ソフトウェア　工学　12－4
（１９７９．１０．２６）

プログラムの複雑さに対する尺度の評価

春原　猛　　高野　彰
（三菱電機（株）　開発本部）

１．まえがき

ソフトウェアの開発と保守の管理をより客観的かつ正確に行うために，プログラムの品質に関する種々の定量的な尺度を使用していかなければならない。プログラムの複雑さの尺度はそのようなもののーつであり，最近いくつかの尺度が提案されている。ここで，「プログラムの複雑さ」とはプログラムを理解する難しさの程度を意味する。しかし，プログラムを理解する難しさの度合いは理解する人間に強く係り合い，客観的尺度を設定することが難しい。これまでに提案されている尺度は，この点に関して充分であるとは言えない。本論では，複雑度が大きいほど，ソフトウェア開発における生産量，作業時間，エラー数等の開発管理データの値も大きくなると仮定し，実際のソフトウェア開発において収集したデータをもとに両者の相関を評価し，複雑度の開発管理尺度としての可能性を論じる。

２．標本データ

複雑さの尺度の評価は，テスト・デバッグツールの開発の際に収集されたデータに基づいて行われた。プログラムは注釈行も含めて約25.5Kステップであり，200個のモジュール（コンパイル単位：平均133.6ステップ，またはアセンブリ単位：平均143.0ステップ）から構成されている。プログラムの70％のモジュールは，構造化コーディング用の機能をもつFORTRAN型の言語で書かれ，残りのモジュールはアセンブリ言語で書かれている。プログラムの開発は，他の仕事も兼務したプログラマ（４人～６人）によって行われた。

2.1　開発管理データ

開発の各段階の作業内容と収集データは次のとおりである。

機能設計・・・・ツールのもつべき機能とその環境とのインタフェイスおよび計算機システムの構成が明らかにされた。

論理設計・・・・機能のプログラムによる実現方法を，対象計算機システムとは独立に，論理的に設計した。その際にHIPO手法を用いてトップダウン設計が行われた。

物理設計・・・・プログラムが実際に運用される計算機システムの特性を考慮しながら物理モジュールとそれらの階層構成，データ構成，インタフェイス等の設計をした。

詳細設計・・・・物理設計で設定された各モジュールについて詳細設計を行い，HIPO記法で記述した。HIPOダイアグラムの処理部はPDL[5]様式で記述することにより制御構造が明確にされた。

◎収集データ：。詳細設計時間　＜TIMED＞
。HIPOダイアグラムの処理部の行数＜LINED＞

コーディング・・コーディング規約にそってコーディングされ，エラーのない

ソースリストが作成された。また，各モジュールに対してテストケースを記述したモジュールテスト仕様書が作成された。

◎収集データ：
- コーディング時間 <TIMEC>
- テスト仕様書の行数 <LINET>

コーディング・レビュー：コーディングレビューでは，ソースリストおよびテスト仕様書のレビューを行った。レビューはチェックシートを用いて行い，エラーが検出されるとエラー記録シートに記入された。また，レビューはコーダーが担当したが，85％のモジュールについてコーダーとレビュー担当者を入れかえた。

◎収集データ：
- コーディングレビュー時間 <TIMECR>
- 設計エラー数 <ERRORD>
- コーディングエラー数 <ERRORC>

単体テスト・総合テスト：当初は，単体テストにおけるモジュール毎の作業時間を収集する予定であったが，種々の事情によりできなかった。エラーが検出されるとエラー記録シートに記入された。

◎収集データ：
- 設計エラー数 <ERRORD>
- コーディングエラー数 <ERRORC>

2.2 複雑度の測定値

評価の対象として考慮した複雑度はステップ数，コントロールフローグラフに基づくもの，オペレータとオペランド数に基づくものであり，以下にあげる各々の尺度に対してモジュール毎に測定した。

