RA-002

日本語プログラミング言語によるプログラムの可読性の評価
Practice and Effects of Programming with a Japanese Programming Language

馬場 祐人† 筧 捷彦‡
BAMBA Yuto† KAKEHI Katsuhiko‡

1. はじめに

日本語プログラミング言語は，変数や関数を日本語で名付け，また日本語の表記や語順に近い文法でプログラムを書くプログラミング言語である．日本語でプログラムを書くことの利点として，英単語へ置き換えることなく母国語でプログラムを表現できる点がある．プログラムそのものを母国語で書くことができることから，プログラミング初学者はもちろん，大規模なソフトウェア開発においても開発やデバッグ，保守などに寄与することが期待される．

筆者らは，日本語プログラミング言語“ プロデル[1]”を開発中である．プロデルは，オブジェクト指向プログラミングに対応したスクリプト型の日本語プログラミング言語である．

またプロデルは，実用規模のソフトウェア開発を対象とした日本語プログラミング環境を目指している．プログラミング入門や情報教育を対象とした比較的短いプログラムを日本語で書くことを目指しているものではない．既存の日本語プログラミング言語の多くは，自然な日本語の表記および語順に近い文法でプログラム文を書けることを目指しているが，プログラム構文レベルの工夫に止まっている．我々は，構文レベルの工夫では日本語プログラミングの利点は少ないと考えており，ライブラリや開発環境といったプログラミング環境全体で，日本語プログラミングを行うための環境を整備し，その上で日本語プログラミングの利点を議論していく必要があると考えている．そのために我々は，実用的なプログラムを開発することができる日本語プログラミング言語を設計し，開発して，実際に実用規模のソフトウェアを開発する試みを行っている．

本研究は，日本語プログラミング言語を用いてソフトウェア開発を行い，日本語表記で書かれたプログラムの可読性について，その効果を検証することを目的とする．第 2 章では，日本語プログラミングの効果について，既存研究との違いについて述べる．第 3 章では，我々が開発する日本語プログラミング言語“プロデル”の特徴について述べる．第 4 章では，プロデルによるソフトウェア開発について開発環境やプログラムの留意点と対処について述べる．第 5 章では Java で書かれたプログラムとソースコード上での比較を行い，第 6 章では可読性についての実験内容とその結果を報告し，考察を述べる．第 7 章でまとめを述べる．

2. 日本語プログラミングの効果

プログラムに母国語である日本語を用いることの利点として，ソフトウェアとして実現したい対象を日本語表記のままでプログラムとして実装することができ，また日本語表記から英単語表記への不要な置き換えによる手間や混乱を避けることができることにある．そのことによりデバッグや保守においても，寄与することが期待される．

2.1. 関連研究

中川らは，日本語プログラミング環境上で日本人による日本語プログラミングを実践し，その効果を実験により検証している[3]．中川らは論文で，仕様書と日本語で書かれたプログラム(以下日本語プログラム)との対応の良さは，保守性にも寄与していると指摘しており，我々が求める日本語プログラミングの方向性と合致する．ただし，中川らの研究で意味する日本語プログラミングとは，識別子，メッセージ，コメントなどを含めてプログラムすべてで日本語を制限なく扱えることである．当時の計算機環境は，OS レベルで多バイト化されつつあったが，プログラミング言語で日本語が利用できる環境が必ずしも整備されている段階にはなかった．一方，現在では，計算機上で日本語を取り扱うことはごく当たり前になっている．Java や Ruby など多くのプログラミング言語では，プログラムに日本語を制限なく用いることができる．そのため，中川らの研究で行われた意味での日本語プログラミング環境は，すでに整備され，日本語プログラミングは容易に実現できる．

ソースコード上でごく普通に日本語が扱えるようになったことで日本語プログラミングの意味が変化していると我々は考えている．現在の日本人プログラマの多くは，日本語で表現した対象物を，難なく実装時に英単語表記へ置き換えてプログラムへ実装している．Java などで識別子といった単語レベルの日本語を書くことは，プログラミング初心者にとっては利点があるが，経験のあるプログラマにとってはあまり大きな利点とならない．我々が示す日本語プログラミングとは，識別子に日本語単語を用いるだけではなく，構文を含めて日本語らしい語順で書くことであると考えている．

一方，現在は，計算機の用途が増え，機能が豊富になり，それと共にライブラリも膨大になっている．ソフトウェア開発においても多くはこれらのライブラリに依存している．そのため Java などで識別子に日本語を用いてプログラムを開発した場合でも，これらのライブラリに依存する部分は，英単語表記のままであり，日本語表記と英単語表記が混在化したプログラムとなり，かえって可読性が低くなる．日本語プログラミングを実践するためには，日本語で名付けられたライブラリを整備することが不可欠であり，ライブラリを含めた開発環境全体で日本語プログラミングを実践する必要があると我々は考えている．我々は，日本語プログラミング言語“プロデル”の開発を通じて，構文のみならず日本語を用いたライブラリも同時に設計し，それらを含めた開発環境全体で日本語プログラミング環境を検討している．

† 早稲田大学 理工学術院 基幹理工学研究科
Graduate School of Fundamental Science and Engineering, Waseda University.
‡ 早稲田大学 理工学術院
Faculty of Science and Engineering, Waseda University.

