日本語プログラミングの実践とその効果

中川正樹[†] 早川栄一[†]
玉木裕二[††] 曽谷俊男[†††]

本論文では，日本語プログラミング環境上での，日本人による日本語プログラミングの実践とその効果，および，評価実験による日本語プログラムの可読性の検定を述べる．なお，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としての日本語で処理（論理）を記述するのではなく，文字種として母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．この環境上で相当規模のソフトウェアを複数作成してきた．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円滑にした．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．仕様書とプログラムのこの対応のよさは保守性にも寄与している．生産物としての日本語プログラムでは，処理を日本語で言い換えたコメントは減り，モジュールの仕様を記述するコメントが大部分を占めるようになった．当初，保守性への考察から開始した日本語プログラミングは，生産過程での利点から研究室の日本人全員が様々な用途に実践するまでになった．一方で，日本語プログラムの可読性などを評価するための実験を行った．その結果，日本人にとっての日本語プログラムの可読性の高さを検証した．

Practice and Effect of Programming in the Japanese Language

Masaki Nakagawa,[†] Eiichi Hayakawa,[†] Yuji Tamaki[††] and Toshio Souya[†††]

This paper describes practice of programming in the Japanese language by Japanese programmers on an unrestricted Japanese programming environment as well as experiments conducted to test the readability of Japanese programs for 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 environment. 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 maintenance. Japanese programming dispenses with paraphrase of program operations into Japanese comments and encourages to provide module specifications. Experiments were made to test the readability of Japanese programs for Japanese. The results prove it. At first, Japanese programming was motivated to ease maintenance, but now it is practiced by all the Japanese members of our laboratory due to the merits in production stages.

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モジュール間の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，母語が使えれば便利であり，これをすべて英文字で表現しなければならないならば，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

† 東京農工大学工学部電子情報工学科
Department of Computer Science, Faculty of Technology, Tokyo University of Agriculture and Technology
†† (株)東芝 システム・ソフトウェア技術研究所
Systems & Software Engineering Laboratory, Toshiba Co.
††† 日本アイ・ビー・エム(株)東京基礎研究所
Tokyo Research Laboratory, IBM Japan, Ltd.

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．プログラムを書くコストより，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，記述生産性にまさ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．

我々は，プログラミングにおけるツールや方法論以前の問題として，母語の役割を見直す必要があると考えてきた．

一方で，ソフトウェアの国際化のために，自然言語を全く排除する動きもある．形式論理に基づく要求仕様言語[1]や視覚プログラミング[2]がこれにあたる．しかし，現実問題として自然言語を完全に排除できるかどうかは明らかではないし，排除できたとしても，そのデメリットを評価する必要がある．現実には，ソフトウェアの全生産におけるそれらの比率は極めて小さい．

以上の認識が，既存のプログラミング言語において，識別子，メッセージ，コメントなどを含めて，文字種として母語（我々の場合は日本語）が制限なく使用できるプログラミング環境を開発する動機となった．

我々は，既存のプログラミング言語の枠を越えて，曖昧性を排除し，かつ，自然言語に近いプログラミング言語[3),4)]を研究開発することを否定するわけではない．ただ，この種の言語では言語設計や教育の問題が絡み，日本語化の効果を予測，評価することは単純ではない．さらに，このような言語，および，処理系を研究開発するためにも，日本語の使用が自由で，実用的なソフトウェア開発ができるプログラミング環境を必要とした．実際，開発された環境は，自然言語に近い処理や論理の記述を研究するのに何の制限も課さない．

プログラミング言語の日本語化の程度には，いくつかの段階が設定できるが[5]，プログラムの記述で，コメントやメッセージはもちろんのこと，識別子でも日本語の使用を可能にした例としては，一言語の範囲ではいくつかの試みがある．COBOL はかなり初期から識別子や予約語に日本語の導入が試みられたし[5),6)]，Pascal[7]，SNOBOL[8]，CLU[9] などでも報告がある[10]．ところが，これらにおいて日本語化の効果についての定量的な報告はない．

さらに重要な問題は，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ということである．識別子に日本語を使ったために，他言語のプログラムとリンクできなかったり，ツールの恩恵に浴せないのでは，日本語を使うことは逆に障害になる．本論文のプログラミング環境は，システム全体で一貫した2バイト固定長日本語文字コード系を採用しており，オペレーティングシステム（OS）レベルからの統一した日本語化を最初に報告している[11]．

