ヒューマンインターフェース 43－1
（1992. 7. 9）

日本語プログラムの可読性の評価と検討

中川正樹，早川栄一，玉木裕二
東京農工大学工学部

〒184 東京都小金井市中町 2-24-16

我々は，2バイト固定長日本語文字コードの計算機システムを研究開発環境とし，その上で日本語プログラミングを実践し，その効果の評価を始めた．なお，本稿で言う日本語プログラミングとは，曖昧性を有する自然言語で処理（論理）を記述するのではなく，文字種として母国語が制限なく使用できる既存プログラミング言語によるプログラミングを意味する．この環境で相当規模のソフトウェアを開発してきた．日本語プログラミング環境は，概念設計からプログラムに至る段階的詳細化を可能にした．仕様書におけるキーワードは最終的プログラムに識別子などの形で反映されている．また，仕様書とプログラムの一体化は保守性にも寄与している．一方で，プログラムの可読性評価実験を行った．その結果，日本人にとっては日本語プログラムの可読性が高いことを検証した．長期的な視点に立てば，各民族が母国語でソフトウェアを開発できることが重要であり，それらの翻訳を妨げないシステムアーキテクチャが不可欠となろう．

Experiments and Discussions on the Readability of Programs in Japanese for Japanese

Masaki Nakagawa, Eiichi Hayakawa, Yuji Tamaki
Dept. of Computer Science, Tokyo Univ. of Agriculture and Technology

2-24-16 Naka-cho, Koganei, Tokyo, 184, Japan

A homemade systems software of the double-byte fixed-length Japanese character code has been providing a programming environment to practice programming in Japanese. Here, programming in Japanese does not imply "programming" in the ambiguous natural language but the liberation of the use of the Japanese language in the largest scope of a programming language grammar. A considerable amount of software has been developed on this system. Smooth stepwise refinement from specifications to programs has been realized. Keywords in specifications remain as functions, data names and so on in programs. Coherence between specifications and programs eases maintenace. To evaluate this, experiments were made to test whether programs are more readable in Japanese than in English for Japanese people. The results prove so. From a long term viewpoint, it should be important that all people can program in their mother tongue and that the systems architecture does not hinder the translation of software between languages.

1．はじめに

プログラミングは高度に知的な活動である．プログラムを書くときは，効率よく経済的なデータ構造と処理を考えなければならない．また，読むときは，モジュールの構造と関係をよく理解しなければならない．重要な点は，こうした思考が我々の母国語に深く結びついていることである．

また，計算機は，数値計算などの万国共通のものから，ますます，社会によって固有の民族的，地域的，文化的，あるいは，経済的活動に利用されるようになってきている．そのためのソフトウェア開発が必要になる．その際，母国語でしか表現しづらい概念や用語が多々現れる．こうした対象を扱うプログラミングでは，当然，母国語が使えたら便利であり，これをすべて英文字で表現しなければならないと，名前付けや不自然な英文字化（日本の場合はローマ字化）のためにプログラム作成の思考に干渉を与える．

さらに重要なことは，ソフトウェアが，本や論文と同等に知的財産を形成しつつあることである．プログラムを読む比重はますます大きくなりつつある．近年，ソフトウェア工学においては，より多くの労力が新規開発ではなく，保守につぎ込まれるようになった．プログラムを書くことより，ますますプログラムを読むコストの方が高くなりつつある．プログラムの可読性は，記述生産性に勝るとも劣らない重要なファクタとして認識されなければならない．

ソフトウェア工学の分野では，品質や生産性を向上させるために，ツール，方法論，パラダイム，環境など様々な提案がなされ，そして，実践されてきた．しかし，非英語圏の人間にとって，母国語の使用が制限される環境では，それらが効果を発揮する以前に大きな問題がある．自然言語を全く排除し，万国共通の形式言語だけでソフトウェアを開発しようとする試みもあるが，ソフトウェア開発の全過程で母国語を排除するトレードオフを評価する必要がある．そのような試みが可能としても，ソフトウェアの全生産における比率は零に近い．

プログラミング言語を日本語化した例としては，COBOL[7]，Pascal[3]，SNOBOL[14]，CLU [2]などがある[11]．しかし，日本語化の効果については報じていない．さらに，一言語，一アプリケーションでの日本語化には限界がある．これは，他アプリケーションとの組合せや他言語プログラムとのリンケージの必要性を考えれば明らかである．例えば，ある言語で日本語識別子が使えても，リンカやソフトウェアツールが対応していなかったら，日本語識別子を使うことは逆に障害になる．我々は，オペレーティングシステムレベルからの完全な日本語化を最初に報告している[4]．

OS レベルから日本語化しても，それが不完全なら一時しのぎにはなっても，かえって問題を複雑にする．商用システムでは，既存システムのコード系に日本語文字コードを混在させて日本語を表現している．しかし，そのために文字列処理が複雑になったり，日本語の使用が制限されることになる．また，アプリケーションの国際化，地域化を遅らせる要因になっている．最近，米国製 OS がマルチバイト化している理由がここにある[12]．