さらに，中川らの研究の後に数千行規模以上のソースコードを日本語で書くことの定量的な測定を行った研究はなく，現状を踏まえた日本語プログラミングの効果を測定することは有意義であると，我々は考えている．

本研究では，現在の計算機環境を踏まえた上で，今現在一般に行われるソフトウェア開発の方法に則って，日本語プログラミングを実践し，その上で効果を評価することが目的である．

2.2. 仕様記述言語としての日本語プログラミング

仕様記述言語を使い日本語で形式的な仕様記述を行う研究がある．大森らは，自然言語で書かれた日本語の仕様書から形式モデルへの変換を支援するツールを提案している[5]．また畠山らは，オブジェクト指向一貫記述言語 OOJ[6]を提案し，理工系分野の専門家向けに日本語で一貫したプログラミング環境を提供している．日本語で形式仕様を記述することで，段階的に仕様を詳細化し，最終的に C++や Java で書かれたプログラムを自動生成する．

一方，我々はプログラミング言語の立場から日本語によるソフトウェア開発環境を模索している．形式仕様記述では，仕様をより詳細化するにつれて，最終的な実装言語のライブラリに依存する部分(クラス名など)を英単語表記で書く必要がある．日本語プログラミング言語ではライブラリも含めて日本語表記であるために一貫して日本語を用いることができる．

本研究は，既存研究を否定するものではない．むしろ日本語プログラミング言語を導入することにより，プログラミング言語の立場から既存研究をより発展させることができると我々は考えている．

2.3. 日本語プログラミング言語

日本語プログラミング言語は，変数や関数を日本語で表記し，また日本語の語順に近い文法でプログラムを書くプログラミング言語である．ごく自然な日本語らしい文法でプログラムを書けることで，仕様書で用いた表現をそのままプログラムに用いることができる．

日本語プログラミング言語は，以前から存在する．実用的なソフトウェア開発を目的とした日本語プログラミング言語には，“まほろば[7]”，“Mind[8]”が挙げられる．Mind は，市販ソフトウェアの開発に利用された事例が報告されている．また，フリーソフトでは，“なでしこ[12]”や“TTSneo[13]”が挙げられる．なでしこは，豊富なライブラリを持ち，事務処理などの実用的なソフトウェア開発に利用されている．しかし，これらはオブジェクト指向プログラミングには対応しておらず，モデル駆動開発といったソフトウェア開発手法による実装が行いづらい．

さらに日本語表記の言語仕様を採用しているプログラミング言語として“ことだま on Squeak[9]”，“ドリトル[10]”および“PEN[11]”が挙げられる．ドリトルはプロトタイプ型のオブジェクト指向プログラミング言語である．これらは情報教育で利用することを目的としている．また，実用的なソフトウェア開発を行った事例は，見あたらない．

我々は，オブジェクト指向プログラミングができ，実用的なソフトウェア開発を目的とした日本語プログラミング言語を開発し，その上で，日本語プログラミング言語の特長を生かした利点について議論したいと考えている．

2.4. 日本語プログラミング言語の利点

Java や C#など最近のプログラミング言語の処理系は文字コードに Unicode を使うことができることから，クラスや変数を日本語で名付けることができる．例えば，図 1 に示すプログラムは，Java のプログラムであり，図 2 は，図 1 の単語を日本語に置き換えた Java プログラムである．クラスやメソッド，変数だけを日本語で名付けても，プログラムの可読性に欠ける．予約語(public, void)やクラス(String)には英単語が使われており，英単語表記と日本語表記が混在したプログラムとなっているからである．

また，図 2 に示したような単語レベルの置き換えでは，結局は，何が動作であり何が動作対象であるかを記号や書き順だけで判別するしかない．日本語プログラミングの利点は，動作の対象や付随する情報をごく自然な日本語らしい文法で書けるようにすることで，プログラムを読み書きしやすくしていることである．この点は，後に詳しく説明する．この利点を生かすためには，構文のみならずライブラリなどの開発環境全体で日本語を使って書けることの利点を議論する必要があると考えている．

```
public void greet(String name) {
    System.out.print(name + "さん、こんにちは");
}
```

図 1　Java プログラム

```
public void 挨拶する(文字列 名前) {
    システム.出力.表示する(名前 + "さん、こんにちは");
}
```

図 2　図 1 の単語を日本語に置き換えた Java プログラム

2.5. 本研究の新規性

本研究の新規性は，次の点にある．

日本語プログラミング言語の導入

本章で挙げた関連研究では，設計時は日本語を用いるが，実装では C 言語や Java などにある英単語表記のライブラリを用いてプログラムを書くことになる．本研究では，ソフトウェア開発の実装段階でも日本語プログラミング言語を用いて実装することで，日本語プログラミングの利点を模索する．

オブジェクト指向によるソフトウェア開発

中川らの研究では，構造化プログラミングによって作成された C 言語のプログラムを用いて実験が行われていた。一方，本研究は，オブジェクト指向開発に沿って現在主流の手法でソフトウェア開発を行い，その効果を検証する．

3. 日本語プログラミング言語“プロデル”

本研究では，日本語プログラミング言語“プロデル”を用いてソフトウェア開発を行う．プロデルは，既存の日本語プログラミング言語の特徴を踏まえた上で，これに加えて，オブジェクト指向プログラミングを行うための基本的な言語仕様と，日本語で名付けられたライブラリを提供している．この点で，既存の日本語プログラミング言語と異なる．

我々は，日本語プログラミング言語“プロデル”を設計し開発すると同時に，プロデルを使って日本語プログラミングによるソフトウェア開発を行うことで日本語プログラミング言語の利点や問題点を模索している[2].