[ステップ数]
- ①最大ステップ数（ソースコードの総ステップ数（行数）） <STEP max>
- ②中間 〃 （①－注釈行） <STEP mid>
- ③最小 〃 （②－宣言ステップ数） <STEP min>

[コントロールフローグラフに基づくもの]
- ④グラフのノード数 <NODE>
- ⑤ 〃 のエッジ数 <EDGE>
- ⑥McCabeの複雑度1（すべての条件を数えた場合[1]） <V(G) max>
- ⑦ 〃 2（各条件節の条件数を1とした場合[3]） <V(G) mid>
- ⑧ 〃 3（⑦とCASE構造の条件数を1とした場合[4]） <V(G) min>

[オペレータとオペランド数に基づくもの]
- ⑨オペレータの種類の数 <ETA1>
- ⑩ 〃 の総出現回数 <N1>
- ⑪オペランドの種類の数 <ETA2>
- ⑫ 〃 の総出現回数 <N2>
- ⑬Halsteadの複雑度[2] <E>

3. 複雑度と生産量の相関

各モジュールの複雑度と生産量（詳細設計仕様書の行数：LINED，ソースコードのステップ数：STEP，モジュールテスト仕様書の行数：LINET）との相関を示したものが表1，2である。ステップ数は複雑度の尺度の一つとして考えられて

いるが，ここでは生産量を示す尺度としても使用している。

表1. 複雑度と生産量の相関係数（コンパイルモジュール137個）

複雑度＼生産量	LINED	STEP max	STEP mid	STEP min	LINET
STEPmax	.778				.647
〃 mid	.762				.603
〃 min	.812				.591
NODE	.863	.844	.836	.901	.593
EDGE	.879	.849	.842	.906	.642
V(G)max	.875	.824	.813	.863	.705
〃 mid	.870	.824	.816	.877	.690
〃 min	.859	.812	.803	.866	.621
ETA1	.681	.697	.674	.698	.503
N1	.783	.818	.813	.834	.701
ETA2	.628	.720	.713	.711	.533
N2	.751	.791	.790	.817	.678
E	.758	.765	.757	.786	.739

表2. 複雑度と生産量の相関係数（アセンブリモジュール63個）

複雑度＼生産量	LINED	STEP max	STEP mid	STEP min	LINET
STEPmax	.681				.555
〃 mid	.681				.534
〃 min	.692				.488
NODE	.892	.682	.690	.724	.439
EDGE	.891	.683	.691	.723	.439
V(G)max	.881	.688	.695	.713	.444
〃 mid	.882	.680	.687	.716	.436
〃 min	.875	.676	.684	.714	.439
ETA1	.699	.939	.937	.972	.524
N1	.717	.958	.962	.992	.505
ETA2	.683	.896	.890	.905	.476
N2	.696	.955	.958	.987	.473
E	.621	.894	.908	.949	.397

詳細設計仕様書の行数（HIPOダイアグラムの処理部の行数）とソースコードの手続き部のステップ数STEP min との相関が強いことが，表1，2で示されている。すなわち，HIPOダイアグラムの処理部の記述とソースコードの対応づけがされていることを暗示している。

詳細設計仕様書の行数と相関が強い複雑度は，コンパイルおよびアセンブリモジュール共に，コントロールフローグラフに基づくものである（危険率1%）。制御構造を明示する形式で記述したことが，強い相関を示す大きな原因の一つであろう。全般的にどの尺度も設計仕様書の行数とかなり強い相関をもっており，仕様書の行数が多いほどプログラムが複雑であるとするならば，いずれの尺度も複雑さを反映しているといえる。

ソースコードのステップ数と相関が強い複雑度は，コンパイルモジュールに対しては，コントロールフローグラフに基づくものであり，アセンブリモジュールに対してはオペレータとオペランドの数に基づくものである（危険率1%）。アセンブリモジュールの場合，ソースコードのステップ数がオペレータの総出現回数N1と強い相関をもつことは，アセンブリ言語のオペレータ中心の記述形式から，当然の結果といえる。

