ソフトウェア工学 58－11
（１９８８．２．５）

# 実験に基づくプログラム設計過程の定量化の試み

工藤　英男*　杉山　裕二**　藤井　護**　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）

概要

これまで、ソフトウエア・プロジェクトの定量化を含む研究はよくなされているが、ソフトウェア製品あるいはソフトウェアのインプリメンテーション過程を対象としたものが多く、ソフトウェア設計過程を対象としたものは余りない。

本稿の目的は、ソフトウェアの設計過程を計量することにより、ソフトウェアの設計において何が重要な要因であるかを検証することである。ソフトウェアの設計過程とは、インプリメンテーションに先だつ仕様記述からコーディングにわたるソフトウェア開発の一連の段階を意味する。学生の被験者を用いた２つの実験を通して、仕様記述の質とプログラム作成者の努力が重要な要因あることを確認した。

# A NOTE ON QUANTIFYING AN EXPERIMENTAL DESIGN PROCESS

Hideo KUDO*, Yuji SUGIYAMA**, Mamoru FUJII** and Koji TORII*
*Education Center for Information Processin, Osaka University
**Faculty of Engineering Science, Osaka University
1-1 Machikaneyama, Toyonaka, Osaka 560, Japan

There have been many studies involving quantitative measurment of software projects whose objects have been to quantify either software products or software development process. Our goal in this paper is to identify important factors which quantify software design process by which we mean phase of software development which spans the specification and coding.

Through two experiments using student subjects, we have identified the quality of the specification and programmer effort as important factors.

## 1. はじめに

ソフトウェア工学において、ソフトウェア生産性や信頼性を高めるためのメトリックスの重要性がますます認識されてきている。ソフトウェア・メトリックスに関する研究は少なくないが、その多くはソフトウェア製品そのものあるいは、ソースコードに関したものである。中には、プログラミング工数とソフトウェア製品との関係を研究したものもあるが、それらの研究はインプリメンテーションに関するもので、仕様記述や設計段階に着目したものはない。これは主に仕様記述や設計書の質を計測することが困難であることに起因すると思われる。

筆者らは、品質向上における要因を究明するために、いくつかのソフトウェア設計の活動（例えば、要求定義、問題仕様記述あるいは詳細設計）においてどの段階が重要であるかを究明するための二つの実験を、学生を被験者として実施した。

学生の共通課題として、酒卸売販売会社の在庫管理問題[1,2]を採用し、学生に問題の要求仕様を与えて、この問題に対するプログラムを作らせた。実験に先立ち、学生にはコーディングの負担を軽減するための入出力ルーチンと、できあがったかどうかをチェックするためのいくつかのテスト・データを与えた。なお、この実験の対象学生は情報工学科２年生で、Ｐａｓｃａｌ、構造化プログラミングやプログラム書法[3]を習得済みである。

## 2. 実験１：初期設計書とソースコード

### 2.1 実験の目的

この実験は、学生が与えられた問題要求を的確に理解しているか否かを定量的に評価することが目的である。学生が問題要求を理解し、その仕様記述を正確にプログラミング時に生かし、プログラム書法の基本を理解して良いソースコードを作る能力をもつであろうと予想して、次の実験を行なった。

(1)各学生には、制限時間約７０分以内に課題を読ませ、ブロック図を用いてデータ構造や補足説明を記述した１ページの初期設計書を完成させた。

(2)各学生には、自分の初期設計書に基づき２週間でプログラムを作成させた。

(3)収集したデータには、各学生から提出されたレポートとプログラム開発工程を計算機で自動的に収集したものがあり、以下のものからなる。

a)初期設計書
b)学生から申告されたコーディング及び机上デバックに費やした作業時間の記録
c)計算機利用時間の記録
d)翻訳回数とプログラム実行回数の記録
e)完成後のソースリスト

### 2.2 データ収集と分析

初期設計書と各学生から最終結果として提出されたソースリスト（実行可能なソースコード）との関係を調べた。初期設計書の評価は困難な作業であり、設計を評価するための良いメトリックスはいまだ存在しない。この実験では、各々の設計書を３人が独立に評価した。

