提案した [2]．Chicken-Little 戦略の中心的な考え方は，旧システムと新システムからなる複合システムを構築することである．この旧システムと新システムは，ゲートウェイを介して連携する．最初に，旧システムの一部が新システムに置き換えられ，ゲートウェイが新システムにリクエストをルーティングする．そして，徐々に旧システムを新システムに置き換える割合を増やしていく．これにより，新システムは徐々に拡大し，段階的な移行が実現する．

Chicken-Little 戦略は移行失敗のリスク低減する一方で，複合システムとしての複雑性に伴うリスクが増大する．そのリスクを次に示す．

1. 旧システムと新システム間の通信制御が複雑になる．
2. 旧システムと新システムのデータ整合性の維持が困難になる．

これらリスクに対応するため，旧システムと新システムを連携するグルーコードが必要になる．グルーコードとは，異なるソフトウェアコンポーネントやシステム間での連携を可能にするコード片やスクリプトを指す．これにより，独立して開発されたコンポーネントやシステムを連携して機能するようにできる．段階的再構築では，旧システムと新システムで異なる技術スタックやプラットフォーム間の連携が必要になるため，グルーコードによってシステム間の通信制御やデータ整合性制御が必要である．そのため，段階的再構築ではグルーコードの開発コストが追加で発生し，システム全体での開発コストの合計は増大する．

2.2 見積もり

見積りでは，時期によって最終的な実績値と比較して誤差が発生する．特に開発初期の計画段階ではプロジェクト全体に不明確な点が多いため，誤差が大きくなる傾向にある．しかし，予算獲得などのおおよその計画を立てるために，早い時期での見積りは必要である．そのため，見積りの不確実性を事前に評価して関係者の間で共有し，適切なタイミングで再見積もりを実施することが重要である．本研究では，開発初期の計画段階における見積りを対象とする．

見積もり作業では，規模，工数，工期，品質（信頼性，性能など），コストなどのさまざまな要素を見積もる [15]．見積もりは，類推法，積み上げ法，パラメトリック法の 3 つに大きく分類される．

類推法とは，過去の類似プロジェクトの実績を基礎に見積る方法である．類推方は簡便でシステム開発初期の見積りに適している一方で，過去の実績プロジェクトについてその背景や制約，特徴などが明らかでないと適用が困難である．情報処理推進機構が開発規模，工数などの実績値を公開している [16] が，段階的再構築の分類がなく，類推法を使った段階的再構築の見積は難しい．

積み上げ法は，プロジェクトの成果物の構成要素を洗い出し，それぞれに必要な工数などを見積って積み上げる方法である．積み上げ法は見積もり精度が高い一方で，プロジェクトの構成要素としての作業項目を WBS（Work Breakdown Structure），成果物の構成要素としてのサブシステムやコンポーネントなどを事前に洗い出しておく必要がある．したがって，不確実なものが多い初期見積りでは難しい．

パラメトリック法は，工数などを目的変数として，説明変数に規模や要因などを設定し，数学的な関数として表す方法である．パラメトリック法では，工数と規模が正比例（工数 $= \alpha \times$ 規模）するもの，工数と規模の累乗が比例（工数 $= \alpha \times$ 規模$^{\beta}$）するものが主に利用されている．工数と規模の累乗が比例する関係式は，COCOMO 法でも採用されている．独立行政法人情報処理推進機構では，工数と規模が正比例するパラメトリック法の考え方に則ってソフトウェア開発の定量データを収集，分析し，公開している [16]．本

項では，このデータを用いてパラメトリック法で工数を見積もる方法を説明する．

3 ケーススタディ：段階的再構築を実施したシステム

本研究では，過去に段階的再構築を実施したシステムを対象に評価を行った．再構築前のシステム概要は以下のとおりである．

概要 金融系基幹システム

システム区分 バッチシステムおよびオンラインシステム

言語区分 オープン COBOL

システム規模 約 4.6MSLOC（COBOL のみ対象）

再構築では，上記のシステムから約 870KLOC を一部機能を Java プログラムとして切り出し，段階的再構築を実施した．しかし，当プロジェクトでは段階的再構築の複合システムとしての複雑性リスクを計画時点で十分に認識できておらず，計画時点の試算結果を実績値が 27.5%超過することとなった．また，保守性が向上することを目的として段階的再構築を実施したが，再構築前後の開発実績を比較したところ必ずしも保守性が向上したとは言い難い結果が得られた．

