オブジェクト指向プログラム開発における機能変更に伴う更新作業工数の見積り

水野 修，伊登 友美，上原 智，菊野 亨
大阪大学 大学院基礎工学研究科 情報数理系専攻
E-mail: o-mizuno@ics.es.osaka-u.ac.jp

要旨

本研究ではオブジェクト指向開発における工数見積りの方法についての提案を行う．作業工数はプログラムの規模と密接な関連を持つので，コード行数 (Lines of code: LOC) による見積り評価が行われることが多い．しかし，LOC を適用することの問題点は従来より指摘されており，オブジェクト指向ソフトウェアにおいてもそれが当てはまる．

提案法の特徴はオブジェクト指向プログラムの特性を考慮して仕様変更作業の工数見積りを行うことにある．本方法では，仕様変更作業開始前に作業内容を推定し，細分化したそれぞれの作業について重み付きの得点を集計することで作業量を求める．具体的な開発事例データを用いた適用例についても述べる．

1 提案する見積り法

対象とするのはオブジェクト指向パラダイムに基づいた設計開発である．これは実際にある企業において行われたプロジェクトであり，開発言語には C++が使用され，その開発プロセスにはプロトタイピングが採用されている (図 1)．

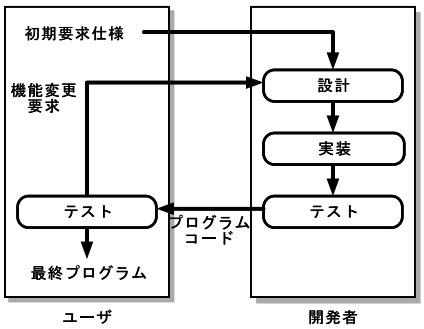

図 1: 開発プロセス

この開発プロセスでは，図1に示すようにユーザからの変更要求が頻繁に発生している．そうした仕様変更に対して迅速に作業量の見積りを行うことが必要となる．対象とする仕様変更はバグの修正ではなく，主としてユーザから寄せられた機能の変更要求への対応作業を指すものとする．

提案する見積りの流れを図2に示す．提案法は大きく2つのフェーズに分かれている．まずフェーズ1では，ユーザからの機能変更要求とその時点で作成されているドキュメントを入力とし，更新作業内容を推定する．次にフェーズ2では，更新作業内容の推定結果と作業毎の素点表を基に，各作業に対する得点を集計して，変更に要する作業量を算出する．以降では主にフェーズ2における重み表作成とその評価について述べる．

2 作業工数推定の枠組み

2.1 重み表の設計

細分化した作業に対して素点を与えるという立場は，すでに提案されているファンクションポイント法 [1] などと基本的に同じである．しかし，本研究でのアプローチの特色は，対象とするオブジェクト指向プログラムの特性を考慮に入れて得点を与える点にある．

オブジェクト指向プログラムにおける本質的な変更として次のものを考える．

- クラスの追加・削除
- クラスの変更
  - メンバ変数の追加・削除
  - メソッドの追加・削除
  - メソッドの変更

これらの各項目に対して，さらに細かい属性付けを行い，各作業にかかる労力を考慮した得点を与える．

まず，できるだけ作業内容を細分化する．それが得点表の形式を決めることになる．今回の試みでは，プログラム中の変数 (局所変数やメンバ変数) に着目している．一般的に1つの変数を新たに追加したり，削除したりする場合，その変数の有効範囲内での修正作業も必要になる．しかも，それに要する労力は変数の性質によって大きく変わる．そこで，ここでは追加・削除する変数の属性や型の情報を利用する．

まず，変数の型に関して次の3つの分類を行う．

基本型 int，charなどのような基本的な型を指す．

既製クラス ライブラリから呼び出して使用するクラスを表す．プログラミング言語のパッケージに含まれているものが相当する．

新設クラス プログラムの作成に伴って新しく定義されたクラスを指す．

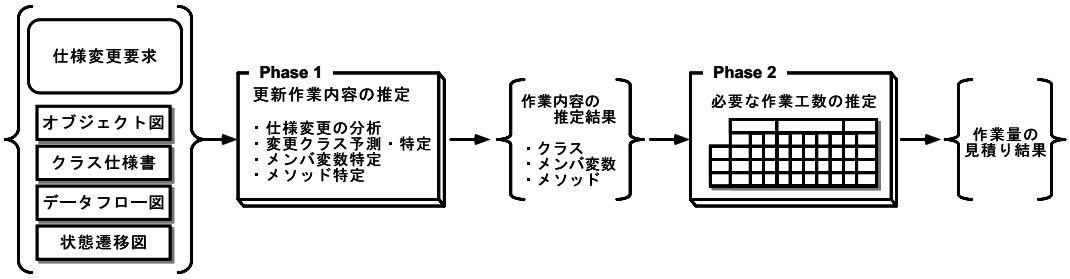

図 2: 提案法の流れ

次に，クラス中のメンバ変数やメソッドの属性として，private，protected，publicの3つを考える．これは，今回の研究では対象となるプログラムをC++ に限定しているためである [4].