モジュールテスト仕様書の行数LINETと相関が強い複雑度は，コンパイルモジュールに対してはオペレータやオペランドに基づくものであり，アセンブリモジュールに対してはステップ数である（危険率1%）。コンパイルモジュールの場合，V(G) max がLINETとかなり強い相関を示し，複雑度V(G)がテストパスと関連していること[1]が，実際のデータでも示されている。モジュールテスト仕様書の行数と複雑度の相関が他の二つの場合よりも小さい理由としては，担当者のテストケース選択基準のバラツキや記述の詳細さの違いなどが考えられる。

4. 複雑度と作業時間の相関

各モジュールの複雑度と作業時間（詳細設計時間：TIMED，コーディング時間：TIMEC，コーディングレビュー時間：TIMECR，およびこれらの時間を合計したもの：TIMET）との相関を示したのが表3，4である。

表3. 複雑度と作業時間の相関係数(コンパイルモジュール)

作業時間 / 複雑度	TIMED	TIMEC	TIMECR	TIMET
STEPmax	.706	.519	.589	.773
〃 mid	.657	.432	.563	.717
〃 min	.628	.396	.531	.679
NODE	.662	.431	.556	.716
EDGE	.679	.452	.535	.720
V(G)max	.681	.474	.498	.708
〃 mid	.679	.463	.485	.699
〃 min	.642	.471	.481	.678
ETA1	.602	.306	.481	.624
N1	.713	.327	.603	.748
ETA2	.567	.235	.544	.620
N2	.681	.295	.609	.727
E	.714	.346	.513	.710

表4. 複雑度と作業時間の相関係数(アセンブリモジュール)

作業時間 / 複雑度	TIMED	TIMEC	TIMECR	TIMET
STEPmax	.177	.102	.432	.443
〃 mid	.160	.106	.401	.411
〃 min	.146	.038	.336	.340
NODE	.097	.064	.206	.223
EDGE	.083	.071	.203	.214
V(G)max	.060	.086	.199	.199
〃 mid	.057	.084	.196	.194
〃 min	.063	.069	.185	.187
ETA1	.162	-.006	.355	.354
N1	.208	.010	.351	.386
ETA2	.223	.051	.354	.407
N2	.215	-.017	.341	.376
E	.044	-.027	.234	.185

詳細設計時間と相関が強い複雑度は，コンパイルモジュールに対してはオペレータおよびオペランド数に基づくものであり（危険率1%），アセンブリモジュールに対してはいずれの複雑度も相関があるとはいえない。アセンブリモジュールに対して複雑度との相関がない理由としては，アセンブリモジュールは機能が比較的単純な共通ルーチン類であったためと考える。すなわち，複雑度の小さいモジュールの詳細設計では，プログラムロジックそのものを検討する以上に他のモジュールとのインタフェイスや共通データ参照のための資料参照等の作業時間の割合が多くなり，ここで取りあげられているプログラムの手続き部とを対象にした複雑度と作業時間の相関は小さいと考えられる。

コーディング時間と相関が強い複雑度は，コンパイルモジュールに対してはステップ数とコントロールフローグラフに基づくものであり（危険率1%），アセンブリモジュールに対してはいずれの複雑度も相関があるとはいえない。コーディング時間と複雑度の相関が詳細設計時間やコーディングレビューの時間の場合と比較して小さい。これは詳細設計の記述がかなりの程度にソースコードに対応するような形式で記述されたため，コーディング作業がある程度機械的に行われ，複雑度があまり影響してないと考えられる。また，アセンブリモジュールの場合には相関がないが，これはアセンブリコーディングではコーディングの自由度が大きくて，多くの場合詳細設計からコーディングする担当者に応じてさまざまなかのコーディングとなる。また，プログラムサイズの制限がきついため，サイズ縮少に時間がかけられていることも原因していると考えられる。