多字種文字を扱うのに，マルチバイト固定長コードを採用すれば，計算機システムの母国語化は系統的に実現できる[10，15]．母国語が使えれば，ソフトウェア開発の生産性や品質の向上が期待できる．さらに，固定長コード体系間ならば，ソフトウェアを翻訳するのはプログラミングの問題ではなくなり，ソフトウェア技術者の労力を必要としない[10]．以上の帰結として，各民族が優れたソフトウェアを自分達のために円滑に相互利用できるようになるはずである．

以下，２章では，日本語プログラミング環境の実現方針を述べる．３章では，日本語プログラミングの現実的な効果の一端を，４章で日本語プログラムに関する一連の評価実験を提示する．５章では，日本語プログラミングの本質的利点について論じる．

2．計算機システムの日本語化

上記の理由から，日本語が制限なく使える計算機システムを開発した[5，6]．特に，そのシステム記述言語Ｃにおいて，識別子，メッセージ，コメントなどを含めて，＜文字＞を日本語文字，英字などを包含する JIS コードセットに拡張した．ここで言う“日本語プログラミング”とは，曖昧性を有する自然言語で記述するのではなく，プログラミング言語の文法の範囲内で，日本語の使用を自由にすることを意味する．

２バイト固定長 JIS コードの計算機システムを作成するために，あるコード体系のシステムから別のコード体系のシステムを生成する，一般的で系統的な手法を考察した．ASCII から JIS への変換を例にして述べる．まず，システム記述言語処理系の文字型を，１バイトから２バイトに変換し，ASCII 文字セットから日本語文字セットに拡張する．そして，システムソフトウェアのすべてのソースファイルの文字コードを，ASCIIコードから２バイト固定長JIS コードにコード変換する．必要に応じて，文字，文字列，メッセージ，コメント，識別子などを日本語に翻訳する．処理やデータ構造の変更の必要はない．最後に，日本語化したシステム記述言語処理系で再コンパイルする．

この過程で，システム記述言語処理系の日本語化が鍵となった．我々はこれについても，上記処理系ソースプログラムの最小修正で系統的に日本語化を行った．

混在コード系よりもむしろ，マルチバイト固定長コード系の方が，簡単にしかも系統的に母国語化できる．この方式は，システム記述言語処理系のソースプログラムさえあれば，我々のシステムに限らず適用できる[5，10，15]．

3．日本語プログラミングの実践

現在，研究室では，システムプログラミングとアプリケーションの区別なく日本語プログミングを実践している．ここでは，１つのアプリケーション（オンライン手書き日本語文字認識システム）を例にとり，日本語プログラミングの実際的効果を考察する．これを取り上げる理由は，日本語がコメントにしか使えない過渡的環境で以前作成した約 15,000 行のプログラム（不完全日本語版）と比較できること，それ自身（完全日本語版），30,000行を超えているのである程度の統計的性質が言えること，である．

完全日本語版の研究開発では，概念仕様書から，設計仕様書，関数仕様書，および部分的ではあるが PADによる図示表現を経て，日本語プログラムへと詳細化した．仕様書におけるキーワードは最終的プログラムに関数名，変数名，データ構造名などの形で反映されている．また，仕様書とプログラムの一体化は保守性にも寄与している．

完全日本語版と不完全日本語版の比較を行った．この結果，コメントがどこでどのように付けられているかに顕著な差異が現れた．結果を表１に示す．不完全日本語版では，プログラムの右に，その処理の説明（言い換え）としてコメントが付けられているのに対し，完全日本語版では，コメント文字の 90%以上が関数やモジュールの先頭で仕様を記述するのに使われていた．日本語での仕様記述の段階的詳細化が関数などの仕様記述に引き継がれたこと，プログラミングでの補足的説明が日本語だと苦なく行えることが，書けと言われてもなかなか書かない仕様記述につながったと言えよう．一方，プログラムの右で処理の説明のために使わるコメントが圧倒的に減ったのは，日本語で識別子に名前付けできれば，処理内容をコメントとして日本語で言い直す必要がなくなったためと考えられる．

4．日本語プログラムに対する評価実験

4.1 可読性の予備実験

日本語プログラムの可読性に関して，簡単な予備実験を行った[9]．まず，コメントを全然付けない２種類のプログラムを用意した．一方は，コメント計数に関するもの（約 150行），他方は，キャッシュに関するもの（約 100行）である．それぞれについて，さらに，日本語によるもの，英語によるものを用意した．それらを略称で，コ＿日，コ＿英，キ＿日，キ＿英，と記す．被験者を２つのグループ G1 ，G2に分け，G1 には，コ＿英とキ＿日， G2 には，コ＿日とキ＿英を与えた．

