ソフトウエア工学 34－17
（１９８４．２．１０）

保守時におけるプログラムの理解と人間要因

則房 雅也 （日本電気（株）ソフトウェア生産技術研究所）

１．はじめに

近年保守対象ソフトが多くなり、逆にソフト要員不足は益々深刻になってきている。この様な状況下では、開発に携わらなかった人や新人にも保守作業を割り当てなければならない。保守作業に関した報告［１］によれば、担当者はドキュメントとソースプログラムの解読作業に最も時間をかけ、ソースプログラムの内容を理解するのに最も苦労しており、しかもソースプログラムを最も重要なドキュメントとして扱っている所が多い。これらの点を考えれば、どういった要因がソースプログラムの理解作業に影響を及ぼしているか調べ、開発や保守担当者自身ですら気付かないでいる問題点を明らかにすることは重要である。

本論文では、対象に対して深い知識を持たない人が、限られた時間内で保守作業を行うことを要求された場合を想定し、ソースプログラムを理解するとき、どの様な要因が関わってくるかを、比較対照実験［３］によって評価したので、その結果を報告する。

２．対照実験

２．１　目的と内容

これまでにもプログラム理解に関する実験報告は行われてきたが、多くの場合プログラミングスタイルやプログラムを説明する手段（コメントや流れ図）の効果に焦点を当てている。［２，５，６，７］これらの結果に対して、学生が被験者として使われた場合、現場で身近なものとして受け入れにくく［４］、ほとんどの報告が英語を使う人達によるもので、実験結果がそのまま日本のプログラマに当てはまるとは限らないと考えられる。

また経験等の人間要因がどう影響するかは、作業を割り当てるとき重要な要素となるものであり、作業者が苦手とする作業を分析することは、プログラミングスタイルやプログラムを説明する手段を改善する為に必要であり、これらこそ実験によって明らかにすべきことである。

実験結果を現場で生かす為には、我々の身近にいるプログラマを被験者として選び、彼ら自身ですら気付いていない問題点を明らかにすることが重要である。

実験は社内の２部門から３８名の被験者を集めて行った。Ｃ言語で書かれた４０－６０ステップのプログラム２種類を各自に与え、それぞれ３０分位で解読し、いくつかの質問に答えてもらった。被験者は皆実際に開発保守作業に携わっており、Ｃ言語に対してもある程度以上の知識を持っている。経験因子として以下のものを調べた。

・ソフト開発経験
・Ｃ言語でプログラムした経験
・Ｃ言語のプログラムを読んだ経験
・Ｃ言語以外の言語での経験

プログラムの選出に当たっては、以下の基準を置き、その評価を行えるものを用意した。以後それぞれに用意したプログラムを、ａタイプ、ｂタイプと呼ぶ。

ａタイプ：共有する領域の内容を変更する等、呼び出し側に副作用を与えるユーザ定義関数を使用しているもの。（図１）

ｂタイプ：ａタイプで扱っているような関数を含んでいないもの。

ａタイプでは、副作用を与える関数に対し、ａ１）英語のコメント（図２）、ａ２）日本語の説明（図３）、ａ３）ソースコードそのもの（図４）のいずれかを与える３通りの問題を用意し、ｂタイプでは、ａタイプで扱うような関数を含まず、行数でａタイプのものと同程度のプログラムｂ１（図５）と、それを機能拡張したプログラムｂ２（図６）の２通りを用意した。これらのプログラムの機能概要は以下の通りである。

ａタイプ：ファイル中のテキストを、指定された行数だけ書き出す。

ｂタイプ：ファイル中のテキスト全てを、行単位で行末から先頭へと逆順に書き出す。

実験に先立って予備実験を行い、実験に使うプログラム、質問内容、実験時間、実験の進め方等を評価し、不備を補うようにした。

２．２　理解作業

実験における理解作業の尺度として、リコールやエラー修正を使ったもの［６］もあるが、ここでは、与えた質問（図７，図８）に対する正解数を用いた。質問としては、

```
#include <stdio.h>

int     linecnt = 10;
int     argc;

main(Argc, argv)
        int Argc;
        char *argv[ ];
{
        char *name;
        char obuf[BUFSIZ];
        static int around;
        int argcnt;

        setbuf(stdout, obuf);
        Argc--, argv++;
        argc = Argc;
        do {
                argcnt = func(argv);
                Argc -= argcnt, argv += argcnt;
                if (argc == 0 && around)
                        break;
                if (argc > 0) {
                        close(0);
                        if (freopen(argv[0], 'r', stdin) == NULL) {
                                perror(argv[0]);
                                exit(1);
                        }
                        name = argv[0];
                        argc--, argv++;
                } else
                        name = 0;
                if (around)
                        putchar('\n');
                around++;
                if (Argc > 1 && name)
                        printf("==> %s <==\n", name);
                copyout(linecnt); /* copyout(B):<function> Outputs
                                                first B lines of stdin */
                fflush(stdout);
        } while (argc > 0);
}
```