まず、初期設計書については、内容点，完成度，表現力，わかりやすさ等について５段階評価を行ない、それを参考に１００点満点で採点した。

つぎに、ソースコードについては、提出された最終プログラムをテストデータによる動作確認を行なわない状態で、モジュールの作り方，名前の付け方，理解のしやすさ，注釈の量や段付け等について同様に評価し採点した。

なお、各採点者の評価点を標準化するために偏差値を採用した（各採点者の評価点の標準偏差が10及び平均が50になるように変換した）。

学生ｉの設計書における採点者ｊによる点を$E_{i,j}$とすると、偏差値は次の式で得られる。なお、$\overline{E}_j$と$\sigma_j$は採点者ｊの平均値と標準偏差である。

$$Z_{i,j}=\frac{E_{i,j}-\overline{E}_j}{\sigma_j}\times 10+50$$

各学生の初期設計点は３人の採点者の偏差値の平均である。また、ソースコード点も同様の方法で算出した。初期設計・ソースコードの評価の結果，作業工数，各学生のソースコードにおける行数及びモジュール数を表１に示す。なお、学生数は３６名であった。

各学生の作業工数は2.1節に述べたb)とc)の合計である。表２より、これらの項目の間には顕著な相関は見られなかった。そこで、設計点

表１　実験１における計測

| ｸﾞﾙｰﾌﾟ | 学生番号 | 初期設計点 | 作業工数(時間) | ﾌﾟﾛｸﾞﾗﾑ・ｻｲｽﾞ(行) | ﾓｼﾞｭｰﾙ数 | ｿｰｽｺｰﾄﾞ点 |
|---|---|---|---|---|---|---|
| A | 23 | 70.0 | 36 | 458 | 15 | 62.1 |
| | 4 | 67.6 | 20 | 331 | 19 | 61.0 |
| | 40 | 62.0 | 22 | 393 | 13 | 51.8 |
| | 22 | 58.5 | 29 | 327 | 9 | 51.1 |
| | 28 | 57.5 | 29 | 297 | 11 | 46.7 |
| | 29 | 56.7 | 21 | 281 | 14 | 60.3 |
| | 5 | 56.0 | 27 | 375 | 15 | 54.3 |
| | 16 | 56.0 | 15 | 388 | 7 | 44.8 |
| | 11 | 55.6 | 34 | 467 | 17 | 65.1 |
| | 26 | 54.8 | 20 | 326 | 8 | 54.5 |
| B | 20 | 54.6 | 62 | 487 | 9 | 62.7 |
| | 31 | 54.5 | 29 | 528 | 18 | 47.8 |
| | 25 | 53.4 | 38 | 425 | 9 | 54.1 |
| | 33 | 52.8 | 45 | 417 | 16 | 58.5 |
| | 2 | 52.1 | 31 | 340 | 13 | 43.8 |
| | 38 | 52.0 | 33 | 378 | 12 | 46.1 |
| | 19 | 51.4 | 20 | 335 | 10 | 58.3 |
| | 9 | 47.9 | 23 | 341 | 9 | 41.9 |
| | 7 | 46.5 | 25 | 342 | 14 | 50.0 |
| | 18 | 44.9 | 21 | 357 | 13 | 58.6 |
| C | 6 | 44.7 | 39 | 285 | 9 | 43.7 |
| | 35 | 42.7 | 31 | 379 | 10 | 44.4 |
| | 12 | 40.6 | 21 | 414 | 12 | 35.1 |
| | 3 | 40.4 | 38 | 370 | 18 | 39.0 |
| | 10 | 40.4 | 28 | 301 | 8 | 45.8 |
| | 36 | 39.1 | 43 | 466 | 24 | 56.0 |
| | 13 | 38.5 | 19 | 323 | 6 | 51.5 |
| | 21 | 36.8 | 46 | 288 | 8 | 46.8 |
| | 17 | 35.4 | 27 | 429 | 13 | 56.5 |
| | 32 | 35.4 | 38 | 432 | 9 | 43.1 |
| D | 8 | 57.0 | 27 | 341 | 12 | 34.0 |
| | 41 | 54.7 | 19 | 363 | 8 | 54.8 |
| | 14 | 52.2 | 25 | 389 | 12 | 43.7 |
| | 34 | 48.6 | 33 | 297 | 3 | 36.5 |
| | 1 | 45.8 | 17 | 210 | 7 | 47.9 |
| | 37 | 43.3 | 25 | 304 | 7 | 47.9 |
| 平均 ｸﾞﾙｰﾌﾟA (標準偏差) | | 59.5 (5.3) | 25.3 (6.8) | 364 (63) | 12.8 (4.0) | 55.2 (6.8) |
| ｸﾞﾙｰﾌﾟB | | 51.0 (3.4) | 32.7 (12.9) | 395 (68) | 12.3 (3.1) | 52.2 (7.2) |
| ｸﾞﾙｰﾌﾟC | | 39.4 (3.0) | 33.0 (9.2) | 359 (66) | 11.7 (5.5) | 46.2 (6.9) |
| ｸﾞﾙｰﾌﾟD | | 50.3 (5.3) | 24.3 (5.8) | 317 (63) | 8.2 (3.4) | 44.1 (7.8) |
| 全体 | | 50.0 (8.6) | 29.3 (9.9) | 366 (68) | 11.5 (4.3) | 50.0 (8.1) |