表１．コメントの割合の変化

目的 ／ 版	ｺﾒﾝﾄ文字数 ／総文字数	仕様記述 ／総ｺﾒﾝﾄ	処理記述 ／総ｺﾒﾝﾄ
不完全日本語版	４５％	６０％	４０％
完全日本語版	３４％	９１％	９％

表２．プログラムと言語と被験者の組合せ

処理 ＼ 言語	日本語Ｃ → 被験者ｸﾞﾙｰﾌﾟ	英語Ｃ → 被験者ｸﾞﾙｰﾌﾟ
コメント計数	コ＿日 → Ｇ２	コ＿英 → Ｇ１
キャッシュ	キ＿日 → Ｇ１	キ＿英 → Ｇ２

被験者には，それぞれのプログラムについて，読むのに要した時間，理解を確認するための設問に答えるのに要した時間を記録するように指示した．

設問に正しく答えた被験者について，それぞれのプログラムを読んで理解するのに要した合計時間をグラフにしたのが図１，図２である．

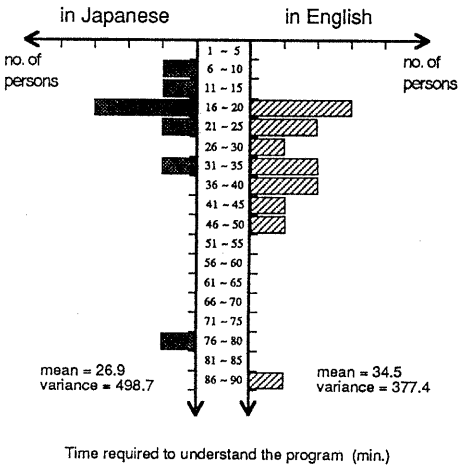

図１．コメント計数の解読時間の分布

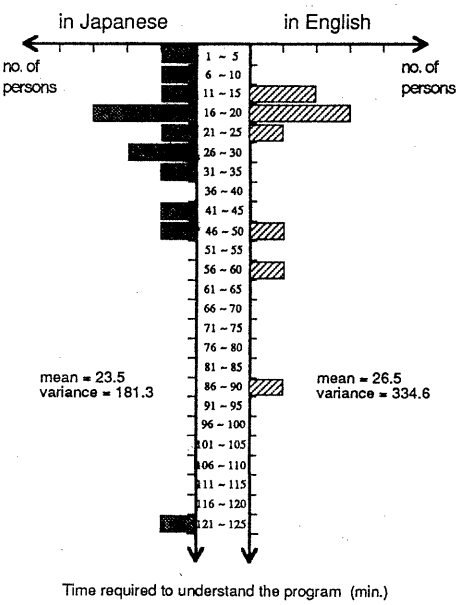

図２．キャッシュの解読時間の分布

両方のプログラムで日本語の方が英語より短時間で理解できていれば，被験者によらず日本語の方が読み易いと言える．しかし，データからはそう結論することはできない．コメント計数プログラムにおける日本語の優位性は言語の差ではなく，被験者の能力差である可能性を否定できない．

しかし，これとは反対の解釈をすることにより，いくつかの仮説が立てられる．まず，すべての場合ではないにしても，日本語プログラムの方が読み易いと言える可能性がある．もしそうなら，コメント計数では状態遷移を追う必要があり，可読性の差が出た．一方，キャッシュの方は処理が典型的なため，深く追わなくても済み，日本語，英語で差が出なかったのではないだろうか．

4.2 デバッグ効率の評価の試み

プログラムのデバッグし易さに，日本語プログラムと英語プログラムで違いがあるか実験を試みた．デバッグ効率はソフトウェアの品質に影響する重要な問題である．評価実験では，間違いを含んだプログラムを2種類用意した．一方は木構造を扱うプログラムで，空ポインタの処理に虫を入れた．他方は数式を逆ポーランド記法に変換するプログラムで，項と因子が連続する場合の処理に虫を混入した．それぞれに日本語と英語の版を用意した．それらを2つのグループの被験者に組合せを互い違いにして提示し，間違いを指摘して正しく直してもらうのに要する時間を計測するというものであった．図3，図4に問題の一部とその正しい訂正例を示す．

ところが，紙面上の実験では，日本語，英語によらず正しく虫を見つけ訂正する人は非常に少なく統計をとれなかった．この解釈として，紙上実験に問題があったと考えている．この形式だと試行錯誤ができないし，実行の確認もできない．しかし現実には，数回の試行を繰り返してデバッグすることに慣れている．だとすれば，実行が確認できるプログラミング環境でユーザの行動を計測する必要がある．

4.3 実験の反省

詳細はここでは述べないが，上のデバッグの実験と同時に，プログラムの可読性の実験をさらに1つのプログラムの組について試みた．一方は，コマンドのオプション解析のプログラムで，他方は 16 進文字列を数値に変換するプログラムであった．前者では正解が少なく統計がとれなかった．後者では，日本語の方が正解が多いという結果になったものの，読解時間の平均に有意差が出なかった．紙上実験の問題と，実験課題の悪さが重なって出てしまった結果となった．

