Vol.2017-SE-195 No.31
2017/3/13

継続的リリースを前提とした要求の優先順位付けのための軽量見積り手法

森崎 修司[1]　小川 健太郎[2]　山口 鉄平[2]　高口 鉄平[3]

概要：継続的リリースを前提としたソフトウェア開発において，変更要求を実現する優先順位付けのための見積り手法を提案する．提案手法では，ソフトウェア利用者の観点，非機能要求実現の難易度，永続データの変更の有無，画面の変更の有無，回帰テストのテスト項目検討の難易度，変更の見通しがつきやすいかどうか，から変更コストと変更リスクを見積もる．これらと変更要求を実現したときに得られる価値を比較することにより，実現する変更要求の優先順位をつけることができる．商用サービスを対象として，提案手法の一部を用いて 10 件の変更要求の変更コスト，変更リスク見積りを試行した．試行により，提案手法によって変更要求ごとの見積りが可能であること，その際に必ずしもソースコードの行レベルで現行ソフトウェアを把握しておく必要がないことを確認した．また，見積りに要する時間は 2 時間程度であり，現実的な時間内で見積もることができた．

Shuji Morisaki[1]　Kentaro Ogawa[2]　Teppei Yamaguchi[2]　Teppei Koguchi[3]

1. はじめに

計画立案，開発作業の割当て，予算確保によってソフトウェア開発を円滑に進めるために，ソフトウェア開発コストの見積り手法が古くから研究されている [2]．これらの手法の多くはプロジェクト型の開発を前提としている．プロジェクト型の開発では実現する要求を一括して収集し，すべて開発した上でリリースする．プロジェクト型の開発では，開発計画に従って必要な開発要員に集まってもらい，開発作業を進める．開発が完了し，リリース後も開発要員が開発作業を継続することは前提となっていない．たとえば，組織の構成員が使う出張費，交通費精算システムを新たに構築する場合，精算に必要な機能を開発しリリースした後は運用が中心となり，システムの改良や機能追加を継続的に実施することは少なく，ユーザサポート，不具合の修正，ハードウェアの障害への対処が作業が中心となる．

近年，Web 検索エンジンや電子メールサービスをはじめとして，リリース後もサービスを運用しながら，ソフトウェアの改良，機能追加を継続するタイプのソフトウェアも増えてきている．このような開発では，リリース後の運用と並行して，開発を継続し，改良や機能追加を継続する．本稿ではこうした開発を継続的リリースを前提とした開発と呼ぶ．一般に，継続的リリースではプロジェクト型の開発と比較すると少数の要求を短い期間で実現し，リリースする．そのため，まだ実現していない要求の変更や新たな要求の追加が容易であり，利用者にとって価値の大きい要求から順番に実現し，リリースできる点で柔軟性が大きい．

継続的リリースを前提とする開発の計画立案や作業見積りにおいて，プロジェクト型の開発と同じような詳細なレベルのものをそのまま流用すると他の開発作業を圧迫する．継続的リリースを前提とした開発では，開発作業着手からリリースまでの期間がプロジェクト型の開発と比較すると短いからである．そのため，要求の変更や追加がある場合やリリース後の変更されたソフトウェアであっても，一度実施した要求の見積りを再利用できる，軽量な見積り手法を利用する，といった必要がある．リリースの頻度を大きくすると，実現により利用者にとっての価値が大きい要求を選ぶことができる．また，すべての機能を一括して開発するときのように実現の作業工数を精緻に見積もる必要は必ずしもない．

[1] 名古屋大学
Nagoya University, Furo, Chikusa, Nagoya, Aichi 464–8601, Japan
[2] ヤフー株式会社
Yahoo Japan Corporation, Kioi, Chiyoda, Tokyo, 102–8282, Japan
[3] 静岡大学
Shizuoka University, Johoku, Naka, Hamamatsu, Shizuoka, 432–8011, Japan

