プログラミングにおける知的生産活動要素

A study about intellectual production activity elements in "programming"

桑原 悟*

コンピュータプログラミングは、人間の知的生産活動であり、一般に、「難しいもの」であると認識されている。本論文では、プログラムを学習する学生を観察した結果から、プログラミング学習において、その難しさを構成するものをいくつかに分類する。

分類した難しさは、そのそれぞれが、知的生産活動の独立した要素に対応するものと考え、何らかの手段によって、プログラミングから機械的に排除できる可能性のある要素と、排除できない要素に大別する。

このうち、排除できる可能性のあるものについては、排除のための手段を与える支援システムについて検討する。

一方、排除できないもの、すなわち人間の知的生産活動だけによらなければならない要素に関しては、プログラミング教育の立場から、この要素だけに集中して教育及び訓練のできる、e-learningシステムについて検討する。

Abstract : Computer-Programming is human intellectual production activity and, generally, is recognized to be a task with "difficulties". In this paper, I classified this "difficulties" in some kinds based on the observation on the students learning Computer-Programming.

As for the classified difficulty, I found two categories. One consists of elements that can be removed rather procedurally by some means and the other consists of elements that cannot be removed from this task or computer-programming which are rather essential activities of human intellectual production.

About the former, I examined the way of exclusion with proposal of a new programming environment or a tool on a computer.

About the latter, from a viewpoint of programming education, I proposed the e-learning system that can be concentrated on education and/or training of essential element of human intellectual production in Computer-Programming.

1．背景と狙い

1.1 背景

コンピュータプログラミングは人間の知的生産活動である。コンピュータの発達は社会におおきな影響を与え、一般市民の生活にも身近な存在になりつつあるが、プログラミングは、専門的であり、高度に知的であり、素養のある人材が行う知的生産活動であると認識されている。ひらたく言えば、コンピュータプログラミングは、難しいことがあたりまえのものであると考えられている。

そして、プログラミングに関する技術的興味としては、用途拡大又は実行系との親和性の観点からの新しいプログラミング言語の創出、プログラミング技法、コードの自動生成などがあるが、これらは、プログラミングを一部の専門家又は限られた数の職業人になる素養の人材が扱えばよいものとした暗黙の前提の下での技術的興味であるといえる。

実際に、自身を振り返ってみると、初心者のころ、プログラミングで実現できることの発展性、可能性を直感し、その難しさが故に興味を増し、課題を完成したときに、満足感、充実感を覚え、さらに高度な課題に取り組もうとする情動を揺さぶられたように記憶している。それは、ある種の優越感であったとも考えられ、限

*KUWAHARA, Satoru［情報システム学科］

られた者だけが取り組めばよいという前提を肯定していたように感じる。

コンピュータの利用が限られていた時代には、プログラミングも、その素養のある一部の人間が対応するという社会構造であり得たが、すでに20年以上前に、「将来のプログラマの不足」が話題となっている。このときの危惧がそのまま現実のものとなっているとは考えにくいが、コンピュータの利用は、ますます多方面に拡大し、また、既存のものも改定や修正、再構築が必要となるので、“プログラミング需要”は、今後も拡大していくことは間違いない。

1.2 狙い

拡大するプログラミング需要に応えるため、その生産性を上げるための研究や取り組みは盛んに行われているが、前述のように、その根本には、プログラミングは限定された人間で対応するものであるという考え方があると言える。

本論文では、プログラミング人口の拡大を妨げている“難しさ”に注目し、それを、人間の知的生産活動におけるいくつかの要素に分類して、そのそれぞれについて、次の(1),(2)により、プログラミングに携われる者の範囲を広げることを狙いとする。また、(1)はプログラミング作業の負荷を軽減することにもつながるので、この点で生産性の向上への寄与も目指す。

(1) プログラミングから、何らかの機械的手段で排除できる可能性のある要素については、その手段を検討する。

(2) 排除できない要素、すなわち、現時点では、人間の知的生産活動によらなければならないものについては、プログラミング教育の立場から、この要素だけに集中して教育、訓練できる、e-learningシステムの具体例を提案する。

2. 難しさと対応する個々の知的生産活動の要素

2.1 プログラミングの作業分類

人間が行う、知的生産活動としてのプログラミングは、おおよそ、図1左のような段階を踏むと考えられる。

まず、使用するプログラミング言語の文法知識と過去のプログラミングの経験から、構想を立て、次に、これに基づいて、コーディングを行う。コーディングを狭義のプログラミングとする考え方もあるが、本論文では、関連する知的生産活動全体をプログラミングと扱うので、これらもプログラミングの一部と考える。