現実のプログラムの保守では明らかに日本語の方が読み易いのに，実験ではうまく検出できない理由を考察してみた．それは，先の実験でも言えることだが，アルゴリズムの例題的プログラムでは，日本語か英語かの差は出にくい可能性が

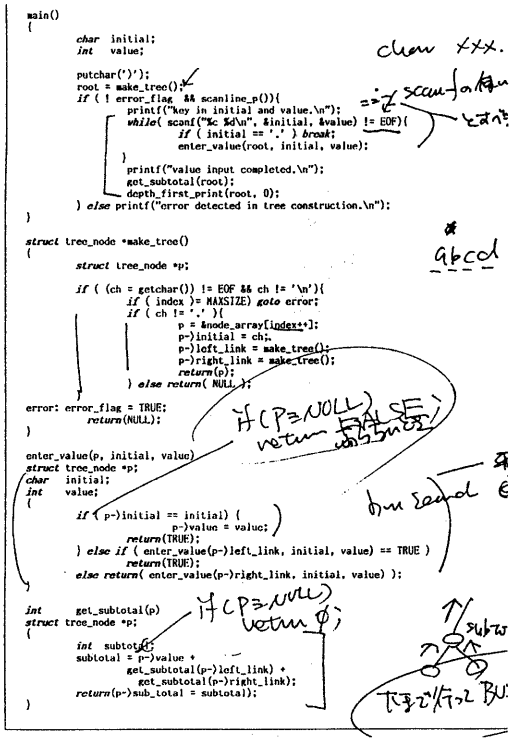

図3．木構造を扱うプログラム（英語版）の訂正例

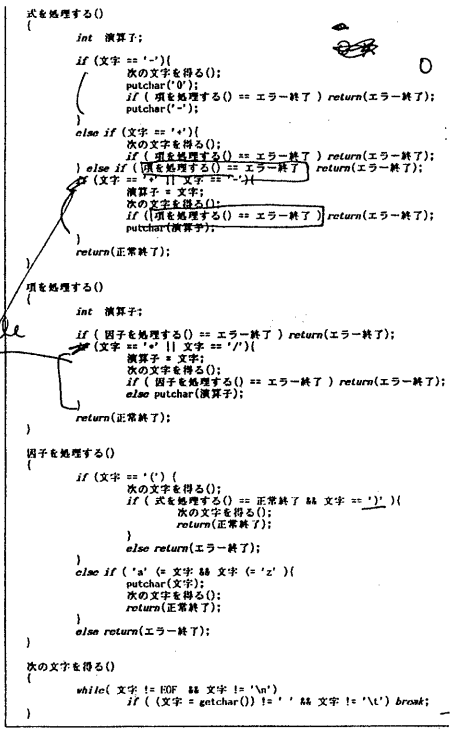

図4．逆ポーランド変換プログラム（日本語版）の訂正例

ある．それらのプログラムではアルゴリズム理解の比重が高く，言語の比重は小さい．ところが現場では，アルゴリズムの理解というより処理の理解が大半になる．状態遷移を追跡する，大域変数を頭において各モジュールの動作を読む，モジュール間のデータのやり取りを確認する，などに大半の時間を要することになる．そうだとすれば，実際に引き継ぐプログラムで評価する必要がる．

さらに，アルゴリズムの例題的プログラムでは，知っている者と知らなかった者で読解時間が極端に分離してしまうという問題もある．

しかし，実際に引き継ぐプログラムでも，大規模なプログラムは評価実験には適さない．そこで，実用されているプログラムからまとまりのよい部分を取りだして，例題を作成することにした．

```
int     数字列と後置語を検索する( テキストファイル先頭, 後置語 )
char    *テキストファイル先頭, *後置語 ;
{
        char    *テキスト_pt ;
        int     文字数カウント ;

        テキスト_pt = テキストファイル先頭 ;
        while( テキスト_pt = 文字列を検索する( テキスト_pt, 後置語 ) ) {
                for(
                        文字数カウント = 0 ;
                        (テキスト_pt > テキストファイル先頭)
                        && ((*(テキスト_pt - 1) >= '0') && (*(テキスト_pt - 1) <= '9')
                        テキスト_pt-- , 文字数カウント++
                )
                        ;
                if( 文字数カウント > 0 ) {
                        登録する( テキスト_pt, 文字数カウント += strlen( 後置語 ) ) ;
                        テキスト_pt += 文字数カウント ;
```

