OUKA
The University of Osaka
Institutional Knowledge Archive

Title	教育用プログラミング言語としての「言霊」と「ことだま on Squeak」の試み
Author(s)	岡田, 健; 杉浦, 学; 松澤, 芳昭 他
Citation	サイバーメディア・フォーラム. 2006, 7, p. 17-22
Version Type	VoR
URL	https://doi.org/10.18910/70223
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

教育用プログラミング言語としての「言霊」と「ことだま on Squeak」の試み

岡田 健*・杉浦 学*・松澤 芳昭*・大岩 元**
慶應義塾大学 政策・メディア研究科*・慶應義塾大学 環境情報学部**

我々は初心者に対するプログラミング教育において使用する環境として、日本語プログラミング言語を提案してきた。教育用プログラミング言語として開発されたプログラミング言語「言霊」と、Squeak を拡張する形で開発されたプログラミング環境「ことだま on Squeak」のそれぞれの特徴と問題点を解説する。

1．はじめに

コンピュータは現代社会に不可欠な存在となり、それに伴いコンピュータ教育の必要性も叫ばれている。高校教育では2003年より情報科目が必修となり、また中学校学習指導要領のなかでも技術・家庭で「情報とコンピュータ」が大きく取り上げられているのはその表れである。

これらの教科の授業実践例などを調べると、コンピュータを使えるようになる為の実用教育に重きが置かれている。Microsoft Wordなどのソフトウェアの活用方法の習得や、掲示板での発言やメールの書き方などのネチケットの教育が中心である。プログラミングを用いた情報処理教育はほとんど行われていない。

情報処理学会は「日本の情報教育・情報処理教育に関する提言2005」[6]（以下、「提言2005」と呼ぶ）を発表している。

「提言2005」において、情報処理教育に必要なのは「手順的な自動処理の構築」であり、その一手段としてプログラミングが挙げられている。図 1にその定義を示す。本稿では図 1の（2）に関する教育（以下、これを「アルゴリズム教育」と呼ぶ）について議論する。

[定義]「手順的な自動処理」の構築とは、次の一連の活動を言う。
（1）問題を自らの判断に基づき定式化し、その解決方法を考える。
（2）解決方法を、アルゴリズムとして組み上げ、自動処理可能な一定形式で記述した、コンピュータ上で実行可能なものとして実現する。
（3）実現したものが問題解決として適切であるかを検証し、必要なら問題の定式化まで戻ってやり直す。

図 1「手順的な自動処理の定義」

アルゴリズム教育において重要なのは、問題解決の方法をアルゴリズムとして表現する能力を育成することである。受講者がアルゴリズムを考案し、アルゴリズムをプログラミング言語の文法に従って記述し、そのプログラムを容易に実行できる環境が必要である。

我々は初心者に対するアルゴリズム教育のためのプログラミング環境として、日本語プログラミング言語「言霊」と、「ことだま on Squeak」を設計、開発してきた。本稿ではその特徴と問題点を報告する。

2．アルゴリズム教育における プログラミング言語の問題点

アルゴリズム教育の実践が難しい理由の一つに、初心者にとって既存のプログラミング言語は、読み書きが難しい点が挙げられる。受講者はアルゴリズムを学ぶために、お手本となるソースプログラムを読んでその意味を理解し、プログラムを書いて実行する必要がある。だが現実に行われている教育では、ソースプログラムを読んで意味を理解することは困難であり、かつプログラムを書いて実行することが困難である。

プログラミング言語の読み書きが問題になる事例として、分岐の概念を教えるために、肥満度の指標であるBMI値を算出するプログラムとして図 2を受講者に提示する状況を想定する。

```
int w = 168;
int h = 55;
double bmi = w / ( ( h / 100 ) * (h / 100 ) );
if( bmi < 18.5 ){
  printf(“あなたは低体重です.”);
} else if( bmi >= 18.5 && bmi < 25 ){
  printf( “あなたは標準体重です. ” );
}
```

図 2 分岐を教えるためのプログラム例

2．1．プログラムの書きにくさ

図 2のプログラムが表現しているアルゴリズムを理解して使いこなすことは難しくない。変数宣言文や代入文や分岐文の概念を理解した上で、これらの構文を用いてアルゴリズムを日本語で表現する

ことは、筆者の経験では多くの受講者にとって容易である。受講者に対して「どんなプログラムを作りたいの？普通の日本語で説明してごらん」と促すと、ほとんどの受講者は日本語で表現することが可能である。