表2．項目間の相関係数

| 項目 | 作業工数 | ﾌﾟﾛｸﾞﾗﾑ・ｻｲｽﾞ | ﾓｼﾞｭｰﾙ数 | ｿｰｽｺｰﾄﾞ点 |
|---|---|---|---|---|
| 初期設計点 | -0.136 | 0.121 | 0.200 | 0.379 |
| 作業工数 | | 0.414 | 0.178 | 0.121 |
| ﾌﾟﾛｸﾞﾗﾑ・ｻｲｽﾞ | | | 0.511 | 0.289 |
| ﾓｼﾞｭｰﾙ数 | | | | 0.345 |

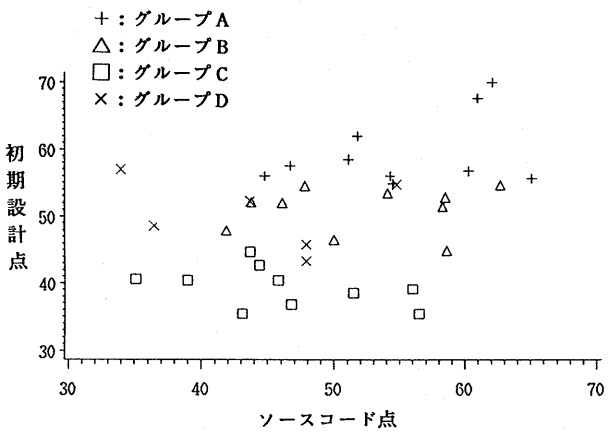


図１　初期設計点とソースコード点の散布図

の良いもの，普通のものおよび悪いもの（Ａ，ＢおよびＣ）と，設計点の善し悪しにかかわらず、提出されたプログラムが正しく動作しなかったバグ有りのもの（Ｄ）の４つのグループに分け、学生における個々の間の関係の代りにグループの間の関係を分析した。

図１に初期設計点と最終ソースコード点との関係を示す。

表１より、次のことが認められる。

1)ソースコード点の平均値は、グループＤを除き、初期設計点の平均値の順になっている。また、グループＤにおける初期設計点の平均値は50.3でありグループＢの51.0と非常に接近している。

2)作業工数の平均値は、グループＤを除き、初期設計点の平均値の逆順になっている。

3)プログラム・サイズの平均値は、グループＤが 317行と他の３つグループに比較してより少ない。

4)モジュール数の平均値は、グループＤが 8.2と他の３つグループに比較してより少ない。

各項目において平均値でみると、グループ間に相違がみられた。そこで、統計的にグループ間の有意差に関する検定を分散分析法を用いて行なった。分散分析法としては、一元配置繰り返しありのモデル[7]を採用し、次の帰無仮説について検定した。