副作用を与える

ユーザ定義関数

図1　aタイプのソース

```
/* func(S):<note>     S denotes string containing 'option indicators   */
/*                    ( -1 -2 ... -X )', where X must be integer.      */
/*          <function> Searches 'option indicators' and counts its     */
/*                    number (Y) that is a return value.               */
/*          <affect>  Assigns last integer (X) to 'linecnt'.           */
/*                    Also, argc = argc - Y .                          */
/*                    If X is not integer the execution will quit.     */
```

図2　a1（英語のコメント）

渡された文字列から、－(ハイフン) の後に整数が続くパターンを、これに当てはまらないパターンが来るか、文字列の終わりになるまでくり返し探し、最後の"－整数"のパターン内の整数部を共通変数linecntに設定する。
調べた"－整数"のパターンの個数をリターン値として返す。
－(ハイフン) の後に整数以外のものが続いていたときには、エラーメッセージを出力し実行を中断する。

図3　a2（日本語の説明）

```
#include <stdio.h>

extern int  linecnt = 10;
extern int  argc;

func(ap)
        char *ap[ ];
{
        register int i, n;
        register char *cp;

        for(n = 0; argc > 0 && ap[0][0] == '-'; n++) {
                cp = ap[0] + 1 ;
                for (i = 0; *cp >= '0' && *cp <= '9'; cp++)
                        i *= 10, i += *cp - '0';
                if (*cp) {
                        fprintf(stderr, "Badly formed number\n");
                        exit(1);
                }
                linecnt = i ;
                argc--, ap++;
        }
        return(n);
}
```

図4　a3（ソースコード）

```
#include <stdio.h>
#define N 256

main(argc,argv)
char **argv;
{
        char line[N];
        FILE *input;
        register i,c;

        input = stdin;
        do {
                if(argc>1) {
                        if((input=fopen(argv[1],"r"))==NULL) {
                                fprintf(stderr,"cannot open %s\n", argv[1]);
                                exit(1);
                        }
                }
                for(;;){
                        for(i=0;i<N;i++) {
                                line[i] = c = getc(input);
                                switch(c) {
                                case EOF:
                                        goto eof;
                                default:
                                        continue;
                                case '\n':
                                        break;
                                }
                                break;
                        }
                        while(--i>=0)
                                putc(line[i],stdout);
                        putc('\n',stdout);
                }
eof:
                fclose(input);
                argc--;
                argv++;
        } while(argc>1);
}
```

図5　b1（aタイプと同行数）

```
#include <stdio.h>
#define N 256

main(argc,argv)
char **argv;
{
        char line[N];
        FILE *input;
        register i,t,b,c;
        int sflag,j;

        input = stdin;
        do {
                if(argc>1) {
                        if((input=fopen(argv[1],"r"))==NULL) {
                                fprintf(stderr,"cannot open %s\n", argv[1]);
                                exit(1);
                        }
                }
                for(;;){
                        sflag=t=b=0;
                        for(i=0;i<N;i++) {
                                line[i] = c = getc(input);
                                switch(c) {
                                case EOF:
                                        goto eof;
                                case '\t':
                                        if(!sflag){
                                                t++;
                                                continue;
                                        }
                                case ' ':
                                        if(!sflag){
                                                b++;
                                                continue;
                                        }
                                default:
                                        sflag++;
                                        continue;
                                case '\n':
                                        break;
                                }
                                break;
                        }
                        j=b+t;
                        while(--b>=0)
                                putc(' ',stdout);
                        while(--t>=0)
                                putc('\t',stdout);
                        while(--i>=j)
                                putc(line[i],stdout);
                        putc('\n',stdout);
                }
eof:
                fclose(input);
                argc--;
                argv++;
        } while(argc>1);
}
```

