7E-3

プログラム設計過程の定量化に関する一考察(その2)
―複合設計法の採用による効果―

工藤 英男[*]・野村 研仁[**]・杉山 裕二[**]・鳥居 宏次[**]
(*大阪大学情報処理教育センター, **大阪大学基礎工学部)

1. まえがき

近年、ソフトウェア工学の分野において、ソフトウェア・メトリックスに関する研究が脚光を浴びているが、それらの多くは信頼性・生産性・工数見積などへの適用である。我々はメトリックスの考えをプログラムの設計過程に導入することについて関心を持っている[1]。

その一段階として、プログラミングにおける設計作業の内容を実験により定量的に見直し、同一の課題を何も設計法に関する知識を与えない状態で自己流にプログラムを作成した場合と、各種の設計法を教授した後のそれぞれの設計法に基づいてプログラムを作成した場合では、モジュールに分解する過程と最終ソースコードにおいて、どのような差異が認められるかを明らかにしたいと考えている。

本稿では、複合設計法を用いた事例で定量的に解析した結果について述べる。

2. 実験の概要

この実験は酒屋の在庫管理の問題[2,3]を共通課題として選び、コーディング時における入出力の負担を軽減させるためと、簡単にテストができるように入出力モジュールとテストデータを事前に与えた。

第1段階は、各人の考えに基づき実際にPascal言語を用いてプログラミングをさせ、最終ソースコードを提出させた。第2段階は複合設計法に関する資料[4](出庫依頼トランザクション処理の階層構造までを前提にして)を与え簡単な説明後、同様にその設計法に基づきプログラミングをさせ、最終ソースコードを提出させた。

なお、期間はそれぞれほぼ2週間であり、実験の被験者は構造化プログラミングやプログラム書法程度の講義と演習を終えている学部(情報工学科)の2年生である。

3. 設計法におけるモジュールに関する計量

本実験では、課題を実現するための使用言語にPascalを用いた関係上、モジュールの概念やその評価基準に関して、本来の複合設計法におけるそれらがそのまま適用できないため、ここで用いる用語や尺度の基準を示す。

まず、複合設計法におけるモジュールの概念の定義として、例えば、つぎのように記述されている[4]。

①複数ステートメントが語彙としてまとまっている。
②ステートメント群が境界識別子で区切られている。
③ステートメント群は名前(モジュール名)によってまとめて参照できる。

しかし、Pascalで多く見られる内部手続きは厳密な意味ではモジュールの概念に反するが、ここではモジュールとして取り扱う。

つぎに、複合設計法の特徴であるモジュール化をはかる際のモジュールの独立性を高める基準として、モジュール内部での関連性(モジュール強度)が最大になるようにすることと、他のモジュール間の関連性(モジュール間結合度)が最小になるようにすることが挙げられる。

そこで、複合設計法におけるモジュールの尺度や評価基準の定義と本実験で用いたそれらの解釈の違いについて以下に示す。

3.1 モジュール強度について

モジュール強度としては強度を示す尺度の高いものから、機能的, 情報的, 連絡的, 手順的, 時間的, 論理的及び暗合的強度の7タイプがある。

本実験では、情報的強度は使用言語の文法仕様からありえなく、連絡的強度は手順的強度との識別が困難であり、暗合的強度は今回解析したプログラムに出現しなかったという理由で除外した。

従って、モジュール強度の尺度として採用したのは、機能的, 手順的, 時間的及び論理的強度の4タイプである。それらの尺度を識別した主たる評価基準を示す。

機能的強度:単一機能の手続きや関数。
手順的強度:複数の機能を逐次的に処理する手続きや関数。
時間的強度:初期化などの手続き。
論理的強度:パラメタにより複数の機能から、実行すべき機能を選択する手続き。

3.2 モジュール間結合度について

モジュール間結合度としては結合力を示す尺度の低いものから、データ結合, スタンプ結合, 制御結合, 外部結合, 共通結合及び内容結合の6タイプがある。

本実験では外部結合も共通結合として考え、内容結合は使用言語の文法仕様からありえないという理由で除外した。

従って、モジュール間結合度の尺度として採用したのは、データ, スタンプ, 制御及び共通結合の4タイプである。それらの尺度を識別した主たる評価基準を示す。

データ結合:モジュール内で使用するデータをパラメタのみで渡している。
スタンプ結合:モジュール内で使用しないデータもパラメタで渡している。
制御結合:制御要素をパラメタとして渡している。
共通結合:モジュール内で使用するデータをパラメタ以外で渡しているのもある。

On the Metrics of the Design Process (Part-2)
Hideo KUDO, Kenji NOMURA, Yuji SUGIYAMA, Koji TORII
OSAKA Univ.