コーディングレビュー時間に対して相関が強い複雑度は，コンパイルモジュールに対してはオペレータとオペランド数に基づくものとステップ数であり（危険率1%），アセンブリモジュールに対してはステップ数およびオペレータとオペランドにもとづくものである（危険率5%）。コーディングレビューにおいては，コーダーとレビュー担当者を入れかえたことにより，モジュールを理解することが必要であり，複雑度が影響すると考えられる。

各作業時間の総和と複雑度の相関を見ると，ステップ数との相関が強い（コンパイルモジュールの場合危険率1%，アセンブリモジュールの場合危険率5%）。全般的に作業時間と相関の強い複雑度の順は，ステップ数，オペレータとオペランド数に基づくもの，そしてコントロールフローグラフに基づくものである。この順序は与作業の多さを反映した順とも考えられる。

5. 複雑度とエラー数の相関

各モジュールに対してコーディングレビューおよび単体・総合テスト過程を通して検出されたエラー数について，設計エラー数（ERRORD）とコーディングエラー数（ERRORC）に大別して複雑度との相関を求めた結果が表5，6である。

表5. 複雑度とエラー数の相関係数（コンパイルモジュール）

複雑度＼エラー数	ERRORD	ERRORC	ERRORT	ERRORP
STEPmax	.310	.426	.488	.515
〃 mid	.346	.442	.517	.520
〃 min	.249	.242	.309	.493
NODE	.259	.246	.317	.508
EDGE	.277	.239	.319	.506
V(G)max	.302	.241	.331	.499
〃 mid	.292	.221	.310	.483
〃 min	.262	.219	.296	.470
ETA1	.269	.175	.262	.475
N1	.363	.266	.379	.601
ETA2	.247	.248	.313	.521
N2	.339	.278	.378	.593
E	.352	.254	.364	.508

表6. 複雑度とエラー数の相関係数（アセンブリモジュール）

複雑度＼エラー数	ERRORD	ERRORC	ERRORT	ERRORP
STEPmax	.056	.640	.609	.633
〃 mid	.028	.644	.604	.634
〃 min	.031	.656	.616	.658
NODE	.071	.680	.652	.714
EDGE	.079	.681	.655	.719
V(G)max	.089	.672	.650	.716
〃 mid	.093	.677	.656	.722
〃 min	.102	.670	.652	.720
ETA1	.105	.636	.622	.654
N1	.066	.644	.617	.657
ETA2	.149	.529	.537	.563
N2	.046	.653	.618	.654
E	-.066	.702	.626	.670

設計エラー数と相関が強い複雑度は，コンパイルモジュールに対してはオペレータとオペランドに基づくものであり（危険率1%），アセンブリモジュールに対してはいずれの尺度に対しても相関があるとはいえない。アセンブリモジュールの設計エラー数と複雑度の相関がない理由としては，アセンブリモジュールは機能が比較的単純なものであり，エラー数が少なかったためと考えられる。コーディングエラー数と相関が強い複雑度は，コンパイルモジュールに対してはステップ数であり，アセンブリモジュールに対してはコントロールフローグラフに基づくものとステップ数である（危険率1%）。設計エラー数の場合と異なり，アセンブリモジュールのほうが複雑度に対して強い相関を示している。また，設計エラー数に対しては，データ参照も考慮したオペレータとオペランド数に基づく複雑度が，コーディングエラー数に対しては，ステップ数のようにより単純な，量的な複雑度が強い相関をもっていることは，両者のエラーの性格の違いをある程度反映していると言えよう。

エラー総数（ERRORT = ERRORD + ERRORC）から宣言文のエラーを除いたエラー数（ERRORP）と相関が強い複雑度は，コンパイルモジュールに対してはオペレータとオペランド数に基づくものであり，アセンブリモジュールに対してはコントロールフローグラフに基づくもの，およびオペレータとオペランドに基づくものである（危険率1%）