図6　b2（b1の拡張）

以下の質問に答えて下さい。

(1) このプログラムの主機能を一行に要約せよ。

---

(2) ａｒｇｃｎｔに０が返されるようなアーギュメントを推測せよ。

---

(3) （ Ａｒｇｃ＞１ ＆＆ ｎａｍｅ ） が真になり、ｐｒｉｎｔｆ（．．．）が実行されるのはどのようなアーギュメントを指定したときか。

---

(4) ａｒｇｃ と Ａｒｇｃ の役割り（関係）を述べよ。

---

(5) ａｒｏｕｎｄ＋＋ は何をカウントしていると考えられるか。

---

(6) （ ａｒｇｃ＝＝０ ＆＆ ａｒｏｕｎｄ ） が真になり、ｂｒｅａｋが実行されるのはどのようなアーギュメントを指定したときか。

---

(7) ｗｈｉｌｅ（ ａｒｇｃ ＞ ０ ） の判定によりループを抜けるのはどのようなアーギュメントを指定したときか。

---

(8) このプログラムに許されるアーギュメントのパターンを全て列挙せよ。

図７　ａタイプの質問

以下の質問に答えて下さい。

(1) このプログラムの主機能を一行に要約せよ。

---

(2) ｆｏｒ（；；）ループを抜けるのはどの様な場合か。

---

(3) ｄｏループを抜けるのはどの様な場合か。

---

(4) ｐｕｔｃ（ｌｉｎｅ［ｉ］，ｓｔｄｏｕｔ）が実行されるのはどの様な場合か。

---

(5) このプログラムを実行させ、″ ａ＼ｎａ ＼ｎ″を入力すると、出力はどうなるか。

---

(6) ″＼ｔａ＼ｔ＼ｎ″，″＼ｔ＼ｔａＥＯＦ″の場合はそれぞれどうなるか。
（＼ｔはタブ文字、ＥＯＦはファイルの終了を示す文字）

---

(7) 出題のソースを入力すると、どの様な出力を得るか。

---

(8) （7）の出力を入力すると、どの様な出力を得るか。

図８　ｂタイプの質問

・機能概要
・ループの終了条件
・文の実行条件
・変数の意味
・可能な入力
・入力を与えたときの出力

等を問うものを与えた。各問につき、正解２点、半解１点、誤解及び無回答０点で採点し、最高は１６点である。採点の基準は、予備実験等を参考にして、あらかじめ決めておいた。この点数を理解度として分析評価に使っている。

被験者の作業順序は、以下の通りである。

１．経験に関するアンケートに答える。
２．ａタイプ中の１つを解読する。
３．作業に関する簡単なアンケートに答える。
４．ｂタイプ中の１つを解読する。
５．作業に関する簡単なアンケートに答える。

これらの作業を９０分位で行った。途中の休息時間は与えなかったので、全ての作業が終了するまで、被験者間で作業内容について検討する機会は無かった。

ａタイプとｂタイプの組み合わせは、片寄らないように前もって用意しておき、実験当日も、なるべく隣合わせた人が同じ問題を聞くことのないように配慮した。

３．人間要因と理解度

理解度の分布を図９に示す。ａタイプの中ではソースコードを与えたもの（ａ３）が一番理解度が高かった。ｂタイプでは拡張による理解度への影響はでなかった。ａタイプとｂタイプを比べると、ｂタイプの方が理解度が高かった。この結果を人間要因［３］を含めて評価した結果を以降に示す。

３．１　経験因子の有意性分析

因子分析により４つの経験因子から、ソフト開発経験とＣ言語プログラム解読経験を選び、理解度との間で重回帰分析を行った。その結果、ａタイプ、ｂタイプそれぞれの場合の重回帰係数が０．３６、０．１１で、これらの因子だけでは理解度を表わすのに不十分とでた。

３．２　高得点者の分析

図９でもわかるように、一部に非常に高い理解度を示している人達がおり、他とは異なる特徴を持っていると予想される。その比較を表１に示す。理解度の平均では、ａタイプで７倍、ｂタイプで３倍程の違いがある。この差に意味があるかを共分散分析により調べた。この結果、Ｆ値がａ：１１３．７６、ｂ：９３．０５で、共に自由