再構築前後の開発実績の統計値を表 1 に示す．本研究では，再構築前 21 件，再構築後 10 件の保守開発案件を集計対象とした．再構築前では COBOL の基幹システム，再構築後では切り出し対象となったサブシステムを保守対象とする案件に絞り込み，そこから製造規模，テスト規模，開発生産性が正しく記録されていない案件を除外した．

開発実績では，以下の 3 項目に対して平均値と中央値の両方で集計した．

結合テスト 1 規模 1 回目の結合テストでテスト対象となるソースコード規模．1 回目は 2 回目よりレベルが小さく，単体テストに近い．

結合テスト 2 規模 2 回目の結合テストでテスト対象となるソースコード規模．2 回目は 1 回目よりレベルが大きく，システムテストに近い．

開発生産性 製造規模あたりの開発工数．単位は KLOC/人月．開発工数は基本設計からシステムテストまでの工数の合計．

テスト規模は値が小さいほど影響範囲が小さいことを示し，保守性が向上していることを示す．開発生産性は値が大きいほど 1 人月あたりの実装行数が大きいことを示しており，保守性が向上していることを示す．再構築前後で開発要件は一致しない．そのため再構築前後で 1 案件あたりの開発要件に違いが大きいと，1 案件あたりの開発工数の統計値を比較した際に，開発要件の影響を受ける懸念がある．そのため，本研究では開発工数を製造規模で割った開発生産性を比較する．結合テスト 2 規模の中央値では増減率が正に，開発生産性の平均値と中央値はどちらも増減率が負になっており，必ずしも段階的再構築により保守性が向上したとは確認できない．

計画時点の試算結果を実績値が超過したことと，段階的再構築による保守性向上効果が充分に得られなかった経験から，当社では段階的再構築の費用および効果を計画段階で試算することに課題を抱えている．しかし，情報処理推進機構など社外の公開情報を調査しても，段階的再構築の開発事例は公開されていない．また，見積り手法やリファクタリングの費用対効果試算に関する既存研究はいくつかあるが，段階的再構築における費用対効果試算の研究は行われていない．そこで本研究では，段階的再構築案件では現行システムのソースコードを入手できることに着目し，ソースコード上の依存関係分析により費用対効果を試算する方法を調査した．

4 本研究における見積手法

本研究では，依存関係分析によりプログラム間の依存グラフを作成し，作成した依存グラフをもとに段階的再構築の費用と効果を試算する．本章では最初に依存グラフについて説明し，費用試算，効果試算について説明する．

4.1 依存グラフ

まず本稿で用いる依存グラフについて述べる．ソフトウェアの構成要素であるコードエンティティ（メソッドや関数など）やデータエンティティ（データベースのテーブルやグローバル変数など）をノードとし，ノード間の依存関係を辺（本稿では，辺はすべて

表 1　段階的再構築前後の開発実績の統計値

		再構築前	再構築後	増減率
案件数		21 件	10 件	-
平均値	結合テスト 1 規模 [KLOC]	22.75	21.24	-7%
	結合テスト 2 規模 [KLOC]	24.04	23.28	-3%
	開発生産性 [KLOC/人月]	0.27	0.19	-30%
中央値	結合テスト 1 規模 [KLOC]	14.78	13.53	-8%
	結合テスト 2 規模 [KLOC]	17.77	21.49	+21%
	開発生産性 [KLOC/人月]	0.24	0.19	-21%

有向辺）とするグラフ構造で表したものを依存グラフと呼ぶ[13]．依存関係には，コール依存やデータ依存など様々な種類があり，この種類を「関係タイプ」と呼ぶ．

4.2　費用試算

費用試算では，段階的再構築する際の開発工数を試算する．工数見積もりでは，試算した工数に対して人月単価を掛け合わせることで金額に換算することが可能である．

システム全体を再構築する場合，システム全体の規模に対して再構築の生産性を掛けることでパラメトリック法に従い開発工数を試算する．段階的再構築ではシステムの一部を切り出して再構築するが，切り出す範囲の規模に生産性を掛けるだけでは，工数見積もりが実績値より少ない試算値となった事例があった．そこで本研究では，新システム，旧システム，グルーコードの開発工数の合計によって段階的再構築の開発工数を求める手法を提案する．