ファイル名： search2.c　　1992年　3月

```
                }
                else
                        テキスト_pt++ ;
        }
}

int     前置語と数字列と後置語を検索する( テキストファイル先頭, 前置語, 後置語 )
char    *テキストファイル先頭, *前置語, *後置語 ;
{
        char    *テキスト_pt, *テキスト_pt_保存 ;
        int     文字数カウント ;

        テキスト_pt = テキストファイル先頭 ;
        while( テキスト_pt = テキスト_pt_保存 = 文字列を検索する( テキスト_pt, 前置語
                for(
                        テキスト_pt += strlen( 前置語 ), 文字数カウント = 0 ;
                        *テキスト_pt && (*テキスト_pt >= '0' && *テキスト_pt <= '9')
                        テキスト_pt++ , 文字数カウント++
                )
                        ;
                if( 文字数カウント > 0 && ! strncmp( テキスト_pt, 後置語, strlen( 後置
                        登録する( テキスト_pt_保存, 文字数カウント + strlen( 前置語 )
        }
}

char    *文字列を検索する( テキスト_pt, 検索文字列 )
char    *テキスト_pt, *検索文字列 ;
{
        while( *テキスト_pt ) {
                if( ! strncmp( テキスト_pt, 検索文字列, strlen( 検索文字列 ) ) )
                        return テキスト_pt ;
                else
                        テキスト_pt++ ;
        }

        return NIL ;
}

int     登録する( テキスト_pt, 文字数 )
char    *テキスト_pt ;
int     文字数 ;
{
        if( インデクス >= 最大大 )
                return エラー ;

        文字列登録表[インデクス].文字列 = テキスト_pt ;
        文字列登録表[インデクス].文字数 = 文字数 ;

        インデクス++ ;
}

int     表示する( テキストファイル先頭 )
char    *テキストファイル先頭 ;
{
        int     i, j ;

        for(i = 0 ; i < インデクス ; i++) {
                for(j = 0 ; j < 文字列登録表[i].文字数 ; j++) {
                        putchar( *(文字列登録表[i].文字列 + j) ) ;
                }
                putchar('¥n') ;
        }
}


/*
※ 読み終ったら、今の時刻を書いて下さい。

※ 次の問題に答えて下さい。
(1) 上のプログラムは何をするプログラムですか。


(2) 入力ファイルとして次の内容が与えられてとき、上のプログラムの出力結果を示せ。

お年は幾つですか。3つです。妹さんは幾つですか。1つです。数100個の赤い玉があります。数
数10台の車があります。数100台の自転車もあります。¥0



※ 問題に答え終ったら、今の時刻を書いて下さい。

ありがとうございました。
*/
```

図５．文脈付き数字列検索プログラム（日本語版）の一部

```
int     search_for_number_and_postfix_char( head_of_textfile, postfix_char )
char    *head_of_textfile, *postfix_char ;
{
        char    *text_ptr ;
        int     character_counter ;

        text_ptr = head_of_textfile ;
        while( text_ptr = search_for_string( text_ptr, postfix_char ) ) {
                for(
                        character_counter = 0 ;
                        (text_ptr > head_of_textfile)
                        && ((*(text_ptr - 1) >= '0') && (*(text_ptr - 1) <= '9')) ;
                        text_ptr-- , character_counter++
                )
                        ;
                if( character_counter > 0 ) {
                        enter( text_ptr, character_counter += strlen( postfix_char ) )
                        text_ptr += character_counter ;
```

ファイル名： search2.c　　1992年　3月

```
                }
                else
                        text_ptr++ ;
        }
}

int     search_for_prefix_char_and_number_and_postfix_char( head_of_textfile, prefix_cl
char    *head_of_textfile, *prefix_char, *postfix_char ;
{
        char    *text_ptr, *saved_text_ptr ;
        int     character_counter ;

        text_ptr = head_of_textfile ;
        while( text_ptr = saved_text_ptr = search_for_string( text_ptr, prefix_char ) )
                for(
                        text_ptr += strlen( prefix_char ), character_counter = 0 ;
                        *text_ptr && (*text_ptr >= '0' && *text_ptr <= '9') ;
                        text_ptr++ , character_counter++
                )
                        ;
                if( character_counter > 0 && ! strncmp( text_ptr, postfix_char, strlen

                        enter( saved_text_ptr, character_counter + strlen( prefix_char
        }
}

char    *search_for_string( text_ptr, target_string )
char    *text_ptr, *target_string ;
{
        while( *text_ptr ) {
                if( ! strncmp( text_ptr, target_string, strlen( target_string ) ) )
                        return text_ptr ;
                else
                        text_ptr++ ;
        }

        return NIL ;
}

int     enter( text_ptr, no_of_characters )
char    *text_ptr ;
int     no_of_characters ;
{
        if( table_index >= MAX_SIZE )
                return ERROR ;

        string_table[table_index].string = text_ptr ;
        string_table[table_index].no_of_characters = no_of_characters ;

        table_index++ ;
}

int     print_out( head_of_textfile )
char    *head_of_textfile ;
{
        int     i, j ;

        for(i = 0 ; i < table_index ; i++) {
                for(j = 0 ; j < string_table[i].no_of_characters ; j++) {
                        putchar( *(string_table[i].string + j) ) ;
                }
                putchar('¥n') ;
        }
}


/*
※ 読み終ったら、今の時刻を書いて下さい。

※ 次の問題に答えて下さい。
```