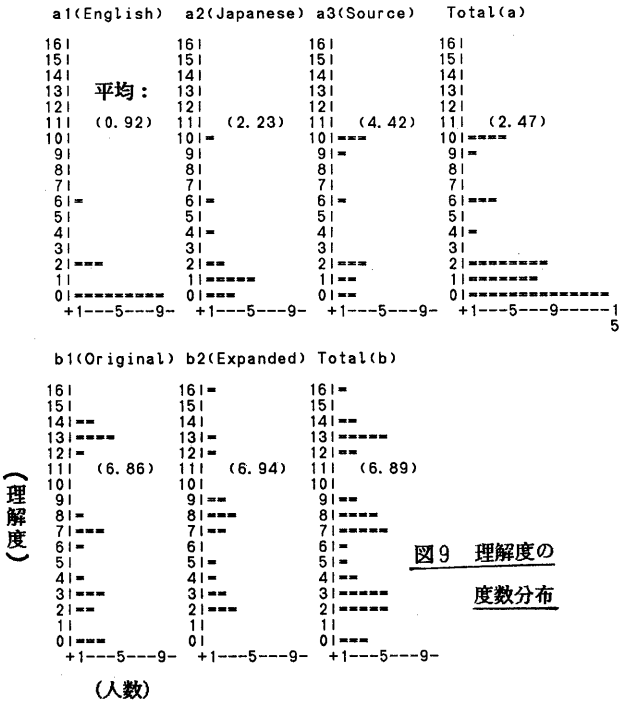

図9　理解度の度数分布

表1　理解度の異なるグループ間の比較表

	グループ分けの条件	人数	平均 理解度	ソフト開発経験	C作成経験	C解読経験	他言語の経験
a	8以上	5	9.8	2.8	3.2	3.6	2.8
	8未満	33	1.36	2.9	2.4	2.2	2.6
b	12以上	10	13.3	2.8	3.0	2.8	2.9
	12未満	28	4.6	2.9	2.3	2.3	2.5

表2　環境の異なるグループ間の比較表

	aタイプの理解度		bタイプの理解度		ソフト開発経験の平均	C解読経験の平均
	平均	分散	平均	分散		
部門1	1.81	2.56	5.38	4.22	2.88	2.31
部門2	3.98	4.38	10.17	3.88	2.82	2.58

度（1，34）でのF分布1%値：7．44より大きく、明らかに差があることが言えた。次にソフト開発とC言語プログラム解読の経験だけをみても差があるかどうか調べたところ、aについてのみ有意性（5%以下の危険率）がみられた。

これより、「平均理解度の低い（aタイプのような）プログラムでは、高い理解度と経験は強く結びつく」と考えられる。

3．3　部門間の差異分析

被験者を募った2部門は作業内容や作業体質等も異なっており、こういったものも影響を及ぼすと考えられる。表2に、これら部門間での理解度の比較を示す。

理解度の平均では、部門2の方がaタイプ、bタイプ共に倍近く高い。共分散分析の結果、F値がa：2．96、b：10．36で、自由度（1，34）でのF分布5%値：4．13より小さいaタイプの場合、差があるとは言えなかった。また、経験だけをみても部門間で差があるとは言えず、経験と理解度との間にも強い相関はみられなかった。

これより、「平均理解度の低い（aタイプのような）プログラムでは、経験以外の要因も余り作用しない」と考えられる。

以上のことから、aタイプのようなプログラムを『理解することが難しいプログラム』と呼び、bタイプのようなプログラムを『理解することが容易なプログラム』と呼ぶこととする。また、aタイプ中のa1、a2、a3は、平均理解度からこの順に難しいとし、bタイプのb1、b2については同等と考える。

4．質問と理解度

各質問ごとの正解率を図10に示す。これを、質問の種類によって分類したものを以下に示す。

1）aタイプの場合
- ・機能概要（1）……24%
- ・ループの終了条件（7）……7%
- ・文の実行条件（3，6）……4%
- ・変数の意味（4，5）……18%
- ・ある条件を満たす入力（2）……33%
- ・一般的な入力（8）……12%

2）bタイプの場合
- ・機能概要（1）……39%
- ・ループの終了条件（2，3）……64%
- ・文の実行条件（4）……39%
- ・部分的な入力に対する出力（5，6）……44%
- ・一般的な入力に対する出力（7，8）……27%

1）、2）を比べると、〝ループの終了条件〟に対する正解率に大きな差が出ている。高かった方のソースコード上の特徴として、無限ループ中の一ケ所からｇｏｔｏ文で飛び出しており、その判断に使った変数に何が入っているかが明白等が上げられる。1）の〝文の実行条件〟に対する正解率は、1）の中でも、2）と比べても特に低い。この特徴として、条件判断に複数個の変数を