新システムと既存システムは，切り出し前のシステム規模と開発生産性により，開発工数を求める．一方で，グルーコードの開発工数の見積もりに，依存グラフを用いる．本稿では，グルーコードの開発工数の見積もり方を中心に説明する．

4.2.1　費用試算の手順

グルーコードの開発工数 E は，次の式で求める．

$$E = \alpha \times \sum_{d \in D} N_d L_d \qquad (1)$$

ここでは，N_d は新システムと旧システムをまたがる依存関係 d の本数，L_d は依存関係 d1 本あたりのグルーコードの実装行数，α は生産性（人月/規模）を意味する．これらを，依存関係の種類 d ごとに合計する．

新システムを旧システムから切り出す場合，新システムと旧システム間の依存関係が分断される．例えば，切り出し前は関数呼び出しでつながっていた場合，切り出し後はネットワーク経由の通信となる．そのため通信方式を変換するための連携部品を開発するコストが発生する．

「依存関係 1 本あたりのグルーコード実装行数」と「生産性」は，既存システムと新システムのアーキテクチャにより変動する．これらの試算は，過去の開発実績を元に試算したり，PoC でグルーコードを開発してデータを集めたりすることで数値を決定する．

4.3　効果試算

モダナイゼーションの効果はさまざまなものがあるが，本研究では段階的再構築後の保守開発工数の削減率を効果として試算する．保守開発工数を削減することで，保守開発期間の短期化や保守開発コストの削減につながる．これにより，1 回あたりの保守開発期間が短くなることで開発アジリティが向上したり，保守開発コストの削減されることで他の IT 分野へ投資が行いやすくなったりなど，モダナイゼーションの効果につながる．

効果試算では，段階的再構築の後に 1 回あたりの保守開発工数をどれだけ削減できるかを効果として試算する．

一般に，ソフトウェア保守では理解とテストの労力

が大きいと言われており [7] [4] [10] [6]，その労力が影響範囲に強い影響を受けることがわかっている [8] [1]．そのため，保守開発工数の削減率は影響範囲の削減率から大きな影響を受けると考えられる．

影響範囲とは，一部のプログラムを変更した場合に，その変更が他のプログラムにも影響を及ぼす範囲を意味する．影響範囲が広いほど，プログラムを理解しテストする範囲が広がるため，保守開発工数も増大する．段階的再構築によってシステムを切り出すことで，この影響範囲の縮小率を求め，それによる保守開発工数の削減率を効果として見積もる．

4.3.1 効果試算の手順

保守開発工数の削減率 R は，次の式で求める．

$$R = 1 - \frac{\sum_{m \in M} P_m Imp'_m}{\sum_{m \in M} P_m Imp_m} \qquad (2)$$

ここでは，M はシステム全体のモジュール集合，P_m はモジュール集合 M に含まれるあるモジュール m が一度の保守開発で変更される確率，Imp_m は再構築前のモジュール m の影響範囲，Imp'_m は再構築後のモジュール m の影響範囲を意味する．

式 2 では，あるモジュールが変更される確率とそのモジュールの影響範囲をかけあわせることで，システム全体での影響範囲の期待値を求める．なぜなら，1 回の保守開発で全てのモジュールが変更されることはないからである．たとえば，頻繁に変更されるモジュールは影響範囲を小さくする効果は高く，めったに変更されないモジュールは影響範囲を小さくする効果は少ない．したがって，式 2 では各モジュールの変更確率を考慮する．本研究において変更確率 $P(m)$ は，モジュール m の規模がシステム全体の規模の中で占める割合により求める．これは，規模が大きいモジュールは変更される確率が高く，規模が小さいモジュールは変更される確率が低いという仮定 [9] に基づいている．

再構築後の影響範囲の期待値は，新システムと旧システムをまたがる依存関係がなくなるものとして試算する．例えば，切り出し前は関数呼び出しで依存関係のあるモジュールが，切り出し後はネットワーク経由の通信となる場合，一般的にモジュール間の結合度は弱くなりテストが行いやすくなる．そのため影響範囲の期待値は小さくなる．