コンパイラを用いる言語の場合、次に、コーディングしたソースコードをコンパイルし、実行する。インタプリタを用いる言語の場合は、翻訳と実行が合わせて行われる。

この段階で、大抵の場合、特に入門者の場合は、実行完了にいたらず、幾つかのエラー情報がコンパイラ又は実行系から与えられる。人間のプログラミング活動としては、このエラー情報を見て、構想又はコーディングの段階へ戻り、ソースコードを修正する。この道筋は、大抵の場合、複数回繰り返される。

修正後、コンパイルと実行の段階で、エラーが無くなった後、そのプログラムが目的とした機能を備えているかどうか、テストを行う。この段階で、目的の機能が実現されていない場合、やはり、修正が必要になるが、これは、コーディングの段階での修正の場合と、構想段階への戻りが必要な場合、及び、その中間的な修正の場合があり得る。これらの各修正は、人間の知的生産活動という視点から見た場合、幾つかの種類に分類できる。

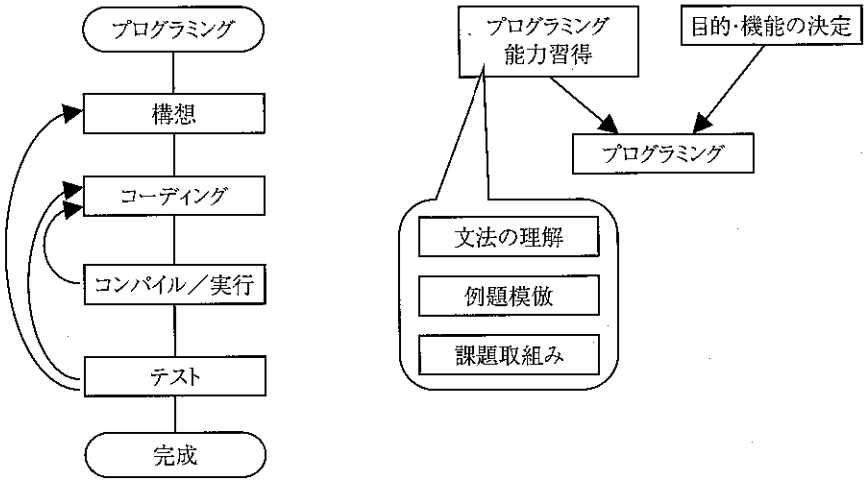

図1　プログラミングの作業段階の分類とプログラミングの位置づけ

また、ここでは、プログラミングの主要な作業の段階分けを行ったが、そもそも、プログラミングを行う状況にいたるまでには、プログラムで解決しようとする課題が存在し、そこから、プログラムの目的と機能が決定されていることは当然であり、かつ、プログラミングを行う人間は、プログラミング以前に、プログラミング能力の取得をなんらかの形で済ませている必要がある。

このプログラミング能力の取得を考えたとき、プログラミングは、図1右に示すような要素から構成されることは間違いないといえる。しかも、この各段階での実際の手順又は操作としては、実際にプログラミングを行うこと、すなわち、能力習得の手段としてではあるが、前述のプログラミングをこの段階でも行わざるを得ないことが分かる。

2．2　知的生産活動の抽出

プログラミングをいくつかの知的生産活動の要素に分離する方法として、本論文では、プログラミングの能力習得過程において、課題達成までのどの段階をどれくらいの学習者が達成するかを観察する方法を試行する。これは、前述のように、この過程において、すでにプログラミングの各作業段階が実践されること及び、入門者がここで習得の難しさを認識するものが、知的生産活動の単位要素として扱えると考えられるからである。

2．3　プログラミング学習の被験者

プログラミングの習得過程に注目するとき、本論文の立場からは、被験者の選択が重要となる。一般に、プログラミングに関しては、個々の能力には自ずと差はあるが、理工学系の学生の方がその素養を備えていると理解されている。そうなると、理工系の学生は、最終段階へ達成する能力を取得する可能性が高く、要素を分離するための被験者としては実は、適当ではない。

したがってここでは、非理工学系学生へのプログラミング教育から、学習者にとって、プログラミングの難しさとなっている事象を抽出し、そこからプログラミングの知的生産活動の要素を明らかにすることとする。

新潟国際情報大学情報文化学部情報システム学科（以下、本学科という）は、数学を選択しなくてもよい入学試験の形態を取り、実際に英語と国語の2科目を選択して合格している学生が存在する。別の言葉で表せば、文系と理系の枠を取り除くという趣旨に基づいた学科である。

したがって、本学科は、理工学系の学生の母集団ではなく、より多様な素養の学生からなる母集団と考えることができる。

一方で、本学科では、当然ながら情報システムを教育し、プログラミングを直接教える科目及び関連する科

目において、学生にプログラミングを教育している。

本学科におけるプログラミング関連の授業はいくつかあるが、学生の理解や課題達成の過程に存在する難しさ、つまり、それがゆえに自身とプログラミングの関係に対して学生が否定的となる要素は、教育経験からいくつかに分類することができる。