使っていることが上げられる。数値の上からは（3）（6）共に同じであるが、解答内容をみると、（3）では惜しい解答があるのに、（6）では全然見られない。これは、（6）の場合、使われた個々の変数の意味が独立している為ではないかと考えられる。

このように、変数の意味や使われ方を理解する作業は難しいといえる。

1）、2）共に、限定された入力や出力に対する質問に比べて、一般的なものは正解率が低くなっている。これより、部分的であっても、入出力例はプログラム理解の助けになると考えられる。

5．理解度の予測

これまで理解度と様々の要因との関連について述べてきたが、重要なことは、いかにして作業者の理解度を前もって知るかである。

経験からは予測することが出来なかった。そこで、2種類のプログラムに対する理解度間である程度相関があることに注目し、bタイプの理解度と、aタイプの難易度（ここでは平均理解度の順位）で、aタイプの理解度が表せるか重回帰分析を行った。

（aの理解度）＝α＋α'（bの理解度）
＋α''（aの難易度）＋ε

その結果、上記重回帰式のF比10．35を得た。これは、自由度（2，33）でのF分布1%値：5．29より大きく、当式の有意性を表している。寄与率が0．4と低いので十分ではないが、「理解度（aタイプの理解度）を他のプログラム解読実績（bタイプの理解度）と対象の難易度（aタイプの難易度）である程度予測することが可能」と考えられる。

6．自己評価と実際

解読作業の後で答えるアンケート中に、作業に対する自己評価を行う項目を入れておいた。この評価と実際との比較を図11に示す。

作業者の評価は概ね当たってはいるが、3分の1位は実際より楽観的にとらえている。誤解しているのに、本人は正しいと思っている為だと考えられるが、aタイプとbタイプを比べると、理解することが容易なbタイプの方が強い傾向にある。