2.2 具体的な重みの決定

前節で決定した重み表の各項目に対して具体的な値を定める．値の大小関係については，労力のかかる作業により大きな値を割り当てることにする．

この値の決定に当たっては，協力企業の開発者A, B, Cの3名にアンケート調査を行った．その結果，決定した重み表の一部を表1に示す．(現時点では重みの具体的な値そのものに対する理論的根拠は無い．実験的データを利用した妥当性評価は今後の課題である．)

表 1: 重み表 (一部)

	属性	メンバ変数の追加			メソッドの追加							メソッドの変更			
					引数		返り値		局所変数				局所変数の追加		
		基本型	既製クラス	新設クラス	参照呼出	値呼出	あり	なし	基本型	既製クラス	新設クラス	基本点	基本型	既製クラス	新設クラス
開発者A	private	1	1.2	1.5	1.5	1	1	0	1	1.2	1.5	3	0.7	1	1.2
	protected	1	1.2	1.5	1.5	1	1	0	1	1.2	1.5	3	0.7	1	1.2
	public	1	1.2	1.5	1.5	1	1.5	0	1	1.2	1.5	5	0.7	1	1.2
開発者B	private	1	5	4	1.2	1	1.1	1	1	5	4	1	1	5	4
	protected	1.4	7	5.6	1.2	1	1.1	1	1	5	4	1.4	1	5	4
	public	3	15	12	1.2	1	1.1	1	1	5	4	3	1	5	4
開発者C	private	1	1	1.2	1	1.2	0.5	0	1	1	1.2	1.5	1.5	1.5	2
	protected	1	1	1.2	1	1.2	0.5	0	1	1	1.2	1.5	1.5	1.5	2
	public	1	1	1.2	1.2	1.5	0.8	0	1.5	1.5	2	1.5	1.5	1.5	3
今回の案	private	1	1.2	1.5	1.2	1	1.5	1	1	1.2	1.5	2	1	1.2	1.5
	protected	1	1.2	1.5	1.2	1	1.5	1	1	1.2	1.5	2	1	1.2	1.5
	public	1	1.2	1.5	1.2	1	1.5	1	1	1.2	1.5	3	1	1.2	1.5

3 適用実験

ここでは1つの実業務に使用されたアプリケーションプログラムへの適用実験について述べる．

3.1 実験データ

実験対象としたプログラムはある企業で過去に開発されたアプリケーションである．この開発において仕様変更は2回発生し，それぞれ M_1，M_2 と表す．この開発は仕様変更の過程が詳細に記録されているため，今回の実験に適している．

仕様変更 M_1 を加える直前の規模は約6.2KLOCで，最終的な規模は約8.9KLOCであった．また，最終的なプログラム中には34個のクラスが存在した．変更は主にユーザからの要求による新たな機能の追加であり，全体で16 項目について機能変更 $m_1, m_2, \cdots, m_{16}$ が加えられている．このうち，$m_1, \cdots, m_{10}$ が変更 M_1 で加えられ，$m_{11}, \cdots, m_{16}$ が変更 M_2 で加えられている．なお，M_1 に要した工数は1.3 人月，M_2 に要した工数は1.0 人月であった．

対象としているプロジェクトがすでに開発を終了しているので，今回の実験では「実際にソースコードに加えられた変更内容が作業内容の推定結果である」と見なしてフェーズ2を適用した．

まず，対象となるプログラムの変更前と後のソースコードから，変更された部分を差分として抽出する．次に，表1に示す作業内容に従って，34個のクラス $C_1, C_2, \cdots, C_{34}$ のそれぞれについて変更内容の詳細を把握する．なお，クラス $C_1, \cdots, C_{20}$ は仕様変更 M_1 の直前に存在したクラスであり，$C_{21}, \cdots, C_{34}$ はその後の M_1, M_2 によって追加されたクラスである．こうして導出した詳細な作業内容に基づいて，各クラスに対して必要な作業量の見積りをフェーズ2で行う．

3.2 見積り結果

前節で得られた作業内容の推定結果に基づいて，各クラスに対する変更作業量を計算する．今，クラス C_3 に対する機能追加 m_5 を例にとる．すると，

- protected 属性で int 型のメンバ変数の追加：1点
- protected 属性で CString 型のメンバ変数の追加：1.2点
- public 属性のメソッド追加：1.5点
- protected 属性のメソッドに対する変更：2点
- public 属性のメソッドに対する変更：3点

となり，m_1 に対しては $(1 + 1.2 + 1.5 + 2 + 3 =)8.7$ 点が作業量として計算される．同様にして，C_3 に加えられた変更全てについて作業量を計算すると，各機能変更についての作業量は $m_6 = 15.7, m_7 = 5.4, m_9 = 4.0$ となり，合計33.8点となる．各クラスについて，機能変更毎に作業量を計算した結果を表2に示す．