これまで，要求を実現するための開発コストと要求を実現したときに利用者が得られるメリットを勘案し，なるべく小さい開発コストで大きな価値を得ることを目的としてコストベネフィット分析をはじめとした，実現する要求の優先順位を決める手法 [4] が提案されている．また，AHP(Analytic Hierarchy Process) を活用して，要求の候補の対を選んで実現したときに得られる価値が大きいと考えられるほうを回答することにより全ての要求の候補の優先順位を決める手法が提案されている [1]．しかし，これらの手法は，リリース後に，実現する要求の候補が追加されたり変更されたりしたときに再度見積りを実施しなければならない．また，継続的リリースを前提とする開発を想定し，リリース後に開発作業や開発コストを見積もる手法が提案されている [3]．しかし，継続的リリースを前提とした開発では開発コストを詳細に見積もる必要はなく，手順には軽量化の余地がある．

本稿では，継続的リリースを前提として，要求の優先順位を付けるための手法を提案する．提案手法では，実現する要求の候補を列挙し，それぞれを実現するための変更コスト，変更リスクを簡潔な質問に回答することにより，明らかにする．また，それぞれの要求を実現したときに得られる価値と変更コスト，変更リスクを比較して，より小さいコストでより大きな価値を得られる要求を優先順位付けする．要求の変更コスト，変更リスクや価値は要求間の相対的な評価ではないため，新たに要求が追加された場合やリリースによって変更されたソフトウェアを対象としても再見積りをする必要は必ずしもない．

2. 対象とするソフトウェアと開発の形態

提案手法では，あるバージョンのソフトウェアを拡張，変更して次のバージョンのソフトウェアとしてリリースするサイクルを繰り返すことを前提としている．リリースしたソフトウェアの変更は利用者に即座に反映されるものとする．サイクルのはじまりでは，そのサイクルで実現する要求の優先順位を決める．優先順位が上位の要求から順番に実現することとする．優先順位を決める時点では詳細な開発作業やその作業に必要な工数を見積もる必要はないものとする．

開発にはソフトウェアの開発計画を立てる企画メンバと計画に沿いながら実際にドキュメントやソースコードを作成，変更する開発メンバが携わるものとする．企画メンバはソースコードの行単位で詳細に把握しているわけではないが，システムやソフトウェアの技術的な構成要素，データや制御の流れ，マニュアルや利用規約といったユーザへの提供情報，大まかな開発コストを把握していることを前提とする．また，企画メンバはソフトウェア開発費の収支のバランスをとる役割を担う．ソフトウェアの価値や収益の構造を理解しているものとする．

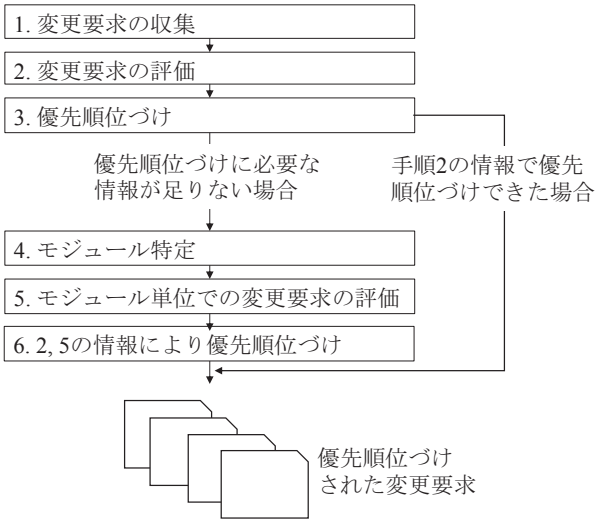

図 1 提案手法の手順

3. 提案手法

3.1 概要

提案手法の手順を**図 1** に示す．提案手法は 6 つの手順から成る．図中の “2. 変更要求の評価” で変更要求の実現コストと実現リスクを見積もる．変更要求ごとに見積もった実現コストと実現リスク，得られる価値を比較して，変更要求を実現する優先順位を決める．図中の “2. 変更要求の評価” の結果だけでは判断が難しい場合には，手順 4 で変更要求を実現するモジュールを特定した上でモジュール毎の実現コストと実現リスクを評価し (手順 5)，“2. 変更要求の評価” で見積もった実現コストと実現リスクを加味して優先順位を付ける (手順 6)．