なお，プロデルは，我々の研究室での利用にとどまらず，Web で公開し一般利用者にも日本語プログラミングを体験し，実践していただいている．

3.1. 自然な日本語で書くプログラム

変数名や関数名などを日本語で書けるだけでなく，助詞を使って日本語の語順でプログラムを書くことができることが日本語プログラミング言語の特徴である．プロデルでも，ごく自然な日本語の表記や語順でプログラムを書くことで，プログラムが読みやすくなるような言語仕様としている．

助詞を使った日本語プログラムについて例を挙げて説明する．図 3 は，C 言語風に書いたメソッド呼出しの例であり，図 4 は，図 3 に相当する例をプロデルで書いたプログラムである．

```
copy("文章.txt", backupPath);
```

図 3　C 言語風のプログラム

```
「文章.txt」をバックアップ先へコピーする。
```

図 4　プロデルのプログラム

図 3 で示したように，一般的にメソッド呼出しは，メソッド名および実引数(必要な場合) を書く．図 4 で示したようにプロデルのメソッド呼出しも同様にメソッド名と実引数を書く．C 言語風のプログラムと異なる点は，実引数と仮引数を結びつけるために“助詞”を書く点である．実引数の直後に添えられた助詞によって，実引数と仮引数が対応付けられる．この対応付けによって，メソッド呼出しで，引数をメソッド宣言で定義された順番に書かなくても，メソッドへ正しく引数を渡すことができる．

助詞を使った文法の利便性は，名前付き引数(キーワード引数とも呼ばれる)の利便性と一致する．例えば Ada ではキーワード引数を使うことで，仮引数の名前と実引数とを対応付ける機能がある．この機能により，引数の順番を意識することなくプログラムを書くことができ，またプログラムが読みやすくなり保守にも貢献している[14]．また Scala や C# 4.0 以降などにも名前付き引数の機能が実装されている[15].

メソッド呼出しで引数を助詞と対応付けて書くことで，操作対象や入力値などの動作の対象や関係を明確にすることができる．またプログラムを書く際にその関数の仮引数の順番を覚えておく必要がなくなり，プログラムを読む際も，直感的に引数の関係が理解できる．

このようにプロデルは，日本語らしくメソッド呼出しを書けるようにすることで，プログラムを読み書きしやすくしている．

3.2. プロデルによる Web アプリケーション開発

プロデルで書かれたプログラムを Web サーバ上で動作させるための動作環境として，プロデル簡易 Web サーバおよび CGI 環境で動作する実行環境の 2 つを用意している．前者は，プロデル専用の単独の Web サーバである．後者は，Apache と CGI で連携して動作させることができる．

Java に代表されるプログラミング環境で Web アプリケーションを開発する場合，Servlet の上で Struts や Spring といったフレームワークを用いて開発することが一般的である．このようなフレームワークを用いる利点の一つとして MVC(Model，View，Control)の各役割を分離して書くことができる点がある．

一方プロデルには，フレームワークによって MVC によって役割を分離して書くための仕組みは用意されていなかった．本研究を通じて，ソフトウェア開発を行うと同時に，フレームワークの最低限の機能を実装した．

4. 日本語プログラミング言語によるソフトウェア開発

4.1. ソフトウェア開発の例

我々は，プロデルを開発すると同時に，いくつかのソフトウェアをプロデルで作成している．その総行数は，大中規模なソフトウェアだけで見ると 13,000 行程になる．図 5 は，我々がプロデルで作成した主なプログラムの実行コード行数とコメント行数をまとめたものである．

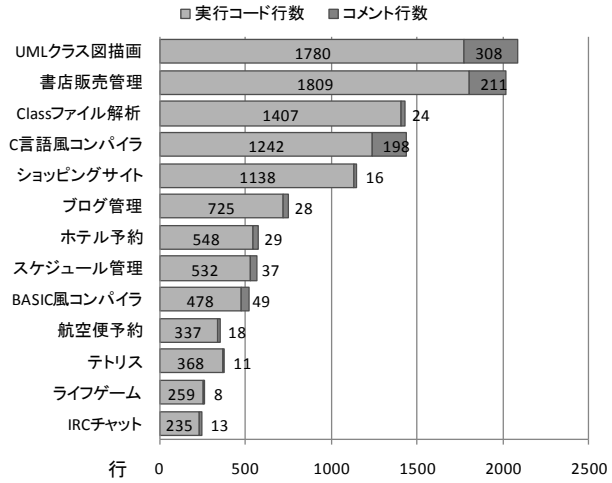

図 5 プロデルで作成したソフトウェアの行数

図 5 で示したようにソースコード総行数に占めるコメント行数の割合が 1〜2 割と一般に Java で書くプログラムよりも少ない．この点について第 5 章で理由を述べる．

4.2. 日本語プログラミングの留意点とその対処

英単語と日本語では，表記による書き分け方が異なる．英単語を組み合わせてプログラム中の名前とする場合は，大文字・小文字の使い分けでその表す対象を区別することが行われている．例えば，Java の仕様書では，クラス名に Pascal 表記を使い，インスタンス名やメソッド名に Camel 表記を使うように推奨している[16]．Java API 群では，実際にこの推奨規則に則った命名が行われている．日本語の名前を使う場合は，識別対象を書き分けるのに，大文字・小文字の書き分けが使えないから，これとは別の工夫をするほかはない．本節では，日本語の名前を使う場合に，命名対象をうまく区別するために採った書き分け方の工夫を紹介する．

クラス名とインスタンス名

自然言語では，クラスとインスタンスをともに同じ名前で呼ぶことが多い．一方，プログラムの上ではこれらを区別したいので，実質的に同じ名前となることがある．その

ために，それらを書き分ける工夫が必要となる．日本語で書き分けるのに，すぐに思いつくのは，同じ名前を漢字で書くか，かなで書くかで区別する方法である．ところが，かな綴りの名前は，日本語プログラミング言語でのプログラムの中では，構文を表すための部分と区別がつきにくく読み取りにくい．名前としては，漢字列やカタカナ列を使うのが視認しやすい．