以上により，再構築前と再構築後の影響範囲の期待値の縮小率から，保守開発工数の削減率を効果として求めることができる，

5 評価実験

5.1 評価目的とリサーチクエスチョン

評価目的は，本研究で提案する見積りモデルがどれだけ正確か調査することである．4 章で説明した見積手法のことを，以降では提案手法という．

そこで，本実験では次の 2 つのリサーチクエスチョンを設定した．

RQ1 費用試算結果と実績値の乖離はどれくらいか？

RQ2 効果試算結果と実績値の乖離はどれくらいか？

この 2 つのリサーチクエスチョンを解くことで，費用試算と効果試算の 2 つの観点から依存関係分析を用いた見積手法の正確性を評価する．

5.2 評価方法

5.2.1 評価 1：費用試算結果の評価

RQ1 に回答するため，過去に段階的再構築を行ったシステムに対して提案手法を適用して開発規模を試算し，実績値との乖離を評価する．

最初に，対象システムに対して依存関係解析を行い依存グラフを作成する．依存グラフの頂点はファイル，辺は呼び出し関係とする．COBOL プログラムでは，ファイル単位でプログラムにコンパイルされ，CALL 文を使って他プログラムを呼び出す．依存関係解析を行うことで，どの COBOL プログラムからどの COBOL プログラムを呼び出しているかを解析できる．

依存グラフを作成した次は，新システムとして切り出して再構築するプログラムと，旧システムとして残すプログラムに分類する．実際に評価対象システムにて再構築の切り出し対象として選定された COBOL プログラムを，本実験においても切り出し対象として選定した．そして，新システムとして切り出し対象の

プログラムと，旧システムに残すプログラム間で跨る依存関係の本数を集計した．

最後にシステム全体の開発規模は，新システム，旧システム，グルーコードの開発規模の合計で求める．新システムおよび旧システムの開発規模は，検証対象プロジェクトの計画時点の試算を再利用する．検証対象プロジェクトでは，計画時点では段階的再構築の際にグルーコードを開発する必要性を認識できておらず，新システムの旧システムの開発規模のみを試算した．そのため，本評価実験では計画時点の試算を再利用する．グルーコードの開発工数は，4.2.1 項で述べた式（1）より求める．ただし，今回は工数見積もりではなく規模見積りで評価するため，生産性 α は計算に含めない．

正確性の評価では，開発規模の実績値 3,691KLOC に対して，コストモデルの試算結果がどれだけ乖離しているか評価する．また，対象システムの計画段階当時の試算 2,675KLOC と比較して，提案手法とどちらが実績値との乖離率が小さいか評価する．費用試算結果の乖離率は次の式で求める．

$$\text{乖離率} = \frac{\text{試算値} - \text{実績値}}{\text{実績値}} \qquad (3)$$

5.2.2 評価 2：効果試算結果の評価

RQ2 に回答するため，過去に段階的再構築を行ったシステムに対して提案手法を適用し，テスト規模の削減量を試算し，実績値との乖離を評価した．テスト工程は開発工程の中で占める割合が大きく，開発工程全体へ与える影響が大きい．テスト規模に比例してテスト工程の工数は増減するという仮定のもと，テスト規模の削減量を試算する．

最初に，対象システムに対して依存関係解析を行い依存グラフを作成し，新システムとして切り出して再構築するプログラムと，旧システムとして残すプログラムに分類する．ここまでは 5.2.1 項と同様である．そして 4.3.1 項で述べた式（2）より，再構築前後の影響範囲の期待値を求める．

表 1 で示した再構築前の各開発実績に対して，上で求めた削減率を掛け合わせることで，再構築後の値を試算する．その後，再構築後の各開発実績と試算結果を比較することで，試算結果がどれだけ乖離しているか評価する．効果試算結果の乖離率も，費用試算の乖離率と同様の式で求める．