3.2 手順

手順 1．変更要求の収集

変更要求を集める．新たな機能の追加，既存機能の変更，性能改善，不具合の修正といった利用者が使う部分を変更する内容に加え，ミドルウェアやオペレーティングシステムのアップデート，運用/運営コストを下げるための機能や仕組みの追加や変更も含める．変更要求は，表 1 に示す基本的な情報により定義する．

手順 2．変更要求の評価

手順 1 で集めた変更要求に対し，**表 2** に示した内容で変更コストと変更リスクを評価する．表 2 の各項目の詳細は次節で述べる．各項目は変更の有無により変更コストと変更リスクを評価することを目的としている．変更がある場合に規模がわかれば評価を精緻化できる．“利用者数”，“項目数”，“画面数” といった規模を推測できる値がわかる場合には，記入する．

手順 3．優先順位付け

手順 2 で評価した変更要求を実現する際の変更コス

表 1　変更要求の基本的な情報

項目名称	値	説明
ID	一意の番号	変更要求を区別するための番号
変更名称	自由記述	変更要求の名称
変更の説明	自由記述	どのような変更要求であるかを説明する文章
変更箇所	自由記述	どこを変更しなければならないかを説明する文章
起案日	変更要求の起案日	変更要求が起案された日
発生源	役割，組織名，担当者名	変更要求がどの立場から起案されたかを示す．企画メンバ，システム開発メンバといった形式で記録する．
分類	機能追加/機能変更/保守/運用/不具合修正	変更のタイプを示す．

表 2　手順 2 における評価項目

分類	項目	値	説明
管理情報	ID		表 1 の ID
利用者	利用規約の変更	有 (利用者数)/無/要調査	利用する際に利用者に同意してもらう必要のある利用規約に変更があるかどうか評価する．ある場合には利用者数の概数により，どのくらいの規模になるかを評価する．
	ユーザマニュアルやガイドの変更	有 (利用者数)/無/要調査	利用者向けのマニュアル，説明を変更する必要があるかどうかを評価する．変更がある場合には，利用者数の概数を規模の参考とする．
	リリースに伴うサービス停止	有 (利用者数と時間)/無/要調査	リリースする際に利用停止時間があるかどうかを評価する．ある場合には，どのくらいの利用者に対してどのくらいの停止時間になるかを規模の参考とする．
非機能要求	非機能要求実現の難易度	高/低/該当なし/要調査	変更部分に該当する非機能要求実現の難しさを評価する．
永続データの変更	データモデルの変更，追加，削除	有 (項目数)/無/要調査	永続データのデータモデルに変更，追加，削除があるかどうかを評価する．他のソフトウェアや同一ソフトウェアの他の機能で共通に利用する等変更による波及範囲も加味する．
	データ移行	有 (項目数)/無/要調査	データモデルの変更がある場合に，現行バージョンの永続データを新バージョンに合わせて変換する必要があるかどうかを評価する．
画面の変更	画面内の変更	有 (画面数)/無/要調査	同一画面内での変更を評価する．変更がある場合には，画面数を規模の参考とする．
	画面遷移の変更，追加，削除	有 (遷移数)/無/要調査	画面が遷移する順序の変更，新たな画面の追加，削除を評価する．変更がある場合には，遷移数を規模の参考とする．
	共通部分の変更	有/無/要調査	画面のデザインや画面内で HTML, CSS 等共通して定義している部分を変更しなければならないかを評価する．
回帰テスト	回帰テストの精査の有無	有 (規模)/無/要調査	開発メンバ以外のメンバでのテストが必要になるかどうか評価する．必要がある場合には，どのくらいの規模になるかを評価する．
変更の見通し	事前調査の必要の有無	有/無	変更する部分の予測がつきやすいかどうか評価する．依存関係が複雑な場合には，詳細な現状調査をしてからでないと変更できない場合がある．そのときには，調査が必要とする．
価値	変更要求により実現する価値	自由記述	変更要求を実現することにより得られる利用者，開発者，運営上の価値を評価する．上述の変更コスト，変更リスクと得られる価値を比較して，実現の優先順位を決める．