そこで，名前には漢字列やカタカナ列を原則として当てることにした，その上で，クラス名には簡潔なものを選び，インスタンス名には個を示すものをクラス名に前置してできる名前を用いることにした．

実装時に導入されるものの名前

クラスやメソッド，フィールドの名前は，設計段階で決まっている．これに対して，カウンタ，イテレータ，バッファなどは，詳細設計や実装段階になって，はじめて決める．実装段階では，さらにメソッドの局所変数や各種作業の一時変数も導入される．与えられた全候補それぞれについて作業を行う反復処理に使われるイテレータ変数については，その候補となるクラス名の前に“対象”を添えた名前を付けることにした．

5. ソースコードによる比較評価

プロデルで作成したソフトウェアについて，コメントの観点から評価する．表 1 は，プロデルで書かれたプログラム(以下プロデルプログラム)として，ブログ管理システム(以下“ブログ管理”)，ホテル予約システム(以下“ホテル予約”)および書店店員向け販売管理システム(以下“書店販売管理”)について，コード行数とコメント行数をまとめたものである．また参考のためにソースコード中の実行コードとコメントの割合を示した．

表 1 実行コードとコメントの行数とその割合

	プロデル				Java			
	実行コード		コメント		実行コード		コメント	
ブログ管理	725	96.3%	28	3.7%	-	-	-	-
書籍販売管理	1809	89.6%	211	10.4%	2699	66.1%	1383	33.9%
ホテル予約	548	95.0%	29	5.0%	608	49.6%	619	50.4%

表 1 に示したとおり，Java プログラムでは実行コード行数とコメント行数の割合が同程度かそれに近い割合であったのに対して，プロデルプログラムでは，実行コード行がほとんどで，コメント行数が 1 割以下になった．

コメントが少なくなった理由は．Java プログラムにおいてメソッドやフィールドの説明となっていた表記がそのままプロデルのメソッドやフィールドの名前となるからである(図 6 左)．そのことで，Java プログラムでは，必要だったコメントのほとんどがプロデルでは不要になった．

メソッドの仮引数でも同様のことが言え，Java では仮引数に対して日本語の説明を書くが，それらに書いた日本語の説明が，プロデルではそのまま仮引数名となるため，わざわざ仮引数の説明を書く必要がなくなった(図 6 右)．このように Java のコメントでメソッドの説明や仮引数の説明として書いた日本語の説明文が，プロデルではそのまま識別子として書く類となる．プロデルでは実行コードそのものを見れば自明となることが多かった．そのことで表 1 で示したように，プロデルプログラム中のコメント行数の割合が，Java プログラムに比べて大幅に少なくなった．

日本語プログラミング言語によるプログラミングによって実行コードそのものが挙動や役割を表すようになったことで，コメントの質が変化した．これまでのコメントは，主に実行コードの一行一行の処理の言い換えによる説明であったが，日本語プログラミングでのコメントは，より仕様としての説明(メソッドを使用するときの前提条件やメソッドの戻り値がどのような時にどのような値を返すか)が主となる．このことで，より詳細な説明をコメントとして書き残すことに労力を回すことができると期待できる．

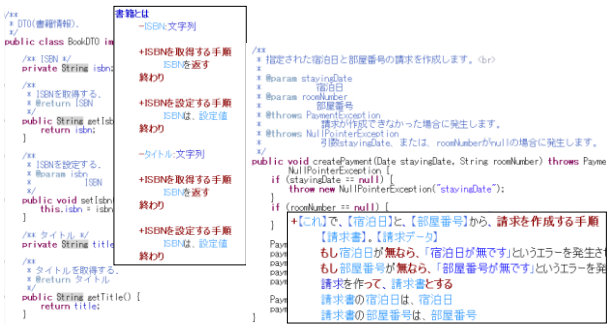

図 6 Java と比較して減少したプロデルのコメント

6. 可読性の評価実験

我々は，日本語プログラミングが実装のみならずデバッグや保守にも利点があると考えている．本研究では，日本語プログラムの可読性が，デバッグや保守にどのような利点をもたらすのかを評価するために，3 つの実験を行った．本章では，それぞれの実験方法について述べる．

6.1. 実験対象と被験者

本実験では，ブログ管理，ホテル予約および書店販売管理の 3 つの例[17]を通じて被験者に実験させた．これらのソフトウェアを対象とした理由は，Web アプリケーションであり，データベース操作を用いるプログラムであるなど，現在，現実的によく開発されるプログラムで検証したいと考えたからである．中川らが行った実験[4]では，コメント計測や数字列検索といった比較的短いプログラムを実験対象にした．しかし，その報告でも指摘しているように，アルゴリズム例題的なプログラムは，アルゴリズムが理解できるかどうかに依存する．我々は，現在一般的な形態に近いソフトウェアで，被験者がプログラムを理解できるかどうかを検証する．

被験者は，我々の研究室の学生 11 名である．彼らは，日本語を話す情報理工学を専攻する大学院生であり，講義や研究を通じてプログラミング経験がある．中にはアルバイトや趣味を通じて多くのプログラムを書いた経験がある学生もいる．

すべての学生が Java や C 言語によるプログラム経験があるが，このうち日本語プログラミングの研究を通じてプロデルによるプログラム経験がある学生は 3 名であり，その他の学生はプロデルによるプログラム経験はない．

Java プログラムのコメントについては，直接的な解答となるソースコード冒頭のタイトルや著作などの情報を削除したほかは，一般に書かれる程度の日本語によるコメントは残し，ありのままで実験してもらった．また，被験者には実行環境を渡さず，ソースコードから得られる内容だけ