6. 開発管理尺度としての複雑度

6.1 モジュール分割の基準

プログラムのモジュールを適当な大きさに限定することにより，保守性や信頼性を向上させることが考えられる。そのようなモジュール分割の基準の代表例として，モジュールのステップ数を50～100程度におさえるのが望ましいといわれている。また，McCabeはステップ数でなく，彼の提案した複雑度を基準とするのが妥当であるとし，基準値としてV(G)=10を提案している[1]。図1，2，3

はSTEP$_{mid}$（注釈行を除いたステップ数），V(G)$_{max}$（McCabeの複雑度），E（Halsteadの複雑度）の各々について，値を適当な区間に分割し，各々の区間において対応する開発管理データの平均値をもとめて図示したものである。複雑度Eの場合は，他の二つと比較して若干変動が大きい。管理データは，複雑度の値が大きくなるにつれて一次関数以上の増加率で値が大きくなる傾向を示しており，モジュール分割の基準値を小さくしておいたほうが良いことは，このような実測データからも示される。

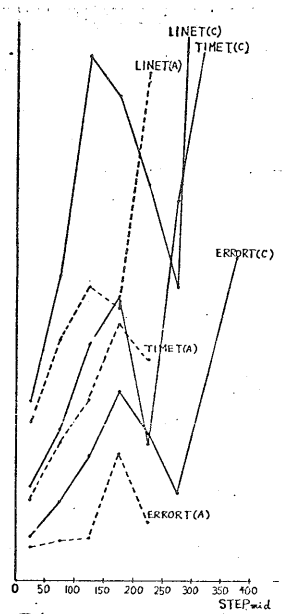

図1. ステップ数と管理データ
注) STEP$_{mid}$ ≥ 200 なるモジュールは9個

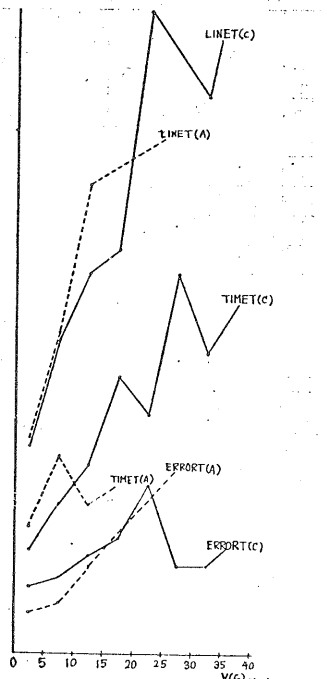

図2. McCabeの複雑度と管理データ
注) V(G)$_{max}$ ≥ 25 なるモジュールは9個

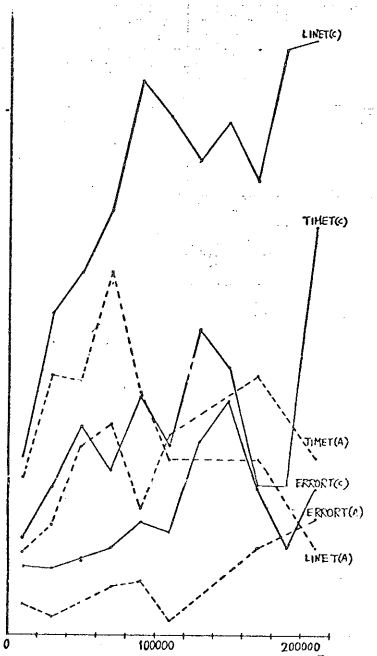

図3. Halsteadの複雑度と管理データ
注) E ≥ 200000 なるモジュールは10個

6.2 生産量の見積り

3章で得られた結果から明らかなように，ここで評価の対象となった複雑度はいずれも生産量と強い相関を示しており，プログラムのステップ数，サイズ，テストケースの数等を見積る際の重要な要因となるであろう。また，McCabeの複雑度は，プログラムにおける条件判定の数から求められるので[1]，詳細設計の方法や記述形式を工夫することにより，プログラムのコーディングに入る前に測定が可能である。したがって，設計が終了した段階でモジュール分割の評価，ステップ数やサイズの予測等に使用できるであろう。