H11: 設計点の善し悪しにかかわらず、各グループのソースコード点は同じである。

H12: 設計点の善し悪しにかかわらず、各グループの作業工数は同じである。

H13: プログラムにおけるバグの有無にかかわらず、プログラム・サイズは同じである。

H14: プログラムにおけるバグの有無にかかわらず、プログラム

のモジュール数は同じである。
それらの分散分析の結果を表３に示す。

表３　分散分析の結果

a. ソースコード点 ［4つのグループ間］

| 要因 | 自由度 | 平方和 | 平均平方 | $F_0$ | |
|---|---|---|---|---|---|
| グループ間 | 3 | 666.5 | 222.2 | 4.41* | (0.011) |
| 誤差 | 32 | 1613 | 50.4 | | |
| 計 | 35 | 2279.5 | | | |

b. 作業工数 ［4つのグループ間］

| 要因 | 自由度 | 平方和 | 平均平方 | $F_0$ | |
|---|---|---|---|---|---|
| グループ間 | 3 | 560.5 | 186.8 | 2.10 | (0.119) |
| 誤差 | 32 | 2844 | 88.9 | | |
| 計 | 35 | 3404.5 | | | |

c. 作業工数 ［グループDを除き、グループAとその他のグループ］

| 要因 | 自由度 | 平方和 | 平均平方 | $F_0$ | |
|---|---|---|---|---|---|
| グループ間 | 1 | 380.0 | 380.0 | 3.97 | (0.056) |
| 誤差 | 28 | 2679 | 95.7 | | |
| 計 | 29 | 3059 | | | |

d. プログラム・サイズ ［グループDと他のグループ］

| 要因 | 自由度 | 平方和 | 平均平方 | $F_0$ | |
|---|---|---|---|---|---|
| グループ間 | 1 | 17210 | 17210 | 4.10 | (0.051) |
| 誤差 | 34 | 142700 | 4198 | | |
| 計 | 35 | 159910 | | | |

e. モジュール数 ［グループDと他のグループ］

| 要因 | 自由度 | 平方和 | 平均平方 | $F_0$ | |
|---|---|---|---|---|---|
| グループ間 | 1 | 84.1 | 84.1 | 5.06* | (0.031) |
| 誤差 | 34 | 564.7 | 16.6 | | |
| 計 | 35 | 648.8 | | | |

( )内の数値：帰無仮説が成立する確率
*：有意水準5%

表３の結果から、次のことが認められる。

(1) a.より、４つのグループのソースコード点に関して、帰無仮説(H11)(学生のソースコード点の分布が４つのグループとも同じである）が成り立つ確率が1.1%であるので、有意水準５％で仮説を破棄できて、グループ間に差があると判断できる。つまり、初期設計の点の善し悪しが最終ソースコードの点の善し悪しに影響を及ぼすことが統計的に認められる。

(2) b.より、グループ間の作業工数(H12)に関しては、初期設計の点の善し悪しによる効果が見られるが統計的には有意差があるとは言い切れない。しかし、c.より、グループＤを除いてグループＡと他の２つのグループ（Ｂ，Ｃ）との間には統計的に有意差があるとほぼ言うことができる。

(3) d.より、グループＤと他のグループの間におけるプログラム・サイズ(H13)に関して、プログラムのバグの有無による効果が見られると統計的にほぼ言うことができる。

(4) e.より、グループＤと他のグループの間におけるプログラムのモジュール数(H14)に関して、プログラムのバグの有無による効果が見られると統計的に言うことができる。

### 2.3 実験１における結果

４つのグループ間における相違の分析から、次のことが結論できる。

(1) 初期設計の質は最終ソースコードの質に大きく影響を及ぼす。

(2) 良い初期設計はその後の設計やコーディングおよびデバッグに要する努力の量を少なくできる。

(3) 最終ソースコードが正しく動作しなかったグループの学生は、初期設計点が悪くないにもかかわらず、プログラム・サイズやモジュール数の平均値は、他のグループにおける学生のそれらの値に比較して少ない。

つまり、グループＤにおける学生はプログラムを完成させるのに必要な平均的な努力を費やさなかったと推測できる。

## 3. 実験２：設計法の比較

### 3.1 実験の目的

設計法間の比較に関して、いくつかの研究があるが、実験に基づいたものはほとんどない。

その原因として、[4]に見られるように、各設計法はそれぞれの特色をもっているので、それらの比較が困難であることが考えられる。そこで、筆者らは、設計法学習の効果を調べるために、設計法学習前の自己流の場合と複合設計法を用いた場合のプログラム作成過程の比較実験を行なうことにした。