で読解してもらった．ただし Java プログラムのデバッグについては，ソースコードの量が膨大で実験時間が長くなることから範囲を半分程度まで絞り込んだ上で実験した．

ソースコードの読解やデバッグには，Java プログラムは Eclipse を使って，プロデルプログラムは専用のエディタを使うことを推奨したが，これらに限らず被験者が作業しやすい環境で作業してもらうことにした．

各実験を通じて，被験者の間で，質問の解答について教えなければ，プロデルの構文や Java のフレームワークの構成は，お互いに教えあっても構わないとした．

6.2. 日本語で書かれたプログラムの理解の実験

プロデルで実装したブログ管理のソースコードを，被験者に何をするプログラムであるかを説明せずに，読んでもらい，その上で，ソースコードについていくつか質問した．質問を通じてソースコードの理解度や理解するのに要した時間を測る実験を行った(以下実験 A)．

被験者には，「このプログラムが何をするプログラムですか」，「このプログラムにはどのような機能があるかを 6 つ挙げてください」といった質問に解答してもらい，その都度，解答がわかった時刻を記録してもらった．

6.3. デバッグ効率の評価実験

書店販売管理とホテル予約のソースコードに意図的に複数のバグを混入し，被験者にプログラムの機能を理解するのに要した時間と，デバッグ作業を行わせたときのバグを発見するのに要した時間を計測した(実験 B,C)．事前に 2 つのシステムを，Java で実装したものとプロデルで実装したものとそれぞれ用意した．また被験者を 2 つのグループ(第 1 グループおよび第 2 グループ)に分けて，グループごとにシステムや実装言語が異なるソースコードを渡し作業してもらった．表 2 には，被験者のグループと作業対象のシステムとの対応を示す．なお，第 1 グループと第 2 グループとでプログラミング経験にできるだけ偏りができないように配慮した．

被験者には実験 A と同様に，渡したソースコードが何をするプログラムであるかを明らかにしない上で，ソースコードを読んでもらった．その上で，「何をするプログラムであるか」また「どのような機能を持つか 6 つ挙げ」させ，ソースコードを理解するのに要する時間を計測した．さらにデバッグではソースコード中からバグを 3 つまで探し，その修正方法を解答させた．

まず実験 B では，各グループに Java プログラムを読ませて，質問に答えてもらい，さらにデバッグさせた．次に実験 C では，プロデルで書かれたプログラムを同様に読ませて，デバッグさせた．

なお，被験者がバグを発見したときには，そのバグが正解かどうかを筆者らに確認しながら進めた．

表 2 実験グループと対象システムの対応

	実験A	実験B	実験C
第1グループ	ブログ管理 (プロデル)	書籍販売管理 (Java)	ホテル予約 (プロデル)
第2グループ		ホテル予約 (Java)	書籍販売管理 (プロデル)

6.4. 混入させたバグ

実験 B,C で作業対象にする，書店販売管理のソースコードとホテル予約のソースコードには，それぞれ 5 種類のバグを混入させた．混入したバグは，業務系アプリケーションで起こり得るものを選び，処理の流れが理解できないと見つけられないものにした．なお混入したバグは，同じシステムについて Java 版およびプロデル版とで，箇所や種類は，等しくした．混入させたバグの 5 つのバグの種類を次に説明する．

呼び出すメソッドの誤り

引数の数やその型が等しい別メソッドを呼び出すバグを混入した．書籍販売管理では，書籍(Book)オブジェクトに格納された値をデータベースへ反映する“変更する(update)”メソッドを“挿入する(insert)”メソッドに書き換えた(図 7)．また，ホテル予約では，請求(payment)オブジェクトの状態を取得する“状態を取得する(getStatus)”メソッドを“部屋番号を取得する(getRoomNumber)”メソッドに書き換えた．

メソッド呼び出し順の誤り

メソッド呼び出しを，前後 5 行の範囲内で本来と異なる別の行と入れ替えるバグを混入した．書店販売管理では，出版社の入力フォームで入力した値をオブジェクトのフィールドへ代入する時に，“電話番号(tel)”と“住所(address)”を入れ替えた．また，ホテル予約では“部屋をチェックインする(checkInRoom)”と“部屋をチェックアウトする(checkOutRoom)”の操作を入れ替え，メニュー画面の表示と実際の動作とが合致しないバグを混入した(図 8)．

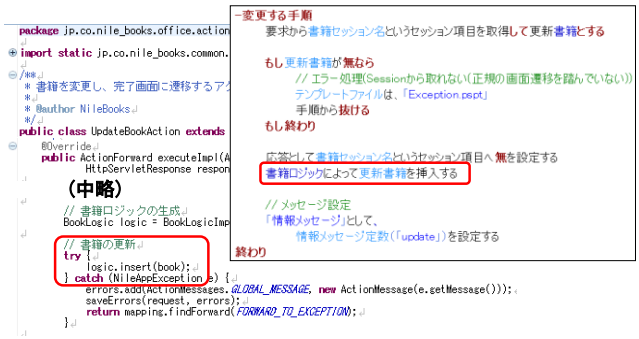

図 7 呼び出すメソッドの誤り

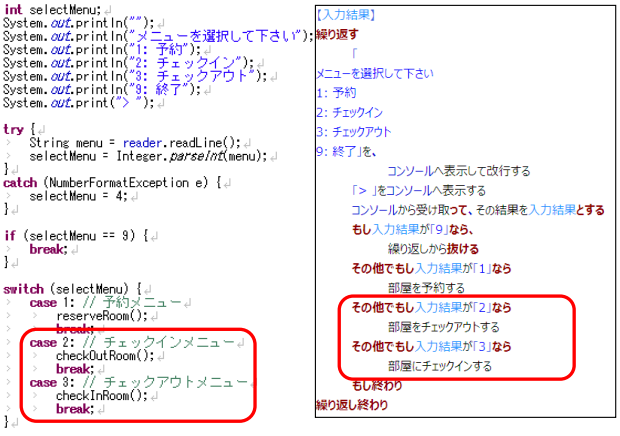

図 8 メソッド呼び出し順の誤り

条件式の誤り

条件分岐の箇所で，正常な条件とは逆の時に真となるバグを混入した．書籍販売管理では，データベースの更新で正常に処理された件数が 0 件の時，例外を発生させる箇所を，0 件以外の時に発生させるようにした．またホテル予約では，変数が null 値の時に新たにインスタンスを生成する箇所を，null 値でないときに生成させるようにした．