3．難しさの知的生産活動要素への分離

3．1 文法違反の発見

プログラミングにおいては、コンパイラ、インタプリタ又は実行系が、エラーメッセージとして表示するものから、そのエラーの原因を特定し、修正する作業が必要である。このとき、エラーメッセージが示すものが、直接修正すべき個所を示す場合と、エラーメッセージからは、修正すべき個所が直接読み取れない場合とがある。

現在のプログラミング言語のコンパイラ、インタプリタ又は実行系の構造から考えて、人間にとってのエラーメッセージの的確さの度合いは、後者の状況を多く起こさざるを得ないといえる。

後者の場合、通常、プログラムをする人間は、そのプログラミング言語に関する他の知識などから修正すべき個所を類推すること、試行錯誤的なコーディングの変更を行って、結果を観察すること又は、これらの組合せ及びその繰り返しを行って、修正すべき個所を特定していく。

学習者は、初期の学習課題においては、すでに教材に示されたコーディングをそのまま入力して動かす課題、それを参考にして指示された限定的な変更を加えて動かす課題も与えられる。

被験者がこの段階の作業を完成できない、すなわち修正すべき個所の特定にいたらない場合も見うけられる。また、指導者の助言や指導者による修正個所の特定の過程を見せても、別の類似のエラー個所で、やはり、この段階の作業を完成できない場合もある。

このことから、エラー発見の作業は、プログラミングにおける難しさを構成する知的生産活動の一要素であるとしてよいことが分かる。

そしてこれは、プログラミングの学習において、一つの障壁となり、本学科の学生では、ここで、『プログラミングは自分には向いていない』、『プログラミングが嫌いである』と思うようになる学生が観察される。

3．2　意図と記述の齟齬の発見

前述のエラーとして表示されるものは除き、プログラムが、プログラミングした人間の意図とは違う振る舞いをする種類の誤りがある。これには、言語によってさまざまな場合があり得るが、名称、記号の誤記、抜け又は過剰及び、記述位置の誤りなどがあり得る。

たとえば、括弧の位置、終止符の抜け、符号の誤った記述（＋を＋＋など）が、編集中の誤操作などにより引き起こされる例、あるいは、変数を予め定義する必要の無いプログラミング言語で、変数名の綴りの一部を誤ると、別の新しい変数として扱われ、意図した結果を得られない例などがこれにあたる。

エラーとして表示されないので、プログラム作成者は、プログラムの振る舞いから誤り個所を特定することになるが、作成者は、正しく記述した積もりになっていることもあり、すぐには発見されない場合が多い。

一文字の綴り誤りを発見するのに、プログラムの実行結果とコーディングリスト、デバッグのためのツールを用いて誤り個所を特定することになり、これもプログラミングの難しさの要因の一つになっている。

3．3 構想の誤りの発見

プログラミングの学習において与えられる課題は、初心者に対する極めて初期のものでない限り、直接コーディングに対する指示である場合は少ない。したがって、学習者は、これまでで得たプログラミング以外の知識、たとえば数学の公式などと、プログラミングの学習で得た知識すなわち、利用できる演算の種類、用意されている関数、手続き型の言語においては、場合分けや繰り返し処理の記述などの知識を用いて、課題解決の構想を立てる。

当然、この段階においても、人間の行う作業として、完成できない場合や誤りを生じる場合がある。これは、明らかに前述の難しさとは独立であると考えられ、一つの知的生産活動の要素と捉えることができる。

3．4　論理の誤りの発見

前述の“構想の誤り”は、ある一つの課題を考えたとき、プログラムする人間と使用する言語との組合せによって、結果が異なることが考えられる。被験者Aが、言語Xでプログラミングを行う場合には誤りを犯さず、同じ被験者Aが、言語Yで同じ課題の解決のためのプログラミングをする場合には誤りを犯す、あるいは、完成にいたらないという事態が起こり得るからである。これは構想に使う言語の知識の誤りに原因をもとめることができる。

これとは別に、言語によらず、被験者によってだけ誤りが起こる状況を考える。これは、被験者の論理的思考の能力に関連するものと考えられ、知的生産活動を分類する立場からは、このことは、論理的思考を別の一要素と考えてよいことになる。