複合設計法は比較的理解がしやすく、モジュール強度やモジュール間結合度の定義が明確であり、良いモジュール設計する際の指針となっている。

複合設計法ではモジュール分割が主体となっているので、複合設計法におけるモジュールの数は自己流によるものより多くなり、それにともない実行時間は多くなると推定することができる。この結果を予想して、モジュールの特徴

について、次の実験を行なった。

(1)実験1で述べたように、各学生が自分の初期設計を基にして、2週間でコーディングを終えた最終ソースリストがある。

(2)実験1から約2カ月後、各学生に複合設計法を用いて、在庫管理の課題を再びプログラムさせ、完成した最終ソースコードを提出させた。

(3)データは各学生のレポートから集められたものと、実験1の場合と同様にプログラム作成過程を計算機で自動的に収集したものがあり、以下のものからなる。

(a)学生から申告された机上での詳細設計やコーディングに費やした作業時間記録

(b)完成後のソースリスト

### 3.2 設計におけるモジュールについての計測

この実験で用いるプログラミング言語としてPascal を採用したために（分割翻訳の機能がないPascal コンパイラを用いたために）、複合設計法におけるモジュールの本質的な概念や評価基準を厳密には直接に適用することができなかった。

複合設計法におけるモジュールの基本的概念は[5]にも述べられているが、筆者らは[6]に述べられているものを採用した。そこではモジュールに含まれる属性として、

(1)複数ステートメントが語彙としてまとまっている。

(2)ステートメント群は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。

Pascal プログラムで頻繁に用いられる内部手続きは上記の3つの属性を満たすので、それらをモジュールとして取り扱った。次節以降に、筆者らが用いたモジュールの評価基準を述べる。

#### 3.2.1 モジュール強度

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弱い順に、機能的、情報的、連絡的、手順的、時間的、論理的及び暗号的強度の7タイプがある[5]。

本実験では、情報的強度と連絡的強度の2つの尺度を除外した。情報的強度は多重入力点をもち1つの入力点で特定の1つの機能を実行するモジュールであり、通常のPascal プログラムに存在しない。連絡的強度は手順的強度に連絡要素をもったものであるが、連絡要素の識別が困難であり、手順的強度とみなした。なお、暗号的強度と論理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、モジュール強度として採用したのは、機能的、手順的及び時間的強度の3タイプがある。それらの強度を識別した主たる評価基準を示す。

(a) 機能的強度：単一機能の手続きや関数。

(b) 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。

(c) 時間的強度：初期化などの手続き。

#### 3.2.2 モジュール間結合度

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は共通結合とみなし、内容結合は使用言語の文法仕様からありえなく、また制御結合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、モジュール間結合度として採用したのは、データ結合、スタンプ結合及び共通結合の3タイプがある。それらの結合度を識別した主たる評価基準を示す。

(a) 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。

(b) 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。

(c) 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。

### 3.3 データ収集と分析

各々の課題について、2つの最終ソースリストを分析した。その1つは各学生の考えに基づく自己流であり、もう1つは複合設計法に基づくものである。

図２に２０番の学生の設計方法について、モジュール設計の結果をモジュール強度とモジュール間結合度の例を示す。

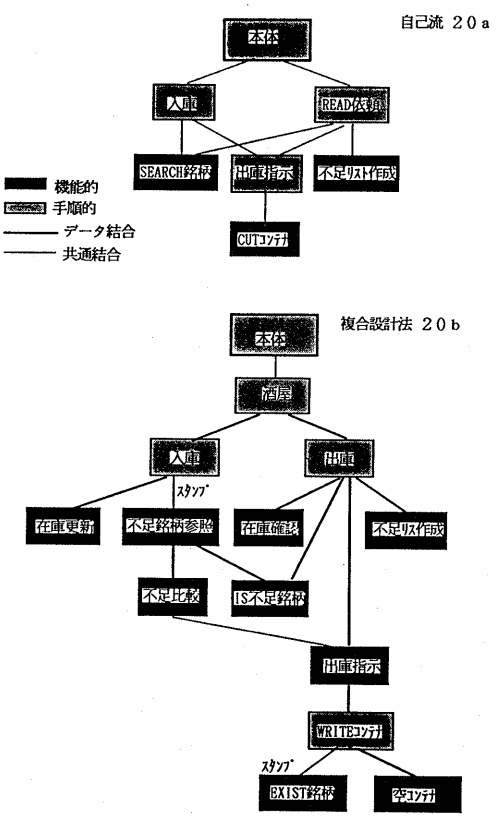


図２　階層構造図

この分析においては、モジュールの内部構造とモジュール性に注意を払った。この課題に関係する学生の数は１１名であったが、下記の理由で最終データの解析においては６名の結果を用いた。