表 2　費用試算：段階的再構築の開発規模

	値	乖離率
コストモデルの試算	3,880KLOC	+5.1%
計画時点の試算	2,675KLOC	-27.5%
実績値	3,691KLOC	-

5.3 評価結果と考察

5.3.1 評価 1：費用試算結果の評価結果

新システムから旧システムへの呼び出し関係は 3,431 本，旧システムから新システムへのよびだし関係は 80 本，合計してシステム間を跨る呼び出し関係は 3,511 本となった．また，プログラム 1 個あたりの SLOC 規模は，343.3SLOC だった．これにより，グルーコードの開発規模は，1,205SLOC と試算できる．試算したグルーコードの開発規模と，グルーコードを含んでいない計画時点の試算 2,675KLOC を合計することで，システム全体の開発規模は 3,880KLOC という試算結果が得られた．

試算結果と実績値の比較を表 2 に示す．提案手法のコストモデルで試算した開発規模は 3,880KLOC となった．実際の開発規模の実績値 3,691KLOC と比較すると，乖離率は+5.1%であった．

5.3.2 評価 1：費用試算結果に対する考察

表 2 より，提案手法のコストモデルで試算した開発規模は，実際の開発規模の実績値と比較して乖離は+5.1%であった．プロジェクト計画時点の試算結果は，実績値との乖離が-27%だったことから，提案手法を用いた方がより実績値に近い試算結果が得られた．計画時点では段階的再構築のコスト増大要因を認識できておらず，グルーコードの開発規模を計画時点の試算に入れていなかった．そのため，プロジェクト計画時点の試算結果にグルーコードの開発規模を足し合わせることで，実績値により近い試算結果が得られたと考える．

RQ1 への回答

提案手法の試算結果と実績値の乖離は+5.1%で，プロジェクト計画時点の試算結果より乖離の少ない試算ができた．

5.3.3 評価 2：効果試算結果の評価結果

再構築前の影響範囲の期待値は 7.99KLOC，再構築後の影響範囲の期待値は 6.99KLOC となった．式 2 より，再構築前後で保守開発工数は 13%削減できるという試算結果が得られた．

再構築後の実績値と，再構築前の実績値に対して 13%削減した試算結果の比較を表 3 に示す．結合テスト 1 および 2 の規模において，提案手法の効果モデルで試算結果は実際の再構築後の実績値より小さい値となった．また，開発生産性は試算結果が実績値より大きくなった．

5.3.4 評価 2：効果試算結果に対する考察

表 3 より，提案手法の効果モデルで試算結果は，実際の再構築後の実績値と比較して乖離は-28%〜+68%であった．乖離の程度はさまざまであるが，テスト規模は実績値の方が大きく，開発生産性は実績値の方が小さいため，全てにおいて試算結果ほど効果を得ることはできなかった．

表 1 より，再構築前後の削減率が正である結合テスト 1（平均値と中央値）および結合テスト 2（平均値）においては，再構築後の実績値と効果モデルの試算結果の乖離が-10%以上と，乖離が小さくなっている．一方で再構築前後の削減率が負の比較項目に関しては，乖離率が-28%と乖離が大きくなっている．

RQ2 への回答

提案手法の試算結果と実績値の乖離は-77%〜+68%となった．ただし一部比較項目において，再開発前より再開発後の実績値が悪化しており，テスト規模削減および生産性向上が前提である効果モデルの正確性の評価が難しい．

5.3.5 妥当性の脅威

コスト試算モデルにおいて，本実験では依存関係 1 本あたりの実装行数を，プログラム 1 個あたりの平均行数から試算した．実際には事前検証などを実施することで，依存関係 1 本あたりの実装行数をより精緻に求めることができる．しかし，一般的な実開発において計画段階で事前検証を行うことは難しい．表 2 より，本実験の計算方法でも計画時点の試算より乖離率が小さいことから，プログラム 1 個あたりの平均行数を使った試算精度であっても許容できる．

効果試算モデルにおいて，乖離率の閾値を決めるベースラインが存在していないため，効果試算モデルが正確性があると判断できる乖離率の上限・加減が存在しない．そのため，効果試算モデルが正確であるか否かの判断が難しい．

さらに，見積りの正確性の評価指標として乖離率以外の指標も存在する．本研究では実績値との乖離率をもって RQ の回答としたが，別の指標を採用することでことなる結果となる可能性がある．

また，表 1 より，結合テスト 2 規模（中央値）の増減率が正に，また開発生産性（平均値と中央値）の増減率が負になっており，むしろ再構築前と比較して再構築後は悪化している．したがって，段階的再構築によりテスト規模の削減および保守開発工数の削減効果が得られない懸念があり，前提から検証する可能性が考えられる．