ところが受講者が日本語で表現したアルゴリズムを、プログラムとして作成して実行するためには、多くのコストがかかるために難しい。受講者がプログラムを作成して実行するためには、エディタを用いてプログラミング言語の文法に従ってプログラムを記述して、ソースファイルとして保存し、コンパイルして、そして実行する必要がある。この一連の作業を行うためには、受講者はエディタの操作能力、ファイルとフォルダの概念理解、コマンドラインの操作能力、コンパイルコマンドと実行コマンドの操作能力、などが必要となる。

特にコストがかかるのがプログラミング言語の文法の理解である。図 2のようなプログラムを受講者が書こうとしたとき、文末のセミコロンを忘れたり、括弧を全角で書いたりしただけで、コンパイルエラーが発生してしまう。一般的なテキストプログラミング言語を用いてアルゴリズム教育を行うと、受講者はコンパイルエラーに悩まされることになり、アルゴリズムを落ち着いて考える状況ではなくなってしまう。

プログラミング言語の文法をちゃんと教えればよいという意見もあるが、我々が受講者に習得して欲しい能力はプログラミング言語の文法に従ってプログラムを記述する能力ではなく、問題解決のためにアルゴリズムを考案する能力である。アルゴリズム教育を円滑に行うためには、プログラミング言語の文法を学ぶコストはなるべく小さくするべきである。

2．2．プログラムの読みにくさ

多くの受講者にとって図 2のプログラムを読んでその意味を説明することは難しい。読解が難しい理由として、(1) 一般的でない記号が数多く存在すること、(2) 語順が日本語と異なること、(3) 述語がない文が存在すること、の3つを挙げる。

(1) 図 2には代入記号 (=) や、論理記号 (&&) や比較記号（>=）などの表記が多く存在する。初心者はこれらの表記を見ただけで意味を理解することは難しい。これらの表記が表す意味を教えれば良いという意見もあるだろうが、そもそもアルゴリズムを考える教育を行いたいのだから、プログラミング言語の文法を教える時間はなるべく少なくなるのが理想的である。このような初心者が見ただけでは理解できない表記はなるべく少なくするべきである。

(2) 図 2には日本語の語順が異なる表記が存在する。例えば文章を出力する**printf**命令が記述されているが。この命令は『“ あなたは低体重です ”という文章を出力する、という意味です」などと解説する必要がある。ところが日本語では「出力する」という動詞は文末に位置するのに、プログラムの記述では動詞にあたる「**printf**」という表記は文頭に位置する。こうした日本語とプログラミング言語の語順の相違は、理解を妨げる原因になると考えられる。

(3) 図 2には、述語がない文が存在する。例えば代入文を説明するときは「変数 w に168を代入します」などと解説するのだが、これに対応するプログラム表記は「**w=168;**」で表記される。「代入する」という述語にあたる表現がないため、受講者はこの代入文のプログラム表記の意味を掴むことが難しくなる。

こうした (1) ～ (3) の問題点がある原因は、図 2で使用されているプログラミング言語がプロフェッショナルを対象に作られているからである。プロであれば様々なプログラムの概念を理解しているので、様々な概念を少ない記号で表記した方が都合がよい。プログラムを記述する際にタイプ数が減るし、読むときも直感的に記号や短い表記からその意味を読み取ることが出来る。

だがこうしたプロフェッショナルにとっての利点は、初心者にとって理解を妨げる障壁である。初心者を対象にしてアルゴリズム教育を行う場合は、こうした問題が起こりにくい初心者用プログラミング言語を利用するべきである。

3．日本語プログラミング言語「言霊」

```
※各教科の点数を設定する
国語点数を作り、４９を代入する。
数学点数を作り、７３を代入する。
理科点数を作り、１００を代入する。
公民点数を作り、４５を代入する。
英語点数を作り、２５を代入する。

※５教科の合計を求める
合計点を作り、国語点数＋数学点数＋理科点数
＋公民点数＋英語点数を代入する。
※５教科の平均点を求める
平均点を作り、合計点÷５を代入する。

※平均点を四捨五入する
平均点を１０倍する。
平均点を１０で割り、その余り≧５ならば {
        平均点に１０を足す。
} をする。
結果を作り、平均点÷１０を代入する。
```