(1) ４種類のテスト・データにより、２名分については、正しくプログラムが動作しなかった。

(2) ３名分については、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、類似度が高かった（複合設計法を用いて作成したとは考えられないもの）。

今回、解析の対象とした被験者６名分の作成法別に、机上において、プログラミングやデバッグに費やした作業時間（工数），プログラムのサイズ（行数），実行速度などを表４に示す。

表4．実験2における計測

| 学生番号 | 設計法 | 工数(時間) | プログラムサイズ(行数) | 実行時間(ms) | モジュールの数 |
|---|---|---|---|---|---|
| 3 | a | 38 | 370 | 712 | 18 |
| | b | 29 | 485 | 646 | 23 |
| 9 | a | 23 | 341 | 648 | 9 |
| | b | 13 | 348 | 758 | 13 |
| 17 | a | 27 | 429 | 676 | 13 |
| | b | 14 | 427 | 694 | 20 |
| 18 | a | 21 | 357 | 518 | 13 |
| | b | 24 | 417 | 722 | 16 |
| 20 | a | 62 | 487 | 656 | 9 |
| | b | 36 | 438 | 658 | 16 |
| 38 | a | 33 | 378 | 736 | 12 |
| | b | 25 | 469 | 670 | 13 |
| 平均 | a | 34.0 | 393.7 | 657.7 | 12.3 |
| | b | 23.5 | 430.7 | 691.3 | 16.8 |

a：自己流
b：複合設計

表４より、それぞれ自己流と複合設計法について、次のことが言える。

1) 工数の平均値は34.0と23.5であり、自己流の方が多い。

2) モジュール数の平均値は12.3と16.8であり、複合設計法の方が多い。

3) プログラム・サイズとテスト・データの実行時間の平均値は２つの方法ともほぼ同じである。

また、モジュール分析の要約であるモジュール強度とモジュール間結合度の各タイプにおける割合を表５に示す。

表5．モジュールに関する計測結果

| 学生番号 | 設計法 | モジュール強度(%) | | | モジュール間結合度(%) | | |
|---|---|---|---|---|---|---|---|
| | | 機能的 | 手順的 | 時間的 | データ | スタンプ | 共通 |
| 3 | a | 56.3 | 37.5 | 6.3 | 0.0 | 0.0 | 100.0 |
| | b | 66.7 | 28.6 | 4.8 | 3.8 | 0.0 | 96.2 |
| 9 | a | 28.6 | 57.1 | 14.3 | 0.0 | 14.3 | 85.7 |
| | b | 27.3 | 72.7 | 0.0 | 41.7 | 16.7 | 41.7 |
| 17 | a | 27.3 | 54.5 | 18.2 | 8.3 | 0.0 | 91.7 |
| | b | 50.0 | 44.4 | 5.6 | 60.0 | 5.0 | 35.0 |
| 18 | a | 45.5 | 45.5 | 9.1 | 8.3 | 0.0 | 91.7 |
| | b | 35.7 | 57.1 | 7.1 | 64.7 | 0.0 | 35.3 |
| 20 | a | 42.9 | 57.1 | 0.0 | 0.0 | 0.0 | 100.0 |
| | b | 64.3 | 35.7 | 0.0 | 73.3 | 13.3 | 13.3 |
| 38 | a | 60.0 | 40.0 | 0.0 | 0.0 | 0.0 | 100.0 |
| | b | 45.5 | 54.5 | 0.0 | 50.0 | 33.3 | 16.7 |
| 平均 | a | 43.4 | 48.6 | 8.0 | 2.8 | 2.4 | 94.9 |
| | b | 48.3 | 48.8 | 2.9 | 48.9 | 11.4 | 39.7 |