4．データの解析

複合設計法を用いた被験者１３名のうち６名の学生の最終ソースコードと自己流で作成した最終ソースコードを分析の対象として、それぞれのモジュールに分解する過程とモジュール構成に着目して解析を試みた。

なお、解析の対象から除外した７名分は、以下の作業工程により却下した。

①４種類のテストデータにより、プログラムの動作確認の結果においてバグが発見されたのが２名分。

②自己流で作成したプログラムと設計法によるプログラムとのソースコード比較において、類似度が高いものが３名分（複合設計法を用いて作成したとは考えられないもの）。

③複合設計法を用いて作成したと考えられるが、解析が困難なものが２名分。

今回、解析の対象とした被験者６名の作成法別に行数，実行速度，作業工数と、今回の分析要約であるモジュール強度及びモジュール間結合度の各タイプの数を表－１に示す。

これより、自己流と設計法との差異における全般的な傾向は、以下に示す４点に要約できる。

①行数及びモジュールの分割数では設計法の方が多くなっている。

②実行速度及び作業工数では両者に極端な差はない。

③モジュール強度においては、自己流では手順的強度が比較的多かったが、設計法ではより強度が高い機能的強度へ移行している。

④モジュール間結合度おいては、自己流では共通結合が比較的多かったが、設計法ではより結合力の低いデータ結合やスタンプ結合へ移行している（被験者3と38を除き）。

なお、②の作業工数で考えられる要因しては、設計法を理解するために時間を費したためと思われる。

5．考察

本実験では、Pascalでのモジュール実現に対する配慮が欠けていた点や与えた入出力モジュールが適切でなかったにもかかわらず、複合設計法に照らし合せてプログラムを作成することによる効果として、今回の少ない事例分析から得られた結果は、以下に示す３点に要約できる。

①モジュールに分解する過程を分析すると、階層構造図を作成する時点で各モジュール間のパラメタ受渡しが明確に記述されていると、実際のソースコードにそれが反映されて比較的よいモジュール間結合になっている。

②分割されたモジュールはモジュール内の機能の凝集度が高く、モジュールとモジュールとの間の結合が低くなるように構成される傾向が認められる。

③複合設計法の欠点として指摘されているように[4]、同一の資料をもとにプログラム作成を指示しても、被験者ごとにモジュール構造が異なっていることが認められる。

なお、③に関しては同一の課題で実験を行なったため、初めの自己流でプログラム作成をした時の考え方が影響しているのかもしれない。

6．あとがき

プログラムの安定性（プログラム変更余波）モデル[5]の適用を考えた場合、モジュール間の完全従属マトリックスを解かなくとも、自己流より設計法を用いた方がモジュール変更による他のモジュールへの変更波及が少なくなると推測される。

今後の課題の一つとして、モジュール間結合度を自動的に解析するツールやモジュール強度の解析を支援するツールさらにはモジュール間の完全従属マトリックスを解くプログラムを用意することが考えられる。

謝辞　今回の実験に協力してくれた被験者の学生諸君及びデータの解析等を援助していただいた鳥居研究室の川瀬　淳君に感謝する。

参考文献

1)工藤・杉山・鳥居：プログラム設計過程の定量化に関する一考察，情報処理学会第32回全国大会５Ｊ－２，pp765-766(1986).

2)山崎利治：共通問題によるプログラム設計技法解説，情報処理，Vol.25,No.9,p.934 (1984).

3)山崎利治：共通問題によるプログラム設計技法解説（その２），情報処理，Vol.25, No.11,p1219(1984).

4)久保未沙：複合設計，情報処理，Vol.25,No.9,pp935-945(1984).

5)久保・国友訳：高信頼性ソフトウェアー複合設計，近代科学社(1976).

表－１　６例のプロフィール

被験者番号	作成法	行数	実行速度(ms)				モジュール強度					モジュール間結合度					作業工数（時間）					
			data 1	data 2	data 3	data 4	分割数	機能的	手順的	時間的	論理的	パス数	データ	スタンプ	制御	共通	設計	コード化	エディット	デバッグ	テスト	合計
3	自己流	369	140	206	186	180	18	9	7	1	1	23	0	1	2	20	8	6	2	10	3	29
	設計法	409	120	196	176	154	23	14	7	1	1	29	1	1	2	25	5	5	6	10	3	29
9	自己流	332	130	198	172	148	9	2	5	1	1	9	0	2	2	5	4	0	3	8	2	17
	設計法	346	154	224	200	180	13	3	9	0	1	17	5	3	4	5	3	0	5	4	1	13
17	自己流	377	136	190	184	166	13	3	7	2	1	15	1	1	2	11	6	3	2	3	1	15
	設計法	403	126	210	188	170	20	9	9	1	1	23	12	2	2	7	3	3	3	4	1	14
18	自己流	357	90	168	136	124	13	5	6	1	1	14	1	1	2	11	8	1	3	8	3	23
	設計法	398	134	206	192	190	16	5	9	1	1	20	11	1	2	6	8	3	4	6	3	24
20	自己流	382	128	194	172	162	9	3	5	0	1	11	0	1	2	8	30	6	3	15	6	60
	設計法	364	128	202	170	158	16	9	6	0	1	18	11	3	2	2	25	3	1	6	1	36
38	自己流	376	136	228	180	192	12	6	5	0	1	13	10	1	2	0	8	5	3	6	4	26
	設計法	468	126	208	170	166	13	5	7	0	1	15	6	5	2	2	8	3	2	10	2	25