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[12]．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および地域化を遅らせる要因になっている．米国製 OS がマルチバイト化している理由がここにある[13]．

我々は，日本語プログラミング環境の実現過程をすでにいくつかの文献で報告してきた[11),14)~18)]．本論文では，制限のない日本語プログラミング環境上での，日本人による日本語プログラミングの実践とその効果，および，評価実験による日本語プログラムの可読性の検定に重点を置いて述べる．以下，2章で我々の日本語プログラミング環境の特徴を述べた上で，3章で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の可読性に関する評価実験を提示する．

2. 計算機システムの日本語化

これまで，いくつかのプログラミング言語で日本語化がなされてきたにもかかわらず，あまり成功をみなかった理由には，1章で述べたように日本語以外の問題が大きくのしかかっている．日本語化の効果を純粋に議論するには，それ以外のところで重荷を負わせない環境でなければならない．

2.1 商用システムの日本語化

商用システムで一般的に採用されている日本語化の方式は，1バイト体系のコードに，2バイトコードを混在させるものである[12]．この方法だと，システムソフトウェアを全面的に作り直さなくて済む．しかし，プログラミングにおいて，文字ポインタを何文字か進

めたり戻すとき，文字列をスキャンしながら1バイト移動するか2バイト移動するか決めなければならない．ファイルシステムや言語処理系，エディタなどのシステムソフトウェアをはじめ，すべてのプログラムでこの処理に煩わされることになる．ASCII 環境で開発されたソフトウェアを日本語化するのに，多くのもので1年程度を要しているのは，この文字列処理を変更する必要があるからである．

この問題を避けて，ツールやユティリティごとに固定長の内部コードに変換することも行われている．しかし，これではアーキテクチャの統一を欠き，性能，保守性，拡張性を悪くしている．

2.2 内部コードレベルからの日本語化

一言語，一アプリケーション単位の日本語化は，かえって障害になることは先に述べた．一方，システム全体を日本語化するとすれば，前節の二つの方法はいずれも問題を先送りしただけで，今後開発されるすべてのプログラムで非本質的な文字列処理を課すことになる．

我々は，最も単純に，2バイト固定長日本語文字コード系を採用して，それをシステムの統一内部コードとすることによって，OS レベルから一貫して日本語化することにした．このようにすれば，ASCII コード系のシステムで英語が識別子（外部識別子も含めて），コメント，メッセージに使えるように，このコード系では，日本語が何の制限もなく，（外部）識別子，コメント，メッセージに使えるようになる．

内部コードレベルからの日本語化は，一見過大な作業を要するかに思えるが，現実には極めて短期間に完了した．我々は元々，自作のコンパイラおよびOSをASCII 環境で開発していた．それらのソースプログラムにできるだけ修正を加えないで，そのコード系を系統的に日本語化することを行った．方式と作業の詳細は文献 11)，19)，20) に譲る．

2.3 日本語2バイト固定長コード系

日本語2バイト固定長コード系として，JIS X 0208, JIS X 0202, JIS X 0201, に準拠したコード系を設定した．具体的には，1バイトの制御コードにすべて，NULL $(00)_{16}$ を前置し（NULL はどこに挿入しても規格に反しない），制御コードを含めてすべて2バイトとした．JIS X 0208 は，ASCII 文字を部分集合として含み，さらに，日本語文字，ギリシア文字，ロシア文字，記号などを包含し，表現力が十分に高い．またこのことから，このコード系ではメッセージなどを英語から日本語へ，またその逆に翻訳するのに，このコード系の中で閉じて行うことができる．さらに，このコード系の利点は，JIS 規格に合致しているので，ほとんどの入出力機器をコード変換なしに利用できること，そのまま情報交換符号として外部にも出せること，である．

2.4 プログラムの国際化

プログラムを日本語で記述すると国際性がなくなるとの指摘がある[21]．ソフトウェアの国際性は確かに重要である．しかし，この問題はプログラムに限定されるものではない．ソフトウェアには，プログラム以外に，仕様，設計，テスト，保守，マニュアルなどの文書が含まれる．プログラムを英語で書いたとしても国際性が確保できるわけではない．