表５より、同様に次のことが言える。

4) モジュール強度における平均値は２つの方法ともほぼ同じである。

5) データ結合における平均値は2.8と48.9であり、複合設計法の方がデータ結合のモジュールの割合が多い。

6) 共通結合における平均値は94.9と39.7であり、自己流の方が共通結合のモジュールの割合が多い。

次に、各要因における設計法あるいは学生間での有意差の検定を二元配置（繰り返しなし）の分散分析法を用いて行なった。ここでは、次の帰無仮説をたて検定を行なった。

H21: 設計法の違いを問わず、作業工数は変わらない。以下同様に、

H22: プログラム・サイズは変わらない。

H23: 実行時間は変わらない。

H24:プログラムのモジュール数は変わらない。

H25: 設計法の違いを問わず、プログラムを構成しているのモジュールについて、機能的強度のモジュールの割合は変わらない。

H26: 同様に、手順的強度のモジュールの割合は変わらない。

H27: 設計法の違いを問わず、プログラムを構成しているのモジュール間の結合において、データ結合の割合は変わらない。

H28: 同様に、結合の割合は変わらない。

表６には、分散分析の統計的結果を示す。

表６より、次のことが言える。

(1) aより、２つの設計方法の間と学生の間における工数については(H21)、ともに有意差である。

(2) dより、２つの設計方法の間におけるモジュール数については(H24)、著しい有意差があり、学生の間にも有意差がある。

(3) gより、２つの設計方法の間におけるデータ結合の割合については(H27)、有意差がある。

(4) hより、２つの設計方法の間における共通結合の割合については(H28)、有意差がある。

なお、H22,H23,H25,H26については有意な結果は得られなかった。

表６　分散分析の結果

a. 作業工数

| 要因 | 自由度 | 平方和 | 平均平方 | $F_0$ |
|---|---|---|---|---|
| 設計法間 | 1 | 330.8 | 330.8 | 7.56 (0.040)* |
| 学生間 | 5 | 1311 | 262.2 | 5.99 (0.036)* |
| 誤差 | 5 | 218.8 | 43.8 | |
| 計 | 11 | 1860 | | |

b. プログラム・サイズ

| 要因 | 自由度 | 平方和 | 平均平方 | $F_0$ |
|---|---|---|---|---|
| 設計法間 | 1 | 4107 | 4107 | 2.12 (0.205) |
| 学生間 | 5 | 16720 | 3344 | 1.73 (0.281) |
| 誤差 | 5 | 9673 | 1935 | |
| 計 | 11 | 30500 | | |

c. 実行時間

| 要因 | 自由度 | 平方和 | 平均平方 | $F_0$ |
|---|---|---|---|---|
| 設計法間 | 1 | 3400 | 3400 | 0.61 (0.471) |
| 学生間 | 5 | 10060 | 2013 | 0.36 (0.857) |
| 誤差 | 5 | 27980 | 5596 | |
| 計 | 11 | 41440 | | |

d. モジュール数

| 要因 | 自由度 | 平方和 | 平均平方 | $F_0$ |
|---|---|---|---|---|
| 設計法間 | 1 | 60.75 | 60.75 | 22.09 (0.005)** |
| 学生間 | 5 | 120.4 | 24.08 | 8.76 (0.016)* |
| 誤差 | 5 | 13.75 | 2.75 | |
| 計 | 11 | 194.9 | | |

e. 機能的強度

| 要因 | 自由度 | 平方和 | 平均平方 | $F_0$ |
|---|---|---|---|---|
| 設計法間 | 1 | 69.6 | 69.6 | 0.56 (0.489) |
| 学生間 | 5 | 1505 | 301.0 | 2.41 (0.179) |
| 誤差 | 5 | 625.1 | 125.0 | |
| 計 | 11 | 2199.7 | | |

f. 手順的強度

| 要因 | 自由度 | 平方和 | 平均平方 | $F_0$ |
|---|---|---|---|---|
| 設計法間 | 1 | 0.14 | 0.14 | 0.00 (0.974) |
| 学生間 | 5 | 1044 | 208.8 | 1.70 (0.381) |
| 誤差 | 5 | 613.5 | 122.7 | |
| 計 | 11 | 1657.6 | | |