6.3 作業時間の見積り

4章での結果から，特にコンパイルモジュールの場合は，複雑度と作業時間の相関が強く，複雑度が作業時間見積りの主要な要因となり得ることを示している。アセンブリモジュールについては，コーディングレビュー時間と複雑度の相関がかなりみられるものの，全体として相関が弱く，ここで評価されているもの以外の尺度を考えるべきであろう。

6.4 エラー数の見積り

5章での結果からは，アセンブリモジュールのコーディングエラー数と複雑度は強い相関を示し，複雑度がエラー数の見積りの主要な要因と考えられるが，設計エラーに対しては相関がないので要因とはなり得ないであろう。コンパイルモジュールの場合は，特に手続き部のエラー数と強い相関を示しており，複雑度をその部分のエラー数見積りの主要な要因として使用可能であろう。Halsteadの複雑度はエラー数と強い相関を示すことが報告されているが[2]，ここでのデータはそれほど強い相関を示してない。その理由としては，エラー検出の期間，エラーの分類と数え方，複雑度を測定するプログラム単位の違い等が考えられる。

6.5 複雑度の比較

従来から開発管理の尺度として一般的に使われているステップ数と，新しく提案されたMcCabeやHalsteadの複雑度を比較してみると，開発管理データとの相関ということではさほどの違いはない。そのことは表7に示されるように，ステップ数とそれらの複雑度の間に強い相関がみられることからもわかる。しかし，McCabeの複雑度は詳細設計が終った段階である程度測定可能であること，また，その複雑度がコントロールフローグラフの基本パスの最大数であるということから，テストパス選びの一つの基準となりうること等の特徴をもっている。一方，Halsteadの複雑度の場合は，さらに，関連する別の意味の尺度がそのまわりに定義されており，尺度の適用範囲が広いという特徴をもっている。

表7. 複雑度間の相関係数

	STEP max	〃 mid	〃 min	SIZE max	〃 min	NODE	EDGE	V(G) max	〃 mid	〃 min	ETA1	N1	ETA2	N2	E
STEPmax		.967	.880	.820	.858	.844	.849	.824	.824	.812	.697	.818	.720	.791	.765
〃 mid	.997		.905	.833	.855	.836	.842	.813	.816	.803	.674	.813	.713	.790	.757
〃 min	.961	.969		.734	.874	.901	.906	.863	.877	.866	.698	.834	.711	.817	.786
SIZEmax	.291	.265	.198		.833	.738	.764	.788	.774	.716	.625	.785	.601	.772	.796
〃 min	.940	.944	.967	.241		.877	.880	.854	.851	.834	.785	.932	.814	.919	.786
NODE	.682	.690	.724	.143	.812		.989	.922	.934	.950	.768	.838	.716	.812	.733
EDGE	.683	.691	.723	.136	.810	.999		.966	.976	.976	.758	.844	.687	.813	.789
V(G)max	.688	.695	.713	.122	.797	.987	.993		.992	.970	.708	.819	.617	.781	.839
〃 mid	.680	.687	.716	.122	.801	.989	.995	.998		.976	.711	.821	.617	.784	.842
〃 min	.676	.684	.714	.104	.800	.988	.994	.997	.998		.727	.813	.658	.780	.779
ETA1	.939	.937	.972	.169	.956	.723	.723	.714	.718	.720		.851	.858	.821	.693
N1	.958	.962	.992	.183	.977	.716	.714	.702	.705	.705	.974		.892	.988	.843
ETA2	.896	.890	.905	.120	.918	.642	.643	.637	.641	.642	.924	.925		.891	.619
N2	.955	.958	.987	.167	.962	.698	.696	.685	.687	.687	.971	.994	.923		.836
E	.894	.908	.949	.149	.884	.732	.734	.731	.733	.732	.901	.918	.779	.928	