2バイト固定長コード系での日本語プログラムの方が英訳しやすいことも分かった．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，その変換を行うツールを作成した．このツールを使って日本語プログラムを英語化し，機械的にコード変換した後，ASCII 環境に移植し利用している．英語の対訳表を作るのに英語力を必要とするが，これはプログラミング能力とは完全に分離して行える．つまりプログラムを読まなくても，プログラマでなくてもできる．一方，混在コード系で開発されたプログラムは，確かに識別子は“英文字列”でも，メッセージの英訳だけでなく，文字処理を書き換えなくてはならない．翻訳の専門家を煩わすだけでなく，プログラマが意味の良く分からない識別子を頼りにプログラムを解読し，文字列処理を変更しなければならない．

逆の翻訳，つまり英語から日本語へでも，2バイト固定長コード系と混在コード系では，同様の優劣の差が生じる．

2.5 日本語化の要点

本章では，2バイト固定長コード系を統一した内部コード系としてシステム全体を日本語化することは系統的に行えること，この方式は一般に適用できること，この環境では日本語を使うことに何の制限も複雑な処理の必要もないこと，プログラムの国際化はかえって容易であること，を強調しておきたい．

3. 日本語プログラミングの実践

現在，研究室では，システムプログラムとアプリケーションプログラムの区別なく日本語プログラミン

グを実践している．1988年6月から1993年3月までの約5年間に開発した主なものだけで合計20万行以上に達する．本章では，まず，ソフトウェア開発の一事例から，日本語プログラミングの効果を述べる．次に，学生他に対するプログラミングのアンケート調査から，日本語プログラミングのヒューマンファクタを探る．

3.1 ソフトウェア開発の一例

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．後者は，後々の保守のことを考えて，前者のソースコードを再利用することなく，完全に日本語プログラミングで作り直したものである．サイズが倍になっているのは機能拡張による．

3.1.1 概念仕様から日本語プログラムへ

完全日本語版の研究開発では，1ページ約1600字の用紙で40ページの概念仕様書から，同用紙で約90ページの設計仕様書，同300ページの関数仕様書，および部分的ではあるが，約20ページの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．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円滑にした．また，仕様書とプログラムの対応のよさは保守性にも寄与している．仕様書作成やプログラムの初期コーディングでは，ワードプロセッサも利用した．

3.1.2 コメントの変化

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，ソースプログラムの右側に処理の意味を日本語で説明する（言い換える）ためにコメントが付けられていた．一方，完全日本語版では，90%以上のコメントが，関数やモジュールの先頭で仕様を説明するために使われていた．

この傾向は，それ以後の日本語プログラミングでも確認できる．**図1**はこの完全日本語版以降に作成されたプログラムのコメントに関する統計を示す．仕様説明の総文字数に対する比率は，20%を下らないのに対し処理の言換えは，多くて4%である．

日本語での仕様記述の段階的詳細化が関数などの仕様説明に引き継がれたこと，プログラミング段階での実現仕様の追加が日本語だと苦なく行えること，が従来不十分だった仕様説明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れるコメントが圧倒的に減ったのは，識別子を日本語で命名できることにより，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．

3.2 日本語プログラミングのヒューマンファクタ

我々の研究室では，日本人全員が自発的に日本語でプログラミングするようになった．その理由として次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返すことが伴うので，日本語の方が思考を発展させやすい
（4）保守が容易になる

表1 コメントの用途の変化
Table 1 Change in the role of comments.

用途 / 版	コメント文字数／総文字数	仕様説明／総コメント	処理言換え／総コメント
不完全日本語版	45%	60%	40%
完全日本語版	34%	91%	9%

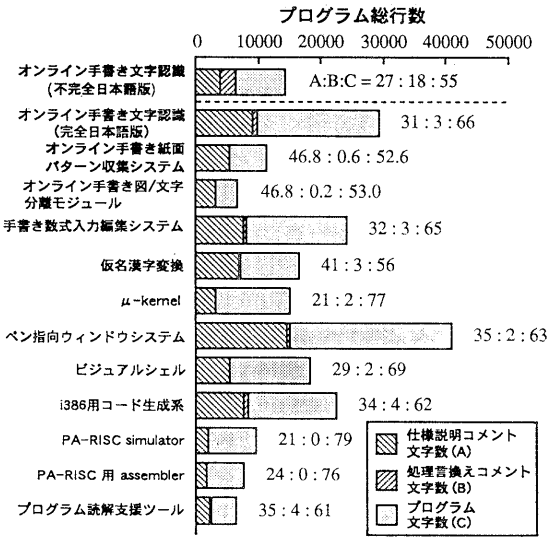


図1 コメントに関する統計
Fig. 1 Statistics on comments.