g. データ結合

| 要因 | 自由度 | 平方和 | 平均平方 | $F_0$ |
|---|---|---|---|---|
| 設計法間 | 1 | 6389 | 6389 | 23.65 (0.005)** |
| 学生間 | 5 | 1797 | 359.3 | 1.33 (0.381) |
| 誤差 | 5 | 1351 | 270.1 | |
| 計 | 11 | 9537 | | |

h. 共通結合

| 要因 | 自由度 | 平方和 | 平均平方 | $F_0$ |
|---|---|---|---|---|
| 設計法間 | 1 | 9125 | 9125 | 20.04 (0.006)** |
| 学生間 | 5 | 2370 | 474.1 | 1.04 (0.483) |
| 誤差 | 5 | 2276 | 455.3 | |
| 計 | 11 | 13771 | | |

( )内の数値：帰無仮説が成立する確率

*：有意水準５%

**：有意水準１%

### 3.4 実験2における結果

2つの設計法におけるいくつかの特徴とモジュール強度とモジュール間結合度の相違の分析から、次のことが結論できる。

(1) 複合設計法によるモジュールの数は自己流によるそれよりも多くなる。
(2) 学生間の工数の量は個人差がある。
(3) 学生間のモジュールの数は個人差がある。
(4) 複合設計法は自己流よりも共通結合の割合が少ない。
(5) これに反して、複合設計法は自己流に比較してデータ結合の割合が多い。

データ分析より、複合設計法は工数の総数がより少なくて済むことを意味するが、複合設計法での実験は、自己流を用いてプログラムを完成させた後に、同じ課題のために工数がより少なかった可能性もある。つまり、プログラミングにおける学習効果が設計法の効果よりもさらに有意であることを示唆できる。また、4つのテスト・データにおける実行時間の合計は2つの方法ともほとんどかわらないが、これはテスト・データの量が少ないためと考えられる。

## 4. おわりに

プログラム設計過程に関する二つの実験を行なった。実験1から、良い初期設計はコーディングとデバックのために費やす工数の合計を減らすことができると結論でき、最終ソースコードの質にも好影響を及ぼすことが確かめられた。

最終プログラムにおけるバグの存在は、設計における欠陥を征服することが不十分であったのに、コーディングとデバックのために費やす時間の合計が総体的に少なかったことが大きな原因となっていると推測される。

実験2では、モジュール結合度の基準において、複合設計法を教えられる前のプログラムよりも、その使用を教えられた後のプログラムの方が、良いモジュール構造になることが認められた。各学生の個人差によりモジュール数が異なるが、自己流よりも複合設計法を用いる方が多くなる。しかし、モジュール強度については、統計的にも有意な結果が得られなかった。今後、複合設計法を教えた学生の成果とそれを教えなかった学生の成果を比較することにより、複合設計法の効果を確かめるなどの実験をすることを計画している。

謝辞

この研究に関して有益な助言を頂いたLloyd G. Williams教授(Colorado大学)、統計解析に適用に関して有益な助言を頂いた大澤　豊教授(大阪大学情報処理教育センター長)、および評価等に協力して頂いた鳥居研究室の松本健一・大西　論・野村研仁の諸氏に感謝の意を表します。

参考文献

[1] 山崎: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).
[2] 山崎: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,p.1219(1984).
[3] 木村訳:プログラム書法,共立出版(1981).
[4] Kelly,J.C. : "A comparison of four design methods for real-time systems", Proc. of 9th International Conference on Software Engineering", pp.238-252,(1987.3).
[5] 久保・国友訳:高信頼性ソフトウェア－複,合設計,近代科学社(1976).
[6] 久保:複合設計,情報処理,Vol.25.No.9,pp.935-945(1984).
[7] 畑村・奥野・津村訳:ｽﾈﾃﾞｶﾞｰ、ｺｸﾗﾝ統計的方法,岩波書店(1972).
[8] H.Kudo,Y.Sugiyama,M.Fujii,K.Torii : "Quantifying a Design Process based on Experiments",Proc. of the 21st Annual Hawaii Int. Conf. on System Sciences, Vol.2,pp.285-292(1988).