図６．文脈付き数字列検索プログラム（英語版）の一部

4.4 可読性の評価実験

プログラムの可読性に関して再実験を行った．コメント計数プログラム（約 150行）は実際に学生が作成し数名が手を入れたものなので，これを問題の１つにした．もう一方は，表記をチェックするプログラムから抽出した．それは，指定された前後関係で出現する数字列を検出するプログラム（約 110 行）である．プログラム単独の可読性を評価する目的から，それらのコメントはすべて削除した．それぞれのプログラムについて，日本語で記述したものと英語で記述したものを用意した．英語版については英語を母国語とする修士学生に手を入れてもらった．これらを，コ＿日，コ＿英，数＿日，数＿英 とする．プログラムの断片を図５，図６に示す．

研究室の内外から学部４年生，大学院生を被験者とした．予備実験と同じコメント計数プログラムを題材にすることから，それから１年以上経過していたが，それとは被験者が重ならないようにした．被験者を G1 と G2 の２つのグループに分けた．さらに G1 を G1.1 と G1.2 に，G2 を G2.1 と G2.2 に分けた．グループ分けでは，過去のプログラム歴や研究分野を参考にして，能力や知識でグループ間に偏りを生じないよう配慮した．G1.1 の被験者には コ＿英 と 数＿日 をこの順に，G1.2 には同じものを逆順に，G2.1 の被験者には コ＿日，数＿英 をこの順に，G2.2 には同じものを逆順に与えた．被験者には，それぞれのプログラムについて，読むのに要した時間，理解を確認するための設問に答えるのに要した時間を記録するように指示した．表３に，設問に正しく答えた被験者について結果を示す．グループ間で被験者数に不均衡があるのは，正解が得られなかった被験者を除いたためである．プログラムを読む時間と，設問に答える時間を別個に見た場合，有意なことは言えない．これは，プログラムを十分理解して設問に答えるタイプと設問を見てからプログラムを見直すタイプの個人差が大きいためであろう．そこで両方の時間の合計で見ることにする．また，G1.1 と G1.2，G2.1 と G2.2 のそれぞれで，実験順序による有意差はなかったので，G1 と G2 にそれぞれ統合する．図７，図８に，以上の統合結果の読解時間分布を示す．

表３．プログラムの正しい読解に要した時間

グループ		被験者	研究分野	課題：数＿日 読み時間(分)	回答時間(分)	合計(分)	順序	課題：コ＿英 読み時間(分)	回答時間(分)	合計(分)
G1	G1.1	1	AP	7	8	15		8	8	16
		2	S	10	3	13		16	1	17
		3	S	15	*	15		40	1	41
		4	AP	11	2	13	←	20	4	24
		5	AP	7	9	16		18	12	31
		6	AP	22	*	22		30	19	49
		7	S	13	18	18		13	11	24
	G1.2	8	AP	17	3	20		19	12	31
		9	S	10	2	12	→	14	9	23
		10	AP	12	6	18		9	46	55
				課題：コ＿日				課題：数＿英		
G2	G2.1	11	S	11	10	21		30	5	35
		12	S	7	2	9		12	3	15
		13	AP	10	10	20		15	*	15
		14	S	2	9	31	→	29	4	33
		15	AP	16	3	19		30	1	31
		16	AP	20	*	20		23	6	29
	G2.2	17	S	10	5	15		20	20	40
		18	AP	9	2	11		16	2	18
		19	AP	35	*	35	←	44	*	44
		20	S	20	6	26		18	6	24

*: 読み時間と回答時間を分離して記録せず．
研究分野における，AP は application，S は systems software.