このことは，日本語プログラミングが日本人プログラマにとってインタフェースがよいことを示している．

一方，研究室に加わったばかりの4年生は，英語でコーディングする傾向が強い．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．

しかし彼らも，一時的に必要なプログラム以外は，後に述べる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，日本語に変換するようになる．それは，上記(4)の理由による．彼ら自身，プログラムリストを引き継いで，日本語プログラムの読みやすさを実感している．そして，次第に(1)〜(3)の理由により，日本語プログラミングに移行している．多くの者が仮名漢字変換辞書に略記法を登録して，少ない手数で入力効率を上げている．

ただし，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．英文をはじめギリシア文字さえ文字集合に含まれるため，プログラマは，記述対象に適した表現を選択すればよい．

4. 日本語プログラムの可読性に関する評価実験

4.1 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．まず，2種類のプログラムを用意した．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．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2**）．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2 プログラムと言語と被験者の組合せ
Table 2 Combination of programs, languages and subject groups.

処理＼組合せ	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

設問に正しく答えた被験者21名について，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2**，**図3**である．以下，図中で標本分散（sample variance）は標本平均（sample

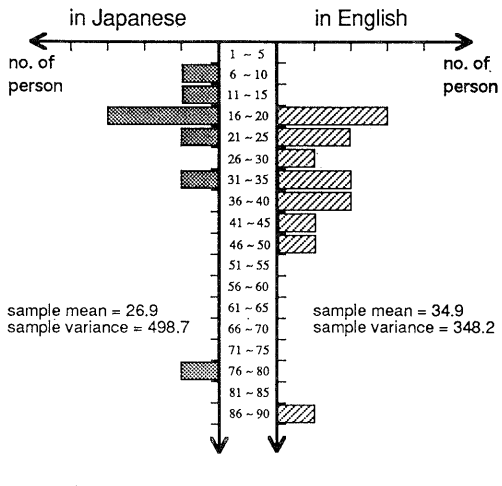


図2 コメント計数の解読時間の分布（予備実験）
Fig. 2 Time required to understand the comment statistic program (preliminary experiment).

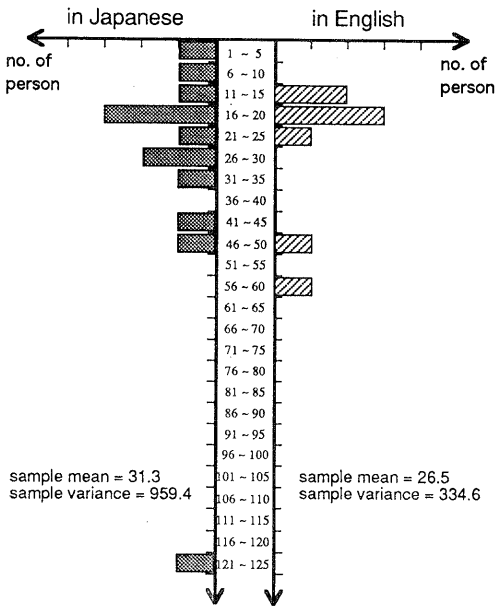


図3 キャッシュの解読時間の分布（予備実験）
Fig. 3 Time required to understand the caching program (preliminary experiment).

mean）との差の2乗和を（被験者数 −1）で除した値である．

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やすいと言える．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やす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，可読性の差が出たのではないか．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．

キャッシュの解読では，別の問題が生じていたことも明らかになった．日本語で解読に2時間を要した被験者は，ここで初めてキャッシングの処理を理解したと述べていた．アルゴリズムを前もって知っている者と知らない者を一緒にはできない．