（コンパイルモジュール）

（アセンブリモジュール）

表8. 管理データの変動係数

管理データ	コンパイルモジュール	アセンブリモジュール
LINED	72.3%	75.7%
LINET	86.7	61.6
TIMED	79.0	82.8
TIMEC	61.1	63.9
TIMECR	74.5	98.9
TIMET	60.7	53.1
ERRORD	157.0	165.0
ERRORC	112.3	142.6
ERRORT	98.1	118.5
ERRORP	102.1	124.3

表9. 複雑度の変動係数

注) SIZE max：絶対ステップ数
SIZE min：絶対ステップ数−データ領域ステップ数

複雑度	コンパイルモジュール	アセンブリモジュール
STEPmax	46.3%	72.4%
〃 mid	59.8	97.3
〃 min	71.2	108.5
NODE	68.0	126.9
EDGE	78.0	146.3
V(G)max	84.8	128.3
〃 mid	84.4	128.3
〃 min	76.7	126.9
ETA1	34.4	53.3
N1	65.0	93.5
ETA2	48.1	57.0
N2	68.9	94.2
E	172.3	335.0

6.6 言語と尺度の関係

管理データおよび複雑度の測定値の平均値，標準偏差から変動係数を求めた結果を表8，9に示す。総じてコンパイルモジュールのほうがアセンブリモジュールよりも変動係数が小さい。このことは，高位言語によるプログラミングのほうがより均質化されたプログラムができること，また，これらの複雑度が管理尺度として有効であることを示唆している。

7. まとめ

実際のソフトウェア開発で得られた管理データをもとに，ステップ数，McCabeの尺度，Halsteadの尺度等のプログラムの複雑度の評価を行い，次のような結果が得られた。

（1）McCabeおよびHalsteadの複雑度はソフトウェア開発の生産量，作業時間，エラー数等の開発管理データと強い相関をもつ。しかし，従来から一般的に用いられているステップ数も同様に，強い相関をもつ。

（2）McCabeの複雑度は設計終了時点で測定可能であり，プログラムのステップ数等の見積りや，モジュール分割の評価基準として有効である。

（3）Halsteadの複雑度はエラー数と強い相関を示すことが報告されているが，ここでは，それほど強い相関を示してない。

（4）ステップ数，McCabeの複雑度およびHalsteadの複雑度の間に強い相関が見られる。

（5）複雑度の測定値，開発管理データは，いずれもコンパイルモジュールのほうがバラツキが少なく，高位言語がプログラミングの均質化をもたらすことを示している。

これらの結果は，一つのプログラム開発におけるデータに基づくものであり，今後もこの種の評価を色々な種類のプログラムの，色々な開発環境におけるデータに対して行い，より普遍的な解をもとめていく必要がある。さらに，データ構造やデータ参照の複雑さ，およびモジュール間インタフェイスの複雑さを反映した複雑度についても，評価を行う必要がある。一方，このような評価に使用されるデータの収集法や，エラーの分類法の確立も評価結果の相互比較を可能にするためには必要である。

参考文献

[1] T.J.McCabe, "A Complexity Measure," IEEE Tr. on SOFTWARE ENGINEERING, Vol. SE-2, No.4, pp.308-320, Dec. 1976.

[2] M.H.Halstead, "Elements of Software Science," Elsevier North-Holland, Inc., 1977.

[3] G.J.Myers, "An Extension to the Cyclomatic Measure of Program Complexity," SIGPLAN Notices, Vol. 12, No.9, pp. 61-64, Oct. 1977.

[4] W.J. Hansen, "Measurement of Program Complexity by the Pair (Cyclomatic Number, Operator Count)," SIGPLAN Notices, Vol. 13, No.3, pp.29-33, March 1978.

[5] S.H. Caine & E.K. Gordon, "PDL-A tool for software design" NCC '75, pp. 271-276, 1975.