定数の誤り

本来設定すべき定数値とは，異なる値の定数値を設定するバグを混入した．書籍販売管理では，書籍の検索処理で，検索条件となる書籍の状態を“正常(NORMAL)”から“欠品(OUT_OF_PRINT)”へ書き換えた．また，ホテル予約では予約登録の時に予約の状態を“予約登録(CREATE)”とすべき箇所を“消費済み(CONSUME)”にした．

処理の欠落

本来行うべき処理を 1 行だけ欠落させるバグを混入した．書籍販売管理では，指定した書籍を検索する時に，検索条件として書籍 ID を設定する処理を欠落させた．またホテル予約では，予約後に空室数を更新する処理を欠落させた．

6.5. 評価

実験 A,B,C を通じて計測した時間について評価する．表 3 は，実験 A,B,C で，プログラムの理解に要した各グループの平均時間を示したものである．なお時間について書体が同じものは，同じシステムであることを表している．

表 3 の実験 A を見ると，第 1 グループと第 2 グループとでプログラム理解にかかる時間に大きな違いはなかった．また実験 B,C を比較すると，太字(書店販売管理)については，プログラムが大規模であったために大きな違いはなかったが，下線(ホテル予約)については，15 分以上早く理解することができた．このことから Java よりもプロデルの方が比較的早くプログラムを理解できる傾向が見える．

図 9 には，実験 B,C を通じて，「このプログラムが何をするためのプログラムであるか」を質問したときの解答にかかった時間とその人数の関係を示した．図 9 左が書店販売管理(Java 版とプロデル版)の解答に要した時間で，図 9 右がホテル予約(Java 版とプロデル版)の解答に要した時間である．図 9 で示した実験 B,C の結果を見ると，プロデルプログラムの方が短時間で理解できる傾向が見える．

表 4 は，実験 B,C において，各被験者がバグを見つけるのに要した時間を示したものである．なお，参考に被験者の主なプログラミング経験場所と日本語プログラミング言語の経験の有無も示す．表 4 で示したように，第 1 グループの実験 B と第 2 グループの実験 C は，共に書店販売管理であり，デバッグが難しかったようで全くバグを見つけることができなかった被験者もいた．一方，第 2 グループの実験 B と第 1 グループの実験 C は，共にホテル予約であり，経験を問わず比較的バグが見つけることができた．

また図 10 は，図 9 の結果から被験者がバグを 1 個発見するのに要した平均時間を，システムと実装言語で分けて示したものである．図 10 に示したように，プロデルでは Java と比較して，書店販売管理では 7 割程度に，ホテル予約では 3 割程度に，バグ発見平均時間が短くなった．

6.6. 考察

本実験の結果から，Java で書かれたプログラムよりもプロデルプログラムの方が，早くプログラムを理解することができ，またソースコードだけを見て，バグを発見するのに要した時間もプロデルプログラムの方が短くなる傾向にあることを確認できた．

表 3 プログラムの理解に要した平均時間

	実験A (プロデル)	実験B (Java)	実験C (プロデル)
第1グループ	0:18	**0:13**	0:06
第2グループ	0:15	0:23	**0:09**

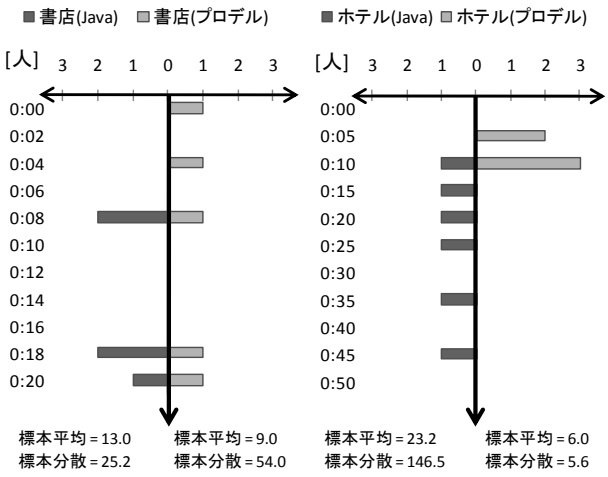

図 9 書店販売管理とホテル予約の理解に要した時間

表 4 被験者ごとのバグ発見に要した時間

		実験B			実験C		
グループ経験	JPL経験	1つ目	2つ目	3つ目	1つ目	2つ目	3つ目
1 大学	なし	0:40			0:02	0:57	0:08
1 大学	なし				0:00	0:29	
1 大学	なし	1:28			0:21	0:13	
1 趣味	あり	1:15	0:19	0:12	0:01	0:12	0:11
1 バイト	なし	0:17	0:16	0:42	0:12	0:05	0:03
2 バイト	なし	0:36	0:38	0:29	0:07		
2 趣味	あり	0:13	0:10	0:56	0:44		
2 大学	あり	0:20	0:21				
2 趣味	なし	0:16	0:26	0:32	0:48	0:17	0:10
2 趣味	なし	1:26	0:13		0:09	0:33	0:03
2 大学	なし	1:09					