6 関連研究

Leitch らは，リファクタリングにおける依存関係分析を用いた費用対効果の試算方法を提案した [9] ．Leitch らの手法では，制御依存とデータ依存の依存関係からリファクタリングにより削減可能なシステムの保守コストを試算する．また COCOMO II を用いてリファクタリングの実施コストを試算する．しかし，Leitch らの手法ではシステム内のリファクタリングの際の投資対効果を算出することしかできず，段階的再構築の投資対効果を試算する点で異なる．

Cui らは，依存関係の修正方法と修正作業コストの相関関係を調査した [5]．これにより，依存関係の種類ごとに修正作業コストを見積もることができる．ただし，依存関係の修正方法にシステム切り出しがない点が，提案手法の費用試算とは異なる．

Rebêl らは，アスペクト指向切り出しにより，オブジェクト指向設計と比較して，設計上の安定性と保守

表 3　効果試算：段階的再構築前後の比較

		実績値（再構築後）	効果モデルの試算	乖離率
平均値	結合テスト 1 規模 [KLOC]	21.24	19.79	-7%
	結合テスト 2 規模 [KLOC]	23.28	20.91	-10%
	開発生産性 [KLOC/人月]	0.19	0.32	+68%
中央値	結合テスト 1 規模 [KLOC]	13.53	12.86	-5%
	結合テスト 2 規模 [KLOC]	21.49	15.46	-28%
	開発生産性 [KLOC/人月]	0.19	0.27	+42%

作業の関係性を調査した [11]．これにより，アスペクト指向切り出しによる保守開発コストの削減効果を試算できる．ただし，アスペクト指向切り出しではなく，別システムに切り出す点で，提案手法の効果試算とは異なる．

Cai らは，ソースコード，アーキテクチャ情報，バージョン履歴をもとに，リファクタリングの費用対効果を判断するフレームワークを開発した [3]．費用対効果を判断するフレームワークとして，提案手法と類似している．一方で費用試算の具体的な試算方法は示されていない．また効果試算は影響範囲に着目しておらず，過去のリファクタリング作業との相関関係により試算する点で，提案手法とは異なる．

Xiao らは，アーキテクチャの技術的負債と保守開発コストとの相関関係を調査した [12]．これにより，技術的負債の存在による保守開発コストの増加量を試算できる．ただし，システム切り出しによって保守開発工数の削減量を試算しない点で，提案手法とは異なる．

小林らは，システム障害予測の精度向上を目的として，変更の影響波及量を定量化するメトリクスを提案した [13]．依存グラフを用いて変更の影響範囲を見極める点で，提案手法と類似しているが，障害予測の精度向上を目的としており開発工数の削減量試算を目的としていない点で，提案手法と異なる．

早瀬らは，保守開発作業の労力見積りに用いることを目的として，影響波及解析を使ったメトリクスを提案した [14]．依存グラフを用いて変更の影響範囲を見極める点で，提案手法と類似している．しかし早瀬らの手法は，特定のプログラムの保守コストの試算のみ実施しており，システム全体での保守コストの削減率を試算する提案手法と異なる．

7　まとめと今後の課題

段階的再構築の戦略を採用することで，ソフトウェアシステムのモダナイゼーション失敗リスクを軽減できるが，再構築コストが計画段階より大きく増大する傾向がある．また，段階的再構築の効果の定量化が難しく，再構築の投資判断が進まない現状がある．そこで本研究では，書くの段階的再構築を実施したシステムに対して，依存関係分析を用いた費用対効果試算の見積手法を適用し，実績値との乖離度合いを比較した．評価の結果，費用試算は実績値と乖離率が小さかったのに対し，効果試算は実績値との乖離が大きかった．

今後の課題として，以下の 3 点が挙げられる．

評価結果の信頼性向上　現時点では 1 つの案件でのみ評価を行なった．評価対象のシステムを増やして交差検証を行うなど，評価結果の信頼性を向上させることは今後の課題である．

依存関係の拡充　本研究ではプログラムの呼び出し関係のみを依存関係として解析した．他にもデータベースアクセスなど，依存関係の種類は存在する．依存関係の種類を増やし，試算結果の精度が向上するのかといった評価は今後の課題である．