このことから、行った作業が多い（解答量が多い＝理解できたと感じた頻度が多い）程良くできたと思い易い傾向があると考えられる。

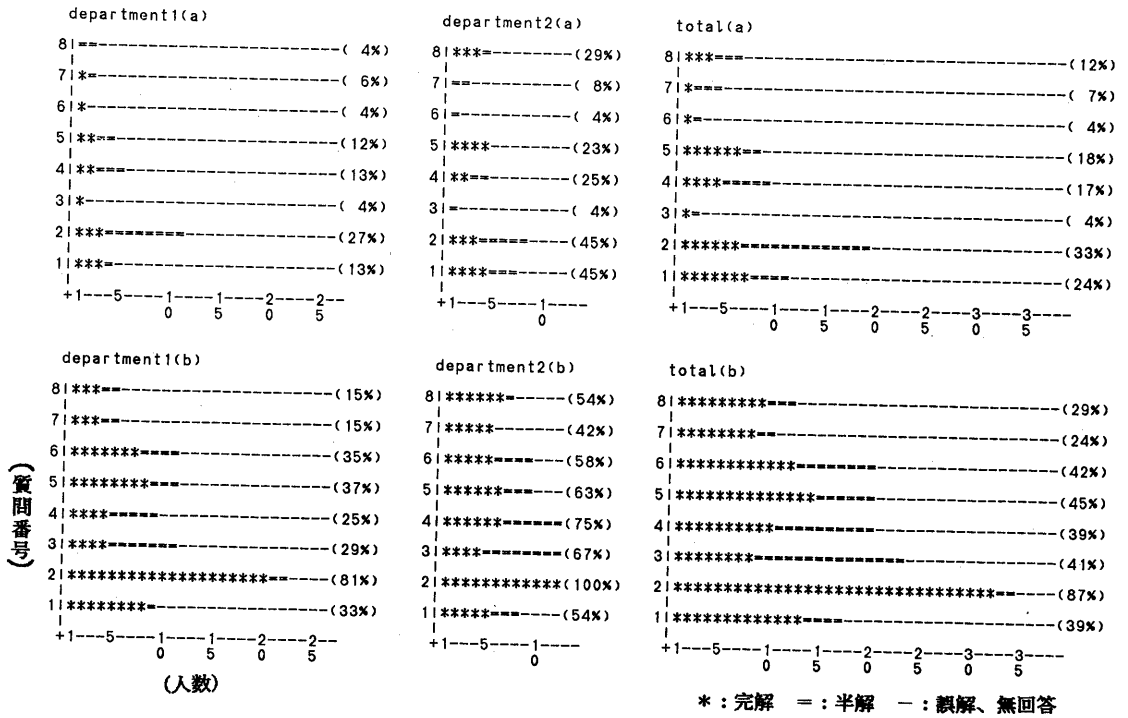

図10　質問ごとの正解率

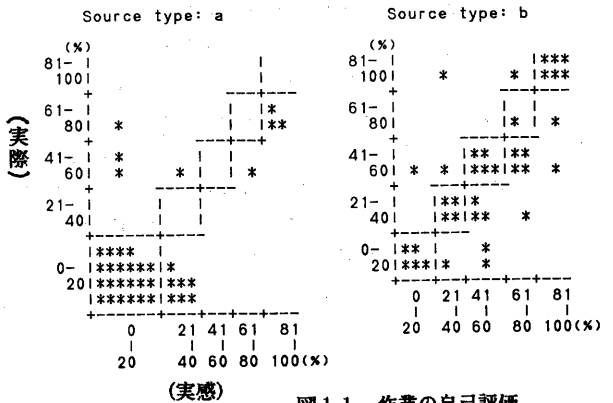

図11　作業の自己評価

7. まとめ

今回の実験から得られたことをまとめる。

人間要因との分析からは、以下のことが言えた。

1）プログラムの理解度と経験に余り相関はないが、理解することが難しいものを十分理解するには、十分な経験が必要である。また、理解することが容易なプログラムであるならば、必ずしも十分な経験を必要としない。

2）プログラムの理解度は、その人のプログラム解読実績と対象プログラムの難易度で、ある程度予測できそうである。

これは、理解することが容易なプログラムを作っておきさえすれば、それを理解するのに十分な経験を必ずしも必要とせず、その上以前の実績を見てどれ位理解できるか予測し易いという利点を示唆している。一方、理解することが容易なもの程出来た気になり易いという、作業者が陥りそうな落とし穴も示唆した。

質問と正解数の分析を行うことで以下のことが言えた。

3）条件式中に複数の変数が使われ、それらが一見無関係と思われるとき、その条件判断の意味を理解することは極めて難しい。

4）入出力例はプログラムの理解を助ける。

これらは、理解し易いプログラムを作る為の1つの基準として使うことができる。

aタイプでは、ソースコードを与えたもの（a3）が一番理解度が高かったが、与えるソースコードの量が多い場合についても調べてみる必要がある。また、コメントやドキュメントは、その質について追求する必要がある。呼び出し側に副作用を与える関数を使わない方が容易に理解できたが、これ以外にも、変数やシステム関数が及ぼす影響についても調べてみる必要がある。

8. あとがき

本論文では、ソフトウェアの保守には欠かすことのできないプログラムの理解について比較対照実験を行い、人間要因との関わり合いを中心に分析評価した結果を報告した。これにより、理解することが容易なプログラムを作ると、余り経験のない人でも経験のある人と同程度に、効果的な保守を行えそうだということを示すことができた。

プログラム理解の難易度については、順位尺度でしか与えられなかったが、事前に知ることができるような尺度を、プログラムの複雑さ等との関連も含めて追求していくことが今後の課題である。不透明なソフトウェアの問題を解決する為には、今後もこのような実験を行い、経験だけでなくデータに基づいた方向付けがなされてゆくべきだと考える。

最後に、本実験を行うに当たり指導して頂いた日本電気（株）ソフトウェア生産技術研究所寺本雅則氏を始め、実験に協力して頂いた多くの方々に感謝します。

参考文献

1）三浦、富田、尾関他　"ソフトウェア保守技術"、bit増刊ソフトウェア評価技法、第3部、（1982）130－174.

2）木村　"ヒューマンファクタ"、第6回ICSE講習会資料、（1982）207－229.

3）"ソフトウェアエンジニアリングに関する調査－ソフトウェア生産における人間要因－"、ソフトウェアエンジニアリング専門委員会報告書、日本電子工業振興協会、1983

4）Moher, T., & Shneider, G. M. "Methods for Improving Controlled Experimentation in Software Engineering", Proc. 5th ICSE(1981) 224-233.

5）Sheppard, S. B., Kruesi, E., & Curtis, B. "The Effects of Symbology and Spatial Arrangements on the Comprehension of Software Specification", Proc. 5th ICSE (1981)207-214.

6）Shneiderman, B. "SOFTWARE PSYCHOLOGY : Human Error in Computer and Information Systems", Winthrop, Cambridge, MA, 1980.

7）Shneiderman, B. "Control Flow and Data Structure Documentation : Two Experiments", Comm. ACM, 25, (1982)55-63.