コメントを除去することについて，非現実的との意見もある．しかし，実際の保守時にはコメントが役に立たないことも多い．また，コメントがプログラムの更新に追従していなくて逆効果になることさえある．信頼すべきはソースコードであり，その可読性の高さが重要である．以下の実験でもコメントは除去する．

4.2 実験の反省

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ある．

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に極端な差が

```
int     main( argc, argv )
int     argc ;
char    **argv ;
{
        char    *テキストファイル先頭 ;

        if( argc < 2 )
                exit(1) ;

        if( ! (テキストファイル先頭 = ファイルを読み込む( argv[1] ) ) )
                exit(1) ;

        数字列と後置語を検索する( テキストファイル先頭, "つ" ) ;
        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, "数", "個" ) ;

        表示する( テキストファイル先頭 ) ;
}

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

図 4 文脈付き数字列検索プログラム（日本語版）の一部
Fig. 4 A portion of the context specified numerical string search program (Japanese version).

つい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．

4.3 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，日本語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，前回同様，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．プログラムの一部を**図4，図5**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により，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．文脈付き数字列検索プログラムの場合の設問は次のとおりである．

（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たとき，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

```
int     main( argc, argv )
int     argc ;
char    **argv ;
{
        char    *head_of_textfile ;

        if( argc < 2 )
                exit(1) ;

        if( ! (head_of_textfile = read_file( argv[1] ) ) )
                        exit(1) ;

        search_for_number_and_postfix_char( head_of_textfile, "つ" ) ;
        search_for_prefix_char_and_number_and_postfix_char( head_of_textfile, "数", "個" ) ;