表 2: 機能変更毎の作業量

Class	仕様変更M1										仕様変更M2						合計
	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}	m_{12}	m_{13}	m_{14}	m_{15}	m_{16}	
C_1		6.0	13.2			22.3	21.3				9.2	26.3	23.1				77.8
C_2					21.3	15.7	5.4		4.0								64.9
C_3			2.4		8.7						1.5	11.4	12.6				33.8
C_4					4.0							6.6	6.2				31.9
C_5					13.3							2.0	2.0	2.0	2.0	1.0	12.8
C_6				2.0								4.5	4.5				14.3
C_7	10.7			2.0													8.0
C_8											10.1						11.0
C_9																	12.7
C_{10}																	10.1
C_{11}			9.4													9.9	9.9
C_{12}	10.6																9.4
C_{13}			3.5	2.0												1.0	10.6
C_{14}										5.8							6.5
C_{15}																	5.8
C_{16}									3.2								3.2
C_{17}				2.0												1.0	3.0
C_{18}																1.0	1.0
C_{19}											2.2						2.2
C_{20}						1.0	1.0	1.0									3.0
C_{21}						40.0											40.0
C_{22}							40.0										40.0
C_{23}										29.5							29.5
C_{24}						6.9											6.9
C_{25}					6.9												6.9
C_{26}							6.9										6.9
C_{27}						5.7											5.7
C_{28}								20.8									20.8
C_{29}										5.9							5.9
C_{30}											27.9						27.9
C_{31}			32.7														32.7
C_{32}												32.1					32.1
C_{33}												44.5					44.5
C_{34}													48.0				48.0
合計	21.3	6.0	61.2	8.0	54.2	91.6	74.6	21.8	7.2	41.2	50.9	127.4	96.4	2.0	2.0	13.9	

表 3: 難易度の順位

Class	順位	Class	順位
C_1	1	C_{11}	11
C_2	2	C_{12}	12
C_3	3	C_{13}	13
C_4	4	C_{14}	14
C_5	5	C_{15}	15
C_6	6	C_{16}	16
C_7	7	C_{17}	17
C_8	8	C_{18}	18
C_9	9	C_{19}	19
C_{10}	10	C_{20}	20

4 評価

4.1 クラス難易度との比較

提案法によって求められる作業量が，クラスの難易度を反映しているかを評価するために，作業者の判定に基づく各クラスの変更難易度との順位相関を調べる．

クラス難易度との比較では，変更前のコード中に存在しているクラスであって，かつ，仕様変更によって実際に変更が加えられた 20 のクラス $C_1, \cdots C_{20}$ を対象とする．そして実験の対象とする 20 個の全てのクラスに対して，コードに変更を加える場合に予想される手間を基準として順位をつけてもらった．結果を表3に示す．同表で C_1 が最も手間がかかり，C_{20} が最も簡単であると判断されている．

次に，表2と表3の各クラスの順位相関を調べた．その結果としての順位相関係数の値は 0.90 であった．従って，適用したデータに対しては今回の重み案が非常に高い相関を示していることが分かった．

4.2 作業工数との比較

次に，2回の仕様変更 M_1 と M_2 に要した実工数との比較を行う．表2より，仕様変更 M_1 に要した作業量の合計は 387.1，M_2 の作業量は 292.6 となり，その $M_1/M_2 = 1.323$ であった．一方，は先に述べたように M_1，M_2 の実工数の比は 1.3 であり，表2から求めた値 1.323 が非常に近いことがわかる．この結果から，提案法で求めた作業量は実工数の比をある程度保存しているということが確かめられた．

5 まとめ

本報告では，オブジェクト指向プログラムへの機能変更に必要となる労力の見積り法について議論した．具体的には，工数の見積りに必要な作業量の重みを与え，開発事例データを用いた適用実験も行った．その結果，極めて限定的ではあるが，提案した見積り法の妥当性を確認することができた．

今後の課題としては次のものが考えられる．

- クラスの削除，メンバ変数やメソッドの削除に要する作業量についても考察して，対象とする更新作業を広げること．
- 別のプロジェクトにおけるデータへの提案法の適用とその評価を行って，本提案法の一般化について検討すること．

謝辞

本研究の初期の段階で，有益なコメントを頂いた大阪大学大学院基礎工学研究科 楠本真二講師に感謝致します．

参考文献

[1] A.J. Albrecht and J.E. Gaffney, Jr., “Software function, source lines of code, and development effort prediction: A software science validation,” IEEE Trans. on Software Engineering, Vol. 9, No. 6, pp.639–648, 1983.

[2] G. Booch: *Object Oriented Analysis and Design With Applications*, The Benjamin / Cummings, 1994.

[3] S. R. Chidamber and C. F. Kemerer: “A metrics suite for object oriented design”, IEEE Transactions on Software Engineering, Vol.20, No.6, pp.476–493, 1994.

[4] B. Stroustrup: *The C++ Programming Language, 2nd edition*, Addison-Wesley, 1991.

[5] J. Rumbaugh: *Object-Oriented Modeling and Design*, Prentice Hall, 1991.