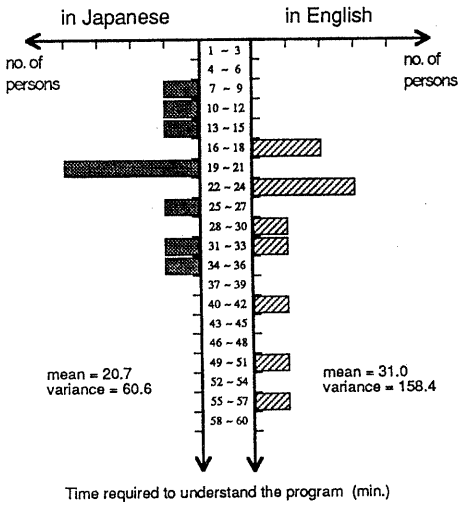

図７．コメント計数の解読時間の分布

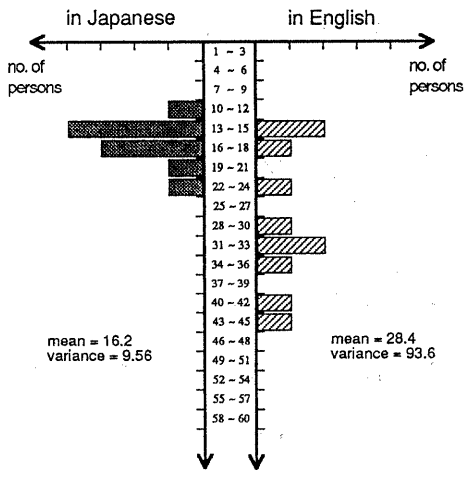

図８．文脈付き数字列検索の解読時間の分布

2つのプログラムのそれぞれについて，日本語のほうが英語より読み易いかどうか検定する目的で，統計量 t を計算し，自由度18 の t 検定を行なった．コメント統計のプログラムについては，t = 2.1 となり，2.5 % の危険率で日本語のほうが早く理解できると言える．数字列検出のプログラムについては，t = 3.6 となり，0.25 % の危険率で日本語のほうが早く理解できると言える．両方の結果から，被験者によらず，日本語プログラムの方が早く理解できる，つまり，読み易いと言える．

5．考察

5.1 プログラムの国際性

「プログラムは国際互換性のために英語で書くべき」との主張がある．しかし，非英語圏の者にとって，それでソフトウェアの生産性や品質が上がるだろうか．識別子に良い英語名が付けられるか．それを読む労力はどうか．結局，処理を逐一説明するコメントが必要になる．コメントを英訳しないでは理解し難い，“英文字列”プログラムができるだけではないだろうか．

ソフトウェアの国際互換性は重要ではあるが，それは，単にプログラムを英語で書けば済む問題ではない．ソフトウェアは，各種の文書を含んでいる．これらまで最初から英語で開発すべきか疑問である．

5.2 プログラミングと国際化の分離

認知心理学では，意識を要する複数の仕事は干渉することが知られている[1]．プログラミングを英語に限ることは，プログラミングと，英語で考える，あるいは，英語に直すことが干渉すると考えられる．だとすれば，最初から国際互換性を考えて不必要な制限を課すより，まず母国語でよいものをつくって，しかる後に翻訳を考えるべきである．

母国語プログラミングの本当の効果は，要求仕様から始まって，幾つかのレベルの設計仕様書を経てプログラムを開発する時に発揮される．文書にするか，プログラムにするかは，それがソフトウェア開発のどの段階にあるかによる．母国語による思考を英語にしなければならないのは，どの段階でも思考を中断させる．

5.3 プログラムの国際化・地域化

母国語でプログラミングすれば処理の言い換えのためのコメントはほとんどなくて済む．このプログラムの方が英訳し易いことも分かった．識別子やメッセージの英訳は，機械翻訳ではなくて，変換表で済む．そこで，日本語のトークンと対応する英語の対訳表を用意し，その変換を行なうツールを作成した．このツールを使って，OS/omicron上で開発した日本語プログラムをパーソナル計算機に移植し，利用している．固定長コード体系間ならば，文字コード長に依存したコーディングをしない限り，国際化や地域化はむしろ容易になる．

5.4 日本語プログラム入力の経済性問題とその解決

我々の研究室では，大半の学生が日本語でプログラミングするようになった．その理由として以下の項目をあげている．

(1) 英語で名前を考える必要がない．
(2) 仕様書からの詳細化が円滑である．
(3) プログラム作成には作成途中のリストを読みことが伴い，日本語の方が思考を発展させ易い．
(4) 保守が容易になる．

このことは，日本語プログラミングがソフトウェア開発者にとってインタフェースがよいことを示している．

一方，英語でコーディングすることを好む学生もいる．英語の方が入力が容易であるという理由による．確かに仮名漢字変換入力はアルファベット入力より手間を要する．もちろん，仮名漢字変換でも略記法を登録して少ない手数で入力効率をあげている学生もいる．

しかし英語入力を好む学生も，一時的に必要なプログラム以外は，先に述べた日本語プログラムと英語プログラム間の変換ツールを利用して，日本語に変換し保守するようになった．日本語プログラミングを不便にする技術的要因があるとしても，それらを改善するツールを作れば済むことである．

日本語プログラミングとは，何もすべてに日本語を使用すべきということではない．英字はJIS コードに含まれるため，英語で名前付けすることも可能である．ループカウンタなどには，英字で i, j のように書いた方が便利である．

5.5 母国語プログラミングか視覚プログラミングか

言語ではなくて視覚プログラミングに移行するのではないかという意見がある．後者の方が，分かりやすさやデバッグ効率，そして一部の問題では記述力においても優れた面がある．しかし，それだけである程度のソフトウェアができるとは考えにくい．