ト，変更リスクと変更要求を実現したときに得られる価値を比較して，変更要求の優先順位をつける．変更コスト，変更リスクが小さく，得られる価値が大きいものが優先順位が高くなり，変更コスト，変更リスクが大きく，得られる価値が小さいものが優先順位が小さくなる．変更コスト，変更リスクが不明なものがあり，優先順位が決められない場合には，手順 4 に進む．変更要求の優先順位が決まれば，手順 4, 5, 6 は実施しない．

手順 4．モジュール特定

個々の変更要求を実現するためのモジュールを特定し，より詳細な変更コスト，変更リスクを評価する．具体的には，変更要求を実現するための設計書の章節，ソースコードモジュールといった単位で構成要素を挙げる．

手順 5．モジュール単位での変更要求の評価

手順 4 で挙げたモジュール単位で**表 3** の項目を評価する．各項目の詳細は後述する．

表 3　手順 5 における評価項目

項目	値	説明
変更要求 ID	表 1 の ID	
変更モジュール ID	手順 4 で明らかにしたモジュールの ID	変更要求 ID の実現に必要なモジュールを一意に識別する ID
変更・追加規模	行数，ページ数をはじめとする見積り値	ソースコードや設計ドキュメントの追加，変更の規模を見積もる．
現行コードとの整合性	整合性を考慮した上で変更する規模の割合	変更・追加規模のうち，現行ソースコード，現行ライブラリとの整合性を考慮した上で変更しなければならない規模の割合
テストコードの有無	変更するモジュールのテストコードの有無	変更するモジュールのテストが自動化できているかどうかを示す．

手順 6. モジュール単位での変更要求の評価

手順 2, 手順 5 の評価項目により，手順 3 と同様に優先順位付けする．

3.3 手順 2 の評価項目

表 2 に示した評価項目の詳細を述べる．

3.3.1 利用者の観点からみた変更コスト，変更リスク

変更要求の実現において，利用者に影響を与える観点から，変更コスト，変更リスクを明らかにする．具体的には，利用規約，ユーザマニュアルやガイドの変更，及び，リリースに伴うサービス停止である．

変更がある場合には “有” とし，影響を受ける利用者数を見積もることができる場合には利用者数を併記する．変更がない場合には “無” とする．調査をしてみないと影響があるかどうかがわからない場合には “要調査” とする．

利用規約の変更，ユーザマニュアルやガイドの変更，リリースに伴うサービス停止は，利用者の不便につながる．また，利用者へのアナウンスや問い合わせ対応をはじめとして，運営にもコストが必要になる．そのため，これらが “有” の場合には，変更コストや変更リスクが大きいと判断する．

3.3.2 非機能要求の観点からみた変更コスト，変更リスク

変更要求を実現する際に考慮しなければならない非機能要求の観点から，変更コスト，変更リスクを明らかにする．具体的には，変更箇所が現行の非機能要求を満たす必要のある部分かどうかで判断する．また，変更要求が新たな非機能要求を追加する場合や現行の非機能要求を変更する場合も同様である．たとえば，現行の非機能要求として，利用者が機能 X を実行後，実行結果が表示されるまでの応答時間の上限値を 3 秒と定義している場合に，機能 X の応答時間に影響するような箇所を変更する場合である．

変更後の非機能要求を実現するために配慮が必要な場合には，変更コストや変更リスクが大きくなる．変更要求を実現する際に考慮しなければならない非機能要求があり，その実現が難しい場合には “高” を，考慮しなければならない非機能要求があり，その実現が難しくない場合には “低” とする．該当する非機能要求がない場合には “無” とする．調査してみないとわからない場合には “要調査” とする．