コード 1「言霊」のコード例
各教科のテストの平均点を求めるプログラム

アルゴリズム教育において使用するプログラミング言語として開発されたのが、日本語プログラミ

ング言語「言霊」[1][2][3]である。プログラムが日本語で記述されていて初心者が意味を読みとりやすい環境を提供できれば、プログラムの読解による学習を円滑に進めることが出来る。また日本語で記述することが出来るため、頭の中で考案したアルゴリズムをそのままプログラムとして記述することが出来る。

日本語プログラミング言語「言霊」のコンパイラは、Javaバーチャルマシンをターゲットマシンとして開発された。コード 1は、日本語プログラミング言語「言霊」のソースコード例である。コンパイルするとJavaクラスファイルが生成され、JavaVM環境で実行させることが出来る。

「言霊」の特徴を、4点述べる。

3．1．日本語の語順と文法で記述されている

従来のプログラミング言語の多くは欧米で開発されたために、プログラムは英語の語順を用いて記述されている。日本語の語順とは異なるため、日本人はプログラムを読解するときも記述するときも、頭の中で語順を変更する必要が生じる。これは一種の翻訳作業であり、特に初心者にとっては負担となり理解を妨げる原因になると考えられる。

「言霊」における文章表現は全て、正しい日本語の語順（動詞が末尾に記述される）を用いて記述されている。日本語の語順を用いることが出来ると、語順変更の負担がなくなるため、プログラムの読解も記述も早くなることが期待できる。

筆者の個人的な体験になるが、日本語でアルゴリズムを考え、日本語のままに記述することが出来ると、気持ちよさを体験できる。それは従来の言語と異なり翻訳する必要がないために負担が減っているためではないかと推測される。

3．2．略記を避けて、正しい日本語表現を用いる

2. 2節で述べたように、一般的なプログラミング言語はプロフェッショナルを対象に作られている。そのため多くの概念を少ない記号や表記で記述するための文法になっている。初心者から見て意味が読み取りにくい表現になっているのは、プロにとっての読み書きしやすさを実現するための略記が多く存在するからである。

言霊におけるプログラム表記は、こうした既存のプログラミング言語の略記を極力避けて、正しい日本語として記述出来るように配慮されている。

```
国語点数を作る。
国語点数（整数型）を作る。
国語点数を作成する。
国語点数（整数型）を作成する。
```

コード 2「言霊」の変数宣言例

例えばコード 2は「言霊」で変数宣言をする際に使える記述である。どの記述例でも日本語として正しく読める表現になっていることが分かる。

日本語として正しい表記をする理由を説明するために、比較対象として日本語プログラミング言語「なでしこ」[4]における変数宣言のコード例をコード 3に紹介する。「なでしこ」は末尾に述語（動詞）が記述されていないため、正しい日本語として記述されているとは言えないし、初心者にとって文章の意味は読み取りにくいと考えられる。

```
国語点数とは整数。
```

コード 3「なでしこ」の変数宣言例

3．3．動詞などの活用語尾に対応している

「言霊」は活用語尾を活用しても正しく解釈できるような言語仕様になっている[3]。前ページのコード 1の2行目では「～を作り、～を代入する。」という表現が用いられているように、「作り」が連用形で記述されている。「言霊」における動詞は、終止形だけでなく連用形、命令形を用いて記述することが可能である。

```
国語点数を作る。４９を代入する。
数学点数を作る。７３を代入する。
理科点数を作る。１００を代入する。
公民点数を作る。４５を代入する。
英語点数を作る。２５を代入する。
```

コード 4 活用語尾が無かった場合の例

日本語の活用に対応することで、意味のまとまりを意識して文章を記述することが可能になる。例えばコード 1の変数宣言が、コード 4のような文の羅列だったとすると、短い文章が連続する不自然な文体になってしまう。一般的に自然言語で文章を作るときには、意味のまとまりを形作り、複数の文を読点で連結して文章として記述する。コード 1の場合、変数作成と初期値代入は意味的に1つのまとまりを作っていると考えられる。「言霊」では連用形を用いて記述できるため、この2つの文をまとめて1つの文章として記述することが可能なのである。

活用語尾の採用は、意味のまとまりを意識して読みやすい文章を作る上で必要な言語の機能である。活用語尾の採用が読みやすいコードを記述させ、それがコードの可読性につながるのである。