        print_out( head_of_textfile ) ;
}

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 ;
                }
                else
                        text_ptr++ ;
        }
}
```

図5 文脈付き数字列検索プログラム（英語版）の一部

Fig. 5 A portion of the context specified numerical string search program (English version).

ます。数 1000 個の白い玉もあります。数 10 台の車があります。数 100 台の自転車もあります。』

表 3 に，設問に正しく答えた被験者 20 名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 6**, **図 7** に，以上の統合結果の読解時間分布を示す．

2つのプログラムのそれぞれについて，日本語の方が英語より読みやす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 検定を行った．コメント計数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方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方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，読みやすいと言える．両方のプログラムとも，日本語の方が英語の 2/3 程度の所用時間で読解できている．

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．大規模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，仕様書との対応のよさが有利なはずである．

表 3 プログラムの正しい読解に要した時間
Table 3 Time required to understand the programs.

グループ		被験者	研究分野	課題：数_日 読み時間(分)	課題：数_日 回答時間(分)	課題：数_日 合計(分)	順序	課題：コ_英 読み時間(分)	課題：コ_英 回答時間(分)	課題：コ_英 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	13	31
		6	AP	22	*	22		30	19	49
		7	S	13	5	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	22	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.

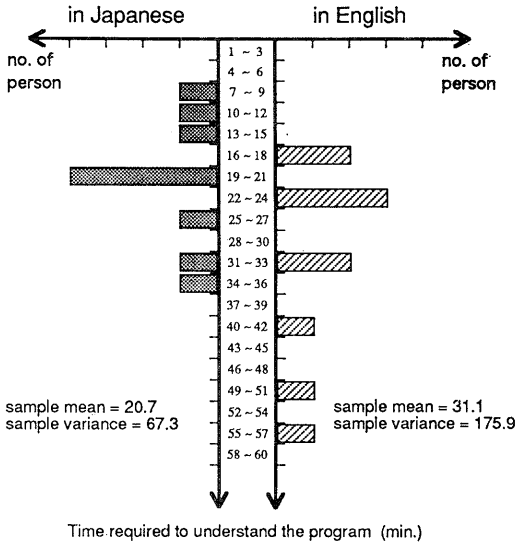


図 6 コメント計数の解読時間の分布
Fig. 6 Time required to understand the comment statistic program.

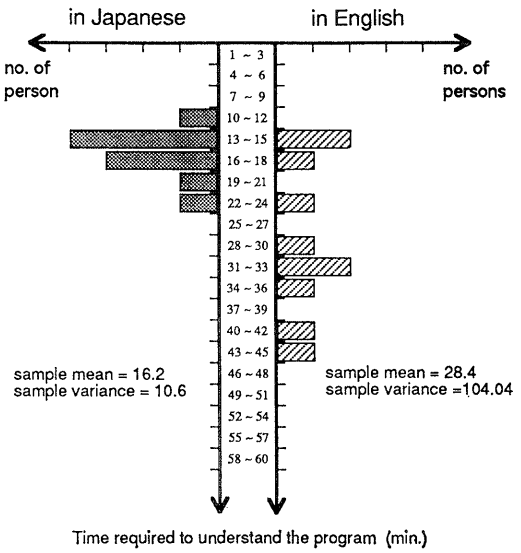


図 7 文脈付き数字列検索の解読時間の分布
Fig. 7 Time required to undertand the context specified numeral string search program.

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．

現実には，1988年6月から1993年3月までの約5年間に，教官および50名の学生が図1に示したものだけでも合計20万行以上の日本語プログラミングを行い，それ以前に作られた日本語化したプログラムを合わせるとソースコードで30万行のプログラムを学生が毎年入れ替わる環境で保守してきた．これには，日本語プログラムの可読性の高さが大きく寄与しているものと考えられる．

5. 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，プログラミング言語の文法の範囲内で，母語を最大限に自由に使用することを意味する．

ここで言う日本語プログラミング環境は，2バイト固定長日本語文字コードを統一内部コードとするOS上に構築されている．これまで，いくつかのプログラミング言語で日本語化がなされてきたにもかかわらず，あまり成功をみなかった理由には，日本語の使用を孤立化したり特殊化したアーキテクチャの問題が大きくのしかかっている．日本語化の効果を純粋に議論するには，それ以外のところで重荷を負わせない環境が不可欠である．

我々は，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．また，仕様書とプログラムの対応のよさ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．約5年間に，合計20万行以上のプログラムを作成し，それ以前に作られ日本語化したプログラムを合わせるとソースコードで30万行のプログラムを学生が毎年入れ替わる環境で保守してこれたのは，日本語プログラミングの効果の1つの実用評価になると考える．

日本語プログラムの可読性の評価では，実験用の小プログラムを用意してその効果を検定した．解読時間は英語に比べ2/3になった．可読性の他にも，デバッグの効率と品質，プログラムの新規作成の生産性と品質などは，日本語と英語で比較実験できるテーマである．今後の課題とする．

日本語プログラミングは，生産過程における円滑な段階的詳細化を可能にするだけでなく，ソフトウェアの可読性，保守性，品質などにおいて，比類のない効果をもたらすものと期待できる．この効果を，実際のソフトウェア工学のなかで評価することも今後の課題である．

謝辞 日本語計算機システムの研究開発を指揮し，かつ，本研究内容についてご討議頂いた，高橋延匡教授，並木美太郎助教授に深謝する．また，被験者，討論者として協力してくれた学生各位，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