3.3.3 永続データの観点からみた変更コスト，変更リスク

変更要求を実現する際にデータベースやファイルに記録する永続データを変更する必要があるかどうかという観点から，変更コスト，変更リスクを明らかにする．具体的には，データモデルを変更，追加，削除する必要があるかどうかを判断する．データモデルの変更がある場合には，変更要求を実現した後に既存データを変換する等して移行する必要があるかどうかも評価する．たとえば，現行では金額は日本円としてのみ記録できる場合に，海外の通貨でも記録できるように変更する場合である．この場合，データモデルとして通貨の名称が新しく加わる．

変更後に永続データの変更，追加，削除が必要な場合には，変更コストや変更リスクが大きくなる．また，変更，追加，削除がある場合に，既存のデータの変換をはじめとした移行が必要な場合には，さらに変更コストや変更リスクが大きくなる．永続データの変更，追加，削除がある場合には，“永続データの変更，追加，削除” を “有” に，ない場合には “無” に，調査をしてみないとわからない場合には “要調査” とする．たとえば，先述した通貨の名称を加える場合，現行のデータには通貨の名称が記録されていないので，既存のデータの通貨の名称として日本円を補完しておく必要がある．

3.3.4 画面の観点からみた変更コスト，変更リスク

変更要求を実現する際に画面や画面遷移を変更する必要があるかどうか，変更がある場合に，他の画面との整合性を考慮する必要があるかという観点から変更コスト，変更リスクを明らかにする．変更を画面内の変更と画面遷移の追加，変更，削除に分けて変更コストや変更リスクを評価する．新たに画面を追加する場合は，画面遷移の追加にあたる．画面内の変更，画面遷移の追加，変更，削除のいずれかまたは両方に該当する場合，その変更が他の画面とのデザイン，HTML, CSS といった共通部分の変更を要するかどうかを併せて評価する．

変更後に画面内の変更や画面遷移の追加，変更，削除があると変更コストや変更リスクが大きくなる．また，変更の際に他の画面との整合性や一貫性を考慮しなければならないときは，変更コストや変更リスクがより大きくなることが多い．現行の画面の変更が必要な場合には，“画面内の変更” を “有” にする．画面数があらかじめわかる場合

には，その数も記入して変更コストや変更リスクの推測に役立てる．現行画面の変更がない場合には “無” に，調査してみないとわからない場合には，“要調査” とする．画面の追加，画面遷移の変更，削除がある場合には，“画面遷移の変更，追加，削除” を “有” とし，変更する遷移の数がわかる場合には，その数も記入する．画面の追加，画面遷移の変更，削除がない場合には “無” に，調査してみないとわからない場合には，“要調査” とする．また，“画面内の変更”，または，“画面遷移の変更，追加，削除” が “有” の場合には，その変更が他の画面との共通部分の変更を伴うかどうかも評価する．共通部分の変更が必要な場合には “有”，ない場合には “無”，調査をしてみないとわからない場合には “要調査” とする．

3.3.5 回帰テストの観点からみた変更コスト，変更リスク

変更要求を実現する際に大規模な回帰テストを実施する必要があるかどうか，実施する必要がある場合，テスト項目を決めるために開発メンバ以外のメンバも必要になるかどうかという観点から変更コスト，変更リスクを明らかにする．開発メンバ以外のメンバによる回帰テストの検討が必要かどうかは，設計，ソースコードといった開発メンバが変更するものだけで回帰テストのテスト項目を決められるどうかで判断する．