3．4．文脈が利用できる

「言霊」では文脈を用いた省略表現が可能であ

る。読みやすい日本語表現を実現するために、文脈は不可欠な文法要素である。

仮に「言霊」の文法に文脈が無かった場合、コード 1の2行目は以下のコード 5のような表現になる。

```
国語点数を作る。国語点数に４９を代入する。
```

コード 5 文脈が無かった場合の例

表現が不自然で硬い表現になっているのは、文脈を用いた省略を用いていないからである。「国語点数」という変数を作成して、その変数に数値を代入することは前後の文脈から明らかである。だから普通に表現しようとするなら、2回目の「国語点数」は省略されるべきである。だがコード 5では省略していないために不自然な表現になってしまう。

省略する代わりに代名詞「それ」を使用することも出来る。「それ」を使った表現を、コード 6に示す。同様のアイデアは「なでしこ」でも実現されている。

```
国語点数を作り，それに４９を代入する。
```

コード 6 代名詞「それ」を使った例

なお、3. 2節では略記法を批判したが、文脈を用いた省略と略記法は全く別の問題である。略記法はプロフェッショナルが記述の労力を減らすことが目的であり、それによって初心者がプログラムの読解が出来なくなることが問題である。文脈による省略は、省略をしないと却ってプログラムが読みにくくなる表現が出てくるために必要な文法要素である。プログラムを読みやすくするために、文脈による省略を行うのである。

3．5.「言霊」の効果と問題点

日本語プログラミング言語「言霊」は、初心者を対象にプログラムの読解と記述を容易にすることを目的として作成された言語である。「言霊」を用いて慶應大学で4回のプログラミング授業を実施したところ、コードの読解が容易になり文法教育のコストが下がったことは実感できた。

だが「言霊」は初心者にとっても書きやすいとは言えないことが実践により明らかになった。その原因は日常言語における表現の幅は広いのに対して、日本語プログラミング言語の表現の幅は狭いことに起因する。例えば関数名として「現れる」という動詞を使うとき、代わりに「あらわれる」という平仮名表現を使ったり、「現われる」と異なる送り仮名を使ったりする。このように慣れ親しんできた日本語が使えると、ついつい日常的な癖で様々な表現を使ってしまう。

日本語表現を使ったプログラムが記述出来ると言っても、こうした表記の揺れなどの日本語表現を解釈する環境を構築するのは、我々プログラミング言語開発者にとってコストがかかる。実現するためには本格的な日本語処理を行う必要があるが、「言霊」はそこまでの機能を持っているわけではない。

4.「ことだま on Squeak」

日本語プログラミング言語の書きにくさを補う方法の一つとして、構文エディタを持ったプログラミング環境を用意することが挙げられる。構文エディタとは、文法的に間違ったプログラムが書けないように設計されているエディタである。

構文エディタを持ち、かつプログラミング教育に最適な環境として近年注目を集めているのが、Alan Kayの提唱するSqueak[5]である。SqueakではSqueak eToys（以後、eToys）と呼ばれるビジュアルプログラミング環境があり、構文エディタを用いてコーディングを行うことが出来る。

我々はeToysの利点を生かしつつ、目指すべき初心者プログラミング環境として「ことだま on Squeak」を設計・実装した。その特徴を以下に3点述べる。

4．1．構造エディタ環境によるプログラム記述環境

「ことだま on Squeak」は命令タイルを貼り付ける“タイルスクリプティング”によってプログラミングすることで、「言霊」が持つ書きにくさを解決している。

タイルスクリプティングの模様を図 3に示す。ユーザは命令タイルを貼り付けることでプログラムを記述していることが分かる。

図 3 タイルスクリプティングの模様

タイルスクリプティング環境では構造エディタを用いるために、2つのメリットが得られる。一つはユーザが自分で日本語を記述する必要がないため、3. 5節で述べた書きにくさを解決することが出来る。もう一つは、構造エディタであるためコンパイルエラーと無縁であり、初心者でも様々なコードを試行錯誤しながら記述することが可能になる点

である。

4．2．日本語の語順と文法で記述されている

「ことだま on Squeak」は、正しい日本語の語順（動詞が末尾に記述される）を採用している。図 4 は「ことだま on Squeak」における命令タイルの例である。「言霊」の読みやすい表記を受け継いでいる。

図 4「ことだま on Squeak」における
命令タイル

4．3．略記を避けて読みやすい日本語表現を用いる

「ことだま on Squeak」は略記を排し、初心者がタイルを読むことで意味が分かるようなプログラム表現を採用している。図 5 に「ことだま on Squeak」における条件分岐タイルのプログラム表現を示す。