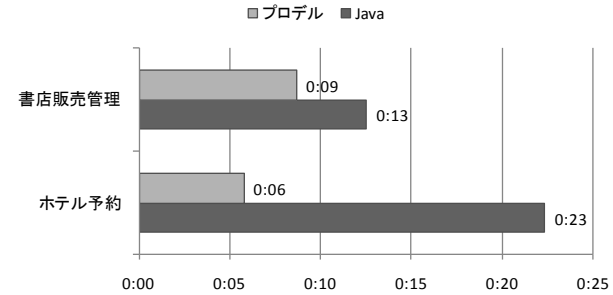

図 10 バグ 1 個の発見に要した平均時間

当然，この実験だけで日本語プログラムが Java のプログラムよりも可読性がよいと言い切ることはできない．しかし，この結果から日本語プログラムは，少なからずプログラム理解やデバッグ効率で，良い影響があると言える．日本語プログラムの可読性について，この実験から考えられる肯定要因と否定要因を次に述べる．

実験結果の肯定要因

Java プログラムのコメントは，質問の直接の答えとなるシステムの名称が書かれたファイル冒頭を除いては，現実の状況と同じように日本語で書かれたコメントもありのままソースコードに残した．一方，プロデルプログラムには，ほとんどコメントは含まれず，被験者は，ソースコードの実行コードのみから理解したりデバッグしたりしたことになる．これは，単にソースコード中の日本語の説明の有無が理解に影響したものではなく，またプロデルで書かれた実行コードから純粋に，プログラムを理解したことを表している．つまり，保守においてプロデルプログラムの場合，他人が作成したコメントが全くないプログラムであっても，プログラムを十分に理解できる可能性がある．

また，実験は両グループとも Java，プロデルの順にデバッグを行った．Java プログラムのデバッグ作業の時点で，混入されたバグの傾向がわかり，バグを探しやすくなった可能性がある．現に，Java プログラムで 3 つのバグを見つけられた被験者は，混入されているバグを経験から予想した上で探していた．しかし，表 4 の第 2 グループを見るように Java プログラムのバグを 3 つ発見した被験者でも，プロデルプログラムのバグが 3 つ発見できない被験者もおり，必ずしもその傾向があることは言えない．

さらに，被験者のほとんどがプロデルで書かれたプログラムを初めて読んだと言っていた．実験時にプロデルの文法説明を A4 両面刷り 1 枚で配布したが，一般に考えれば，すでに慣れている Java プログラムに比べて，初見のプログラミング言語のプログラムを読む場合，プログラム構文を理解するのに手間取るはずである．しかし，実験 C の時間を見ると，そのようなことはなかった．日本語プログラミング言語であることで構文の理解も容易にしたことが考えられる．

余談ではあるが今回の実験を通じて，Java とプロデルで書かれたソースコードに，あらかじめ混入したバグとは異なるバグがそれぞれ 1 ヶ所見つかった．プロデルでのバグは，混入した他のバグと同種のバグであった．Java プログラムで見つかったバグは，正しい実行コードに対してその直前にあるコメントが誤っているものであった．これは，同じようなプログラムからコピー&ペーストした時に，実行コードだけ書き換え，コメントを書き換えなかったことによって生じたものと思われる．プロデルで同様の実行コードを書いた場合，実行コードとコメントが同じ表現になるため，そもそもコメントを書く必要がなく，このような誤りが生じる可能性は低い．

実験結果の否定要因

Java は厳密な型宣言が必要なプログラミング言語であるのに対して，プロデルはスクリプト型のプログラミング言語である．この特徴により，Java よりも簡潔にプログラムを書くことができる．この点が理解を簡単にした可能性がある．

また，クラスの抽象化などの実装方針やフレームワークの構造を理解するために時間を要した可能性がある．Java 版の書店販売管理では Struts 1.3 を用いて開発しており，フレームワークの利用方法に沿って MVC に分かれてクラスが定義されている．また同様に環境に依存する実装ロジックを抽象化するために多数のクラスが定義されていた．なお，XML ファイルなどの設定ファイルは，今回の実験の対象外としており，実験対象は Java ファイルのみである．そのため Struts 1.3 の構成を理解していなくても十分に解答可能である．一方プロデルプログラムでも，抽象クラスを定義するなどできるだけ似た設計で実装した．そのためフレームワークを用いた開発の未経験者であれば，Java とプロデルとでソースコードを理解することに同程度の難しさがあったはずである．ただ，プロデルのフレームワークはシンプルな構成であり，Java の成熟された技術とは比較できない上，平等な環境を用意することは難しい．

さらに，本実験は，15 分の休憩を挟み 4 時間かけて実験 A,B,C の順で連続して行った．実験 B の Java プログラムのデバッグ時点で被験者の多くがかなりの疲労感を感じていることが見て取れた．その直後にプロデルのプログラムを読みデバッグしたことを踏まえると，疲労度を考慮した厳密な実験では，さらにプロデルでのデバック時間が短縮することも考えられる．

6.7. 被験者の意見

本実験の最後に，被験者に自由に意見を書いてもらった．その中には，「プロデルでプログラムを書くことは面倒であるが，読みやすい．一方 Java は書きやすいが他人が書いたプログラムは読みにくい」という意見や，「定数名が英語表記の場合，定数名の間違えに気づきにくいが日本語だとわかりやすい」という意見があった．また，日本語プログラムでは，コメントをほとんど書く必要がなくなった分，「プログラムが見渡しやすかった」という意見もあった．