変更要求を実現する際に，変更していない部分に変更の副作用がないかを確認する回帰テストが必要な場合には，変更コストや変更リスクが大きくなる．特にどのような回帰テストを実施するかを決めるために開発メンバ以外のメンバも含めて検討しなければならない場合には，回帰テストの規模が大きくなる場合がある．開発メンバ以外のメンバも含めて，回帰テストを精査しなければならない場合には，“回帰テスト” を “有” とする．回帰テストの規模が大まかに予測できるときには，テスト項目の実施件数，実施に必要な工数といった規模も記入する．回帰テストの検討に開発メンバ以外のメンバが必要ない場合には “無” に，調査してみないと必要かどうかがわからない場合には “要調査” とする．

3.3.6 変更の見通しの観点からみた変更コスト，変更リスク

変更要求を実現するために変更が必要になる箇所や変更作業の想像がつきやすいかどうかという観点から変更コスト，変更リスクを明らかにする．複雑な部分や他の部分への依存や他の部分からの依存が大きい場所は事前調査や現行の整理等の作業が必要になり，変更の見通しがつきにくいと見なす．

変更要求を実現する際に，事前調査により整理する必要がある場合には，“変更の見通し” を “有”，ない場合には，“無” とする．

3.3.7 変更要求を実現したときに得られる価値

変更要求を実現したときに得られる価値を明らかにする．利用者，開発者，運営者，サポート担当者といった関係者を示して，その関係者の満足度を示す．変更コストや変更リスクを加味して，どの変更要求を実現する優先順位が大きいかを判断する．

3.4 手順 5 の評価項目

表 3 に示した評価項目の詳細を述べる．手順 4 において変更要求を実現する際に現行ソフトウェアのどの部分を変更するかをモジュールのレベルで調査し，列挙できていることを前提としている．本評価項目により，変更要求を実現するために各モジュールにどのような変更が必要かを明らかにする．

列挙したモジュールを識別できるよう ID をつける．提案手法ではモジュールの単位に前提は置かないが，たとえば，モジュールをソースコードファイルとすれば，変更要求を実現する際に，どのファイルを変更しなければならないかを手順 4 で列挙しておく．ファイル名が重複しなければ，モジュールの ID はファイル名とする．モジュールは設計書での機能定義，ソースコードでのクラス，メソッド，関数といった単位とできる．

- モジュール毎の変更・追加規模の観点からみた変更コスト，変更リスク
  モジュール毎に変更規模，追加規模を見積もる．削除の場合，削除する部分が他から参照されており，変更の必要があれば，参照している部分を変更規模とする．変更規模や追加規模が大きいほど変更コスト，変更リスクは大きくなると考える．
- 現行コードとの整合性の観点からみた変更コスト，変更リスク
  見積もったモジュール毎の変更規模，追加規模と，そのうち現行ソースコード，利用しているライブラリとの整合性を考慮しながら変更する必要のある規模の割合を調べる．整合性を考慮しながら変更する必要のある割合が大きいほど，変更コスト，変更リスクは大きくなると考える．
- テストコードの有無の観点からみた変更リスク
  変更するモジュールのテストを実行するテストコード，回帰テストが必要になるモジュールのテストを実行するテストコードが存在すれば，変更リスクを小さくすることができる．変更後のモジュールに副作用がないかどうかを確認するためのテストコードが存在すれば “有” に，存在しなければ “無” とする．