結局，モジュール間または外部とのリンケージは言語でとることになるのではなかろうか．それを言語でとるとすれば，母国語である方が覚え易いし，分かり易い．そのためには，サブシステムの母国語化は有効ではなくて，システム全体の母国語化が必要になる．パタンと言語の両方をうまく使いこなすのが人知の本質とすれば，視覚プログラミングと母国語プログラミングということになるのかもしれない．

6．おわりに

我々は，日本語が制限なく使用できる計算機システムを研究開発環境とし，その上で日本語プログラミングを実践し，その効果の評価を始めた．なお，本稿で言う日本語プログラ

ミングとは，曖昧性を有する自然言語でプログラムを記述するのではなく，プログラミング言語の文法の範囲内で，母国語を最大限に自由に使用することを意味する．

この環境で相当規模のソフトウェアを開発してきた．日本語プログラミング環境により，仕様書からプログラムへの円滑な詳細化が可能になった．仕様書におけるキーワードは最終的プログラムに識別子などの形で反映されている．また，仕様書とプログラムの一体化は保守性にも寄与している．さらに，プログラムの処理内容を日本語のコメントで言い換える必要がなくなり，反対に各関数やモジュールの仕様を記述することを助長している．

日本語プログラムの可読性の評価では，実験用の小さいプログラムを用意してその効果を検定した．現場では，日本語プログラミングはさらに高い効果が期待できる．大きいサイズのソフトウェアを保守するためには仕様書と並行してプログラムを読む必要があり，その場合，仕様書との一体性が有利なはずである．我々はこのことを実感している．ただし，評価は大規模にならざるを得ない．

日本語プログラミングの効果は，実際のソフトウェア工学のなかで評価する必要があり，そのためには，システム側でプログラマの行動を計測し，記録していく仕掛けが必要である．

ソフトウェアの国際性は重要ではあるが，その問題はプログラミングに限定される訳ではない．なぜなら，ソフトウェアは各種の文書を含むからである．これらまで最初から英語で開発すべきか疑問である．プログラムについては，その識別子やメッセージを日本語から英語へ，またその逆の変換を行えばよいことである．ソフトウェア全体については，最初から国際互換性を考えて不必要な制限を課すより，まず母国語でよいものを作って，しかる後に翻訳を考えるべきである．

長期的な視点に立てば，各民族が母国語でソフトウェアを開発できることが現実的に重要であり，それらの翻訳を妨げないシステムアーキテクチャが不可欠となろう．

謝辞

研究内容についてご討議頂く，高橋延匡教授，並木美太郎助手に深謝する．また，被験者として討論者として協力してくれた研究室の学生各位，英語プログラムの作成を助けてくれた，Rodney G. Webster 君に感謝する．

参考文献

[1] アンダーソン（富田他訳）：認知心理学概論，p.552，誠信出版，東京（1982）．

[2] 久野，佐藤，鈴木，中村，二瓶，明石：CLU マシンシステムの開発，情処学オペレーティングシステム研資 33-4 (1986)．

[3] 島崎真昭：日本語Pascal：パスカル，情処学第23回全大 1H-1, 215-216 (1981)．

[4] 鈴木，小林，田中，中川，高橋：OS/omicron における日本語プログラミング環境，情処学「コンピュータシステム」シンポジウム，11-18 (1987)．

[5] 鈴木，小林，田中，中川，高橋：OS/omicron における日本語プログラミング環境，情処学論，30，1，2-11 (1989)．

[6] 高橋延匡：研究プロジェクト総説：OS/omicron の開発，情処学オペレーティング・システム研資，39-5(1988)．

[7] 床分，今城：COBOL における日本語機能の現状と今後の動向，情処学プログラミング言語研資，16-9(1988)．

[8] The Unicode consortium: The Unicode Stadard - Worldwide Character Encoding Version 1.0(1991)．

[9] 中川，早川：日本語プログラミングのヒューマンファクタの一考察，1991信学会春季全大，D-72，6-72(1991)．

[10] 中川，玉木，早川，曽谷：母国語プログラミングへの方式，実践とその効果，情処学ソフトウェア工学研資，85-2(1992)．

[11] 中田育男：プログラミング言語における日本語化の現状と今後の動向，情処学プログラミング言語研資，16-6 (1988)．

[12] 日経エレクトロニクス：特集 国際版OS登場 パソコン・ソフトの国境が消える，日経エレクトロニクス，no. 550, 109-131 (1992.3.30)．

[13] 長谷川，岡崎：漢字 CP/M のコード体系，情処学マイクロコンピュータ研資，26-2 (1983)．

[14] 吉田，牛島：SNOBOL4 既存処理系への日本語テキスト処理機能の追加，コンピュータソフトウェア，2，3，88-98(1985)．

[15] Souya,T., Hayakawa, E., Honma, M., Fukushima, H., Namiki, M., Takahashi, N. and Nakagawa, M.: Programming in a Mother Tongue: Philosophy, Implementation, Practice and Effect, Proc. 15th IEEE COMPSAC., Tokyo, 705-712(1991)．