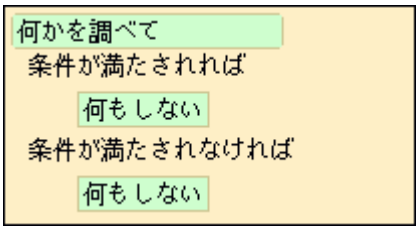

図 5「ことだま on Squeak」における
条件分岐タイル

「ことだま on Squeak」は、プログラム初心者が読んでも意味を理解できるようなプログラム表現を採用している。図 5 の条件分岐タイルを読めば、“とにかく何かを調べる”ためのタイルであることが理解できる。

「ことだま on Squeak」における条件分岐タイルの読みやすさは、条件節に調べるべきタイルを入れたときに、その効果が発揮される。図 6 は色判定タイルを条件節に入れたときのプログラム表現である。

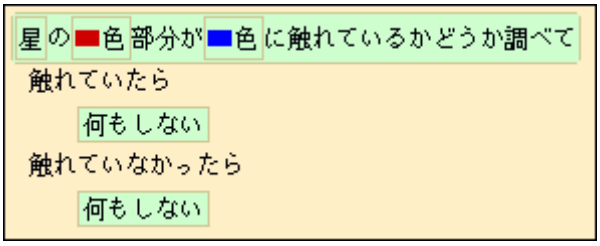

図 6「ことだま on Squeak」における
色判定タイルを条件節に入れたときの
条件分岐タイル

「ことだま on Squeak」の条件分岐タイルは条件をセットすると、条件分岐タイルの2行目と4行目において命令タイルの実行条件を補足する。図4の例では「触れていたら」「触れていなかったら」という表現になっている。

4．4．「ことだま on Squeak」の効果と問題点

「ことだま on Squeak」は、日本語プログラミング言語「言霊」が持つコードの可読性を受け継ぎつつSqueak-eToyの構造エディタを持つプログラミング環境を利用することで、読みやすく書きやすい日本語プログラミング言語を目指して開発された。

「ことだま on Squeak」を用いた実験授業を神奈川県藤沢市立村岡中学校で実施した。その結果、既存のSqueakを用いて行われた授業と比較して、文法の質問が減ったことが確認された。ちゃんとした日本語でタイルが記述されているため、受講者は特に質問をしなくてもタイルの意味を理解できるのである。その結果、受講者は本質的なアルゴリズムに関する議論に集中することが可能になった。

だが、一方で「ことだま on Squeak」の読みやすさ、書きやすさは初心者を対象にしている。少しでもプログラムが出来る人にとっては、「ことだま on Squeak」の日本語表記は長すぎて、本質的な部分を読み取るのに煩わしい要素が多い。またタイルスクリプティング環境も、初心者にとってはミスが起こらないために記述しやすいが、ミスを自分で解決する能力がある熟練者にとっては操作が面倒な環境である。

「ことだま on Squeak」は初心者を対象としてプログラムの読みやすさ・書きやすさを実現したが、中級者以上からは使いやすい環境とは言えないのである。こうした問題の改善は、今後の課題である。

5．まとめ

我々は初心者に対するプログラミング環境を模索し、プログラミング言語の読みやすさと書きやすさを改善するために日本語プログラミング言語「言霊」を開発した。また「言霊」は読みやすさを改善した一方で、依然として書きやすい言語とは言い難かった。そこで我々は初心者を対象として、読みやすく書きやすいプログラミング環境を実現するために「ことだま on Squeak」を開発した。

参考文献

[1] 岡田健、中鉢欣秀、鈴木弘、大岩元(2002) 日本プログラム言語「言霊」、情報処理学会2002FIT

[2] 岡田健、大岩元(2002) プログラミング言語としての日本語 慶應義塾大学湘南藤沢学会. Keio SFC Journal、 Vol.2 No.1 pp114-134

[3] 岡田健、大岩元(2005)日本語プログラム言語「言霊」におけるメソッドの記述方法. 情報処理学

会第46回プログラム・シンポジウム
[4] 日本語プログラミング言語「なでしこ」
http://nadesi.com/
[5] Dan Ingalls、 Ted Kaehlei、 John Maloney、 Scott wallace、 and Alan Kay (1997). Backto to the Future: The Story of Squeak 、 A Practical Smalltalk Written in Itself 、 Proc. of ACM OOPSLA '97、 pp.318
[6] 情報処理学会情報処理教育委員会(2005) 日本の情報教育・情報処理教育に関する提言2005.
http://www.ipsj.or.jp/12kyoiku/proposal-20051029.pdf