4. 試行

4.1 概要

提案手法を実際に実行し，軽量な見積りが可能かどうかを確かめることを目的として，インターネットを通じて提供されるサービスの変更要求を対象に手順 1 の変更要求の

表 4　試行対象の基本的な情報

ID	変更名称	変更の説明	変更箇所	発生源	分類
1	実行可能タスク一発検索	実行できるタスクの検索機能を改善する．	フロントエンド (利用者)	企画	新規追加
2	最大獲得ポイント表示	獲得できるポイントの最大値を表示する．	フロントエンド (利用者)	企画	新規追加
3	お好みタスクお知らせメール	指定した条件のタスクが公開されたときにメールで通知する．	フロントエンド (ユーザサポート)，バックエンド，バッチ	企画	新規追加
4	タスク期間変更機能	ユーザサポート画面からタスクの公開期間を変更できるようにする．	フロントエンド (ユーザサポート)	企画	新規追加
5	一覧非表示機能	ユーザサポート画面からタスク一覧画面よりタスクを非表示にする．	フロントエンド (ユーザサポート)	企画	新規追加
6	社外問い合わせ窓口変更対応	社外からの問い合わせ窓口を変更する．	フロントエンド (ユーザサポート)	企画	変更
7	共通データベース保守作業	共通データベースの保守作業を実施する．	バックエンド	開発	変更
8	サービス共通の通信方式変更	サービス間で共通の通信方式を変更する．	バックエンド	開発	変更
9	ヘルプページリンク変更	利用者むけのヘルプページへのリンクを更新する．	フロントエンド (利用者，ユーザサポート)	開発	変更
10	デバイス切り替えリンク変更	閲覧デバイスを変更するためのリンクを変更する．	フロントエンド (ユーザサポート)	開発	変更

表 5　試行対象の基本的な情報

ID	利用規約の変更	マニュアル等の変更	リリースに伴うサービス停止	非機能要求実現の難易度	データモデルの変更等	画面内の変更	画面遷移の変更	共通部分の変更	回帰テストの精査	事前調査の有無
1	無	有	無	低 (タスク絞込みの性能)	無	有	無	有	有	有
2	無	有 (≤5 千)	無	無	無	有	無	有	有	有
3	無	有 (≤5 千)	有 (≤5 千)	高	有	有	有	有	有	有
4	無	無	無	無	無	有	有	無	有	無
5	無	無	無	無	無	有	有	無	有	無
6	無	無	無	無	要調査	有	無	無	有	有
7	無	無	無	無	無	無	無	無	有	有
8	無	無	無	無	無	無	無	無	有	有
9	無	無	無	無	無	無	無	無	有	有
10	無	無	無	無	無	無	無	無	有	有

収集，手順 2 の評価の一部を試行した．また，本試行では，変更コストと変更リスクを洗い出せるかどうかを確かめることを主な目的としたため，変更要求がもたらす価値は評価しなかった．文献 [5] をはじめとして，別の方法により変更要求ごとの価値評価ができたことを前提として，優先順位付けできるかどうかを確かめる．

試行の対象は Yahoo!クラウドソーシングである．変更要求の基本情報は**表 4** のとおりである．なお，起案日は除いている．手順 2 の評価のみで優先順位付けができることを前提とし，手順 4 以降は実施しないこととする．手順 1 は企画メンバ，開発メンバから得る．手順 2 の評価は，本来，起案者が実施し関係者でレビューしながら合意するが，本試行においては，本稿の著者の 1 人である企画メンバが実施する．

4.2　結果と考察

試行の結果は次のとおりである．**表 5** のような評価結果を得た．作成には 2 時間を要した．変更要求の実現方法をソースコードの各行のレベルでわかっていない企画メンバが詳細な調査を実施せずに評価できた．

企画メンバから起案された ID1 6 の変更要求の評価では評価結果の “有” の数の合計が 3, 5, 8 となった．開発メンバから起案された ID7 10 の変更要求の評価では評価結果の “有” の数の合計がすべて 2 となった．

また，価値を見積もって実際に変更要求を優先順位付けできる可能性が大きいことを確認した．

考察は次のとおりである．評価を実施した企画メンバから評価作業において特に困る内容はなかったという意見を得た．手順 2 までの提案手法が妥当であることを示す結果が得られたといえる．今回は価値評価の基準設定に時間を要すると考えたため，手順 3 を実施しなかった．しかし，

ID1〜6 のように変更要求の評価結果の “有” の数が異なっているため，仮に個々の変更要求を実現することにより得られる価値に大きく違いがない場合でも優先順位をつけることができる．一方で，ID7〜10 の変更要求のように評価結果が似通った変更要求において，実現することにより得られる価値が似通っている場合，手順 4, 5 の実施が必要になる．