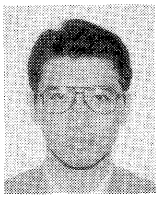
1) 深澤良彰：形式的仕様記述言語とその支援ツール，情報処理，Vol. 30, No. 4, pp. 373-379 (1989).
2) 西川博昭，寺田浩詔：視覚的プログラミング環境，情報処理，Vol. 30, No. 4, pp. 354-362 (1989).
3) 水谷静夫：次第書き言語《小朱唇》の設計思想，情報処理学会プログラミング言語研究会資料，5-3 (1986).
4) 小谷善行：日本語に基づく論理プログラム表現，情報処理学会論文誌，Vol. 34, No. 5, pp. 973-984 (1993).
5) 床分真一，今城哲二：COBOLにおける日本語機能の現状と今後の動向，情報処理学会プログラミング言語研究会資料，16-9 (1988).
6) 西村恕彦：日本語とCOBOL，情報処理，Vol. 8, No. 3, pp. 157-160 (1967).
7) 島崎真昭：日本語Pascal：パスカル，第23回情報処理学会全国大会論文集，1H-1, pp. 215-216 (1981).
8) 吉田和幸，牛島和夫：SNOBOL 4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，Vol. 2, No. 3, pp. 88-98 (1985).
9) 久野 靖，佐藤直樹，鈴木友峰，中村秀男，二瓶勝敏，明石 修，関 啓一：CLUマシンシステムの開発，情報処理学会論文誌，Vol. 29, No. 10, pp. 966-974 (1988).
10) 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，情報処理学会プログラミング言語研究会資料，16-6 (1988).
11) 鈴木茂夫，小林伸行，田中泰夫，中川正樹，高橋延匡：OS/omicronにおける日本語プログラミング環境，情報処理学会「コンピュータシステム」シンポジウム，pp. 11-18 (1987).
12) 長谷川均，岡崎 健：漢字CP/Mのコード体

系，情報処理学会マイクロコンピュータ研究会資料，26-2 (1983).
13) 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, pp. 109-131 (1992. 3. 30).
14) 高橋延匡：研究プロジェクト総説：OS/omicron の開発，情報処理学会オペレーティング・システム研究会資料，39-5 (1988).
15) 並木美太郎，屋代 寛，田中泰夫，篠田佳博，藤森英明，中川正樹，高橋延匡：OS/omicron 用システム記述言語C処理系 cat のソフトウェア工学的見地からの方式設計，電子情報通信学会論文誌，Vol. J 71-D, No. 4, pp. 652-660 (1988).
16) 並木美太郎，関口 治，鈴木茂夫，小林伸行，中川正樹，高橋延匡：OS/o における日本語プログラミング環境と日本語ワードプロセッサの PWB 化，電子情報通信学会論文誌，Vol. J 71-D, No. 6, pp. 994-1003 (1988).
17) 鈴木茂夫，小林伸行，田中泰夫，中川正樹，高橋延匡：OS/omicron における日本語プログラミング環境，情報処理学会論文誌，Vol. 30, No. 1, pp. 2-11 (1989).
18) 岡野裕之，堀 素史，中川正樹，高橋延匡：多重OS「江戸」の設計と実現，情報処理学会論文誌，Vol. 30, No. 8, pp. 1012-1023 (1989).
19) 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 COMPSAC*, pp. 705-712, IEEE Computer Society Press (1991).
20) 中川正樹，玉木裕二，早川栄一，曽谷俊男：母国語プログラミングへの方式，実践とその効果，情報処理学会ソフトウェア工学研究会資料，85-2 (1992).
21) 石田晴久：ページ記述言語の現状と動向，情報処理，Vol. 31, No. 11, pp. 1563-1569 (1990).
22) 中川正樹，早川栄一：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電子情報通信学会春季全国大会，D-72, 6-72 (1991).
23) 中川正樹，早川栄一，玉木裕二：日本語プログラムの可読性の検定，第 45 回情報処理学会全国大会論文集，Vol. 5, pp. 225-226 (1992).
24) 中川正樹，早川栄一，玉木裕二：日本語プログラムの可読性の評価と検討，情報処理学会ヒューマンインターフェース研究会資料，43-1 (1992).

(平成5年6月10日受付)
(平成6年6月20日採録)

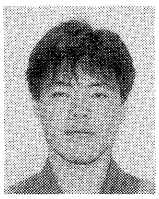
中川 正樹（正会員）

昭和29年生．昭和52年東京大学理学部物理学科卒業．昭和54年同大学院修士課程修了．同在学中，英国 Essex 大学留学 (M. Sc. in Computer Studies)．昭和54年東京農工大学工学部助手．現在，同工学部電子情報工学科助教授．理学博士．



早川 栄一（正会員）

平成元年東京農工大学工学部数理情報工学科卒業．平成3年同大大学院博士前期課程修了．平成6年同大学院博士後期課程中退．平成6年同大工学部電子情報工学科助手．オペレーティングシステム，日本語情報処理の研究に従事．IEEE, ACM 各会員．



玉木 裕二（正会員）

1967 年生．1990 年東京農工大学工学部数理情報工学科卒業．1992年同大学院工学研究科電子情報工学専攻修士課程修了．同年(株)東芝入社．現在，システム・ソフトウェア生産技術研究所所属．ソフトウェア工学に興味を持つ．

曽谷 俊男（正会員）

昭和 39 年生．平成元年東京農工大学大学院修士課程（数理情報工学専攻）修了．平成4年同大学院博士後期課程(電子情報工学専攻)修了．同年，日本アイ・ビー・エム(株)東京基礎研究所入社．工学博士．オンライン手書きユーザインタフェース，オンライン手書き文字認識の研究に従事．電子情報通信学会，IEEE, ACM 各会員．