他言語への適用　本研究では，COBOL 言語を対象に試算を行った．COBOL 以外の言語でも提案手法を適用して試算結果が得られるのか，今後の課題である．

謝辞 本論文の執筆に際し助言および議論していただいた井上 克郎氏 (南山大学)，松下 誠氏 (大阪大学)，上田 永樹氏 (NTT データグループ) に感謝する．

参 考 文 献

[1] Briand, L., Labiche, Y., and Soccar, G.: Automating impact analysis and regression test selection based on UML designs, *International Conference on Software Maintenance, 2002. Proceedings.*, 2002, pp. 252–261.

[2] Brodie, M. L., Stonebraker, M., and Ai, S.: DARWIN: On the Incremental Migration of Legacy Information Systems, 1993.

[3] Cai, Y., Kazman, R., Silva, C. V., Xiao, L., and Chen, H.-M.: Chapter 6 - A Decision-Support System Approach to Economics-Driven Modularity Evaluation, *Economics-Driven Software Architecture*, Mistrik, I., Bahsoon, R., Kazman, R., and Zhang, Y.(eds.), Morgan Kaufmann, Boston, 2014, pp. 105–128.

[4] Corbi, T. A.: Program understanding: Challenge for the 1990s, *IBM Systems Journal*, Vol. 28, No. 2(1989), pp. 294–306.

[5] Cui, D., Fan, L., Chen, S., Cai, Y., Zheng, Q., Liu, Y., and Liu, T.: Towards characterizing bug fixes through dependency-level changes in Apache Java open source projects, *Science China Information Sciences*, Vol. 65(2022).

[6] Harrold, M. J., Jones, J. A., Li, T., Liang, D., Orso, A., Pennings, M., Sinha, S., Spoon, S. A., and Gujarathi, A.: Regression Test Selection for Java Software, *Proceedings of the 16th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA '01, New York, NY, USA, Association for Computing Machinery, 2001, pp. 312–326.

[7] K., F. R.: Application program maintenance study : Report to our respondents, *GUIDE*, Vol. 48(1983).

[8] Ko, A., Aung, H. H., and Myers, B.: Eliciting design requirements for maintenance-oriented IDEs: a detailed study of corrective and perfective maintenance tasks, *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, 2005, pp. 126–135.

[9] Leitch, R. and Stroulia, E.: Assessing the Maintainability Benefits of Design Restructuring Using Dependency Analysis, *Proc. of the 9th International Symposium on Software Metrics*, 2003, pp. 309.

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

[11] Rebêl, H., Lima, R. M. F., Kulesza, U., Ribeiro, M., Cai, Y., Coelho, R., Sant'Anna, C., and Mota, A.: Quantifying the effects of Aspectual Decompositions on Design by Contract Modularization: a Maintenance Study, *Int. J. Softw. Eng. Knowl. Eng.*, Vol. 23, No. 7(2013), pp. 913–942.

[12] Xiao, L., Cai, Y., Kazman, R., Mo, R., and Feng, Q.: Detecting the Locations and Predicting the Maintenance Costs of Compound Architectural Debts, *IEEE Transactions on Software Engineering*, Vol. 48, No. 9(2022), pp. 3686–3715.

[13] 小林健一, 松尾明彦, 井上克郎, 早瀬康裕, 上村学, 吉野利明: 大規模ソフトウェア保守のための影響波及量尺度インパクトスケール, 情報処理学会論文誌, Vol. 54, No. 2(2013), pp. 870–882.

[14] 早瀬康裕, 松下誠, 楠本真二, 井上克郎, 小林健一, 吉野利明: 影響波及解析を利用した保守作業の労力見積りに用いるメトリックスの提案, 電子情報通信学会論文誌 *D*, Vol. J90-D, No. 10(2007), pp. 2736–2745.

[15] 独立行政法人情報処理推進機構: ソフトウェア開発見積りガイドブック ～*IT* ユーザとベンダにおける定量的見積りの実現～, オーム社.

[16] 独立行政法人情報処理推進機構: ソフトウェア開発分析データ集 2022, 2023-7-1. `https://www.ipa.go.jp/digital/chousa/metrics/metrics2022.html`,（参照 2023-8-10）.