また，評価結果は優先順位付けの判断基準になるだけでなく判断のエビデンスにもなる，企画メンバと開発メンバで同じ視点から俯瞰的に開発案件を共有できる，サービス停止や利用規約の改定といった企画時点で考慮すべきことを事前に確認できる，という意見を得た．

本稿で示した提案手法の手順は開発サイクルの冒頭で実施することを前提としている．つまり，ある変更要求を実現しサイクルが終わり，次のサイクルの冒頭で実現する変更要求の優先順位を決める際に提案手法を使う．しかし，開発サイクルの途中で変更要求を実現している途中に，変更要求が突発的に追加された場合でも，再度提案手法で優先順位付けをすることができる．

価値の評価基準の設定は今後の課題であるが，筆者らの研究グループで提案する手法 [5], [6] が考えられる．

5. おわりに

継続的リリースを前提としたソフトウェア開発において，変更要求を実現する優先順位付けのための見積り手法を提案した．提案手法は，実現する変更要求の候補が複数ある場合に，どの変更要求を優先して実現すればよいかを詳細な見積りを実施せずに決めることを目指している．提案手法では見積りを 2 段階に分け，1 段階目では，変更要求を実現するモジュール (ソフトウェアの構成要素) を列挙しなくても，実施できるようにした．1 段階目で優先順位をつけられないときにのみ，変更が必要になるモジュールを列挙し，詳細な見積りを実施する．1 段階目では，利用者の観点，非機能要求実現の難易度，永続データの変更の有無，画面の変更の有無，回帰テストの検討の難易度，変更の見通しがつきやすいかどうか，から変更コストと変更リスクの大小を変更要求間で比較できるようにする．変更要求ごとにその変更要求を実現したときに得られる価値と変更コスト，変更リスクを比較すれば，どの変更要求がより大きな価値をより小さな変更コストと変更リスクによって実現できるかがわかり，実現する変更要求の優先順位付けができる．

商用サービスとして運営している Yahoo!クラウドソーシングを対象として，提案手法の 1 段階目の見積りの一部を試行した．具体的には，10 件の変更要求を収集し，それぞれの変更コスト，変更リスクを提案手法により見積もった．試行では，提案手法によって変更要求ごとの見積りが可能であること，その際に必ずしもソースコードの行レベルで現行ソフトウェアを把握しておく必要がないことがわかった．また，見積りに要する時間は 2 時間程度であり，試行したサービスにおいては，比較的短時間で済むことがわかった．今後の課題は，提案手法により変更要求に優先順位を付けること，他の見積り手法と比較することである．

謝辞 本研究の推進にあたって多大な支援をいただいたヤフー株式会社 石垣 光香子氏，柴田 宏行氏，八木 智章氏に感謝する．

参考文献

[1] M. Akker, S. Brinkkemper, G. Diepen, and J. Versendaal, Software Product Release Planning through Optimization and What-if Analysis, Jounral of Information Software Technology, vol.50, vol.1-2, pp.101-111(2008)
[2] B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Selby, ”Cost Models for Future Software Life Cycle Processes: COCOMO 2.0, Journal of Annalysis of Software Engineering, vol.1, no.1, pp.57-94(1995)
[3] D. Greer, and G. Ruhe, Software Release Planning: An Evolutionary and Iterative approach, Journal of Information and Software Technology, vol.46, no.4, pp.243-253(2004)
[4] J. Karlsson, Software Requirements Prioritizing, In Proc. of the Second International Conference on Requirements Engineering, pp.110-116(1996)
[5] 高口 鉄平，森崎 修司，小川 健太郎，石垣 光香子，クラウドソーシングサービスにおける機能価値の評価，2016 年度秋期情報通信学会大会予稿集，pp. (2016)
[6] 森崎 修司，高口 鉄平，永瀬 美穂，大平 雅雄，ユースケースの多面的評価によるソフトウェアモジュールの価値計測，ソフトウェアエンジニアリングシンポジウム 2015 論文集，pp.160-165(2015)