一方，「読み慣れた Java の構文の方がバグを見つけやすかった」という意見や，4.2 で述べたように日本語プログラムは，クラスと変数を名前で区別して書きづらく，「類似した名前の箇所があり，それが逆に読みづらかった」という意見もあった．その他には「専用のエディタが使いづらい」という意見もあった．

6.8. 実験の反省点

Java プログラムのデバッグ作業については，1 時間経過した時点で過半数の被験者がバグを 1 個だけしか見つけられなかったことから，各グループで当初よりもさらに対象範囲を全体の 4 分の 1 程度に絞り込むことにした．時間内でバグを 3 つ挙げることができた被験者は，アルバイトや趣味で相当量のプログラムを書いていたが，そのような被験者でも感想で「デバッグは難しかった」と言っていた．特に書店販売管理は，フレームワークの利用や抽象化によりクラス数が多くなり，また被験者がこの類のプログラムに不慣れだったこともあり，デバッグが難しかったと思われる．被験者を学生に留めず，ソフトウェア開発に従事する経験者などに広げることでよりスムーズに実験が実施できると考えられる．

また実験の都合上，日中連続して実験した．より厳密な条件で実験するには，十分な休憩を入れたり，実験ごとに実施日を変えたりして，実施方法を工夫する必要がある．

さらに，本実験は被験者の数が十分であるとは言い難い．既存研究でも被験者は 20 名ほどであることから，同じ実験をさらに多くの人数と組み合わせで実験することで結果が多少なりとも変わる可能性がある．ただし，既存研究も各グループの被験者数は，5 名程度で同じである．

7. おわりに

現在の日本語プログラミングは，単に識別子を日本語で書くことだけでなく，日本語らしい語順でプログラムを書くことができる言語処理系や日本語で命名されたライブラリを用いてソフトウェア開発することである．我々は，日本語プログラミング言語“プロデル”でそれらの開発環境を整備しつつ，日本語プログラミング言語によるソフトウェア開発を実践している．

本研究は，日本語プログラミング言語によって書かれたプログラムの可読性について，実験した．ソースコードからプログラムを理解したりデバッグしたりする作業に要する時間を測定する実験を行った．その結果，Java で書かれたプログラムと比較して日本語で書かれたプログラムは，理解やデバッグに要する時間が短くなる傾向があることを確認できた．このことから，日本語プログラミング言語を使ったソフトウェア開発がデバッグや保守の点で日本人技術者にとって何らかの利点があると言える．

本研究では，ソースコードを中心とした日本語プログラミングの利点を議論してきた．今後の課題として，仕様書とプログラムとの対比をした上で，ソフトウェア開発全体で日本語プログラミングの利点を議論すべきであると考えている．要求定義や基本設計，詳細設計で作成される仕様書から，それをプログラムとして実装し，テスト段階に至るまでの工程を一貫して日本語で行った時にどのような利点や問題があるか調べる必要がある．

謝辞

実験題材や研究に対しての助言を頂きました鷲崎弘宜教授に感謝いたします．また被験者として我々の実験に快く協力してくれた筧研究室の学生に感謝いたします．

参考文献

[1] “日本語プログラミング言語「プロデル」”, http://rdr.utopiat.net/, (2011).
[2] 馬場 祐人,筧 捷彦, “日本語で一貫したプログラミングの実践～プロデルを用いて～”, 研究報告ソフトウェア工学（SE）,2011-SE-171, No.17, pp.1-9(2011).
[3] 中川 正樹ほか, “日本語プログラミングの実践とその効果”, 情報処理学会論文誌, Vol.35, No.10(1994).
[4] 中川 正樹ほか, “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フェイス研究会資料, 43-1(1992).
[5] 大森 洋一ほか, “自然言語による仕様記述の形式モデルへの変換を利用した品質向上に向けて”, 情報処理学会論文誌,Vol.3, No.5(2010).
[6] 加藤木和夫ほか, “オブジェクト指向プログラム設計記述言語 OOPD とその記述環境”, 情報処理学会論文誌,Vol.43,No.5(2002).
[7] 今城 哲二ほか, “日本語プログラム言語“まほろば”の言語仕様”, 情報処理学会研究報告(SE)(2001).
[8] 木村 明, 片桐 明, “日本語プログラミング言語 Mind について”, 情報処理学会第 16 回プログラミング言語研究会, pp.25-32(1988).
[9] 杉浦 学ほか, “アルゴリズム構築能力育成の導入教育：実作業による概念理解に基づくアルゴリズム構築体験とその効果”, 情報処理学会論文誌, Vol.49, No.10, pp.3409‐3427 (2008).
[10] 中谷 多哉子, 兼宗 進ほか, “オブジェクトストーム：オブジェクト指向言語による初中等プログラミング教育の提案”, 情報処理学会論文誌, Vol.43, No.6 (2002).
[11] 西田 知博ほか, “初学者用プログラミング学習環境 PEN の実装と評価”, 情報処理学会論文誌, Vol.48, No.8 (2007).
[12] “日本語プログラム言語「なでしこ」”, http://nadesi.com/ (2011).
[13] “日本語プログラミング言語「TTSneo」”, http://tts.utopiat.net/, (2011).
[14] Ian Sommerville, Ron Morrison, “Software Development with Ada”, Addison-Wesley, 1987.
[15] MSDN, “C# プログラミング ガイド - 名前付き引数と省略可能な引数”, http://msdn.microsoft.com/ja-jp/library/dd264739.aspx, (2011).
[16] Sun Microsystems: Code Conventions for the Java Programming Language,http://www.oracle.com/technetwork/java/codeconventions-135099.html, (2011).
[17] http://rdr.utopiat.net/study/