しかし、構想の誤りとこの論理の誤りを、実際のプログラミング作業とその結果から、完全に識別することは難しいとも考えられる。

4．各知的生産活動要素の検討

ここでは、3．で上げた四つの要素について、機械的に排除できる可能性のある難しさであるかどうかを検討する。

4．1 文法違反の発見

前述のように、コンパイラや実行系の表示するエラーメッセージは、多くの場合、プログラミングをする者が直接知りたい情報を与えてはいない。コンパイラを作成する立場から見ると、コンパイル時又は実行時のエラーメッセージを、プログラマが直接知りたい内容にすることは困難であるといえる。

そこで見方を変え、コーディングが文字入力による自由記述の形態で行われることに、この種の誤りが発生する原因があると捉えた場合、これに対しては対策が比較的取りやすい。

自由記述を制限し、選択肢からの選択をすることでプログラミングを行う環境がその一つの答えとなる。この環境を使用することにより、予約語と同じ名称の変数の使用、関数の引数型や演算においての型の不整合なども防止することができる。

既存のプログラミング言語に対し、このような環境を用意する場合は、詳細な検討が必要となるが、そのサブセット又は、ここで上げた他の各要素にも配慮した新たなプログラム言語の創出をも視野に入れると、この要素は、プログラミングから排除できる可能性のある難しさであるということができる。

4．2　意図と記述の齟齬発見

意図と記述の齟齬に関しては、完全にはこの要素を排除することはできないが、4．1で述べたプログラミング環境は、この要素に対しても有効である。特に、変数名の綴り誤りは、選択肢からの選択で回避できる可能性が高い。

この要素は、プログラミングにおける難しさとして、相当に軽減できる可能性があり、そのための支援システムは、本論文の趣旨に合致するものである。

4．3 構想の誤りの発見

プログラミングにおいての構想の作業は、プログラミングの本質の一つであると考えられ、これを排除することは難しい。

しかし、特に初心者にとっては、言語のマニュアルなどの書籍を常備することの煩わしさや、それを参照する手間といった、この要素に付随する事象に関しては、プログラミング環境における適時の表示やオンラインマニュアルなどで軽減することができる。このことは、本論文の趣旨に合致するものである。

4．4　論理の誤りの発見

論理の構築は、プログラミングの本質又は根幹をなすものと理解でき、この要素を排除することはできない。

したがって、この要素は、プログラミング教育のシステムで扱うべき要素と考えることができる。このとき、この教育システムでは、当然ながら、前述の他の要素の排除又は軽減が実現されていることが必要な要件の一つとなる。

5．プログラミング環境の検討

4．で述べた難しさの一部を機械的に排除又は軽減するためのプログラミング環境としての要件は、自由記述を制限することである。具体的には、ドラッグ・アンド・ドロップなどのマウス操作によって選択肢から選択してプログラムを作成する形式をとることである。

この場合の選択肢には、予約語、演算子、その他の予め意味をもたされている記号及び、プログラミングの過程で決められた、利用者定義の変数名、関数名も含まれる。当然、利用者が定義するものは、初期の選択肢には存在しないので、専用の入力フィールドを設け、ここに、キーボード入力する形式をとる。

この入力フィールドは、利用者定義のものとして用いることができる範囲を逸脱していないか、たとえば、予約語をそのまま変数名としていないかなどを検査する機能を有する。

さらに、これに加えて、プログラミング環境として考慮すべき点として、次のことがあげられる。

プログラミング言語の中には、実行系との親和性の観点からの表記の選択肢をもつものもある。しかし、ここで提案するプログラミング環境では、実行速度、メモリの占有状態などの実行系との関係は、プログラマが意識してコーディングに取り入れるのではなく、オプティマイザとして別のシステムのもつべき機能であるという立場に立つ。このことにより、特に初心者の知的生産活動の負荷を軽減し、かつ、熟達者と同じ程度の効率と実行系との親和性を志向する。

効率と実行系との親和性に関連する事項は、オプティマイザに担当させ、同じ機能又は実行結果を得るコーディング上の選択肢を少なくすれば、初心者には有利である。しかし、熟達者の場合、同じ機能又は結果であっても、アルゴリズム記述上の意味によって、コーディングの選択肢の中から使い分けることを好む傾向がある。

この使い分けは、自身及び他の特に熟達者による修正や再利用時にコーディングを人間が読む必要が生じることを意識しているからと考えられる。

これは、自然言語において、同じ内容を、子供に聞かせるために平易に記述する場合と、大人同士又は専門家間でやり取りする場合の記述とは、異なることになぞらえて考えることができる。

本論文の立場は、熟達者以外にも対象を広げようというものであるが、熟達者に遠ざかられるようなものを提案することではないので、コーディング上の選択肢は残し、初心者に与える最小セットのコーディング規則からは除外し、初心者には使えない設定ができるようにすることを考える。

6．プログラミング教育システムの検討

6．1　外部仕様

プログラミング教育システムは、4．及び5．で検討した要件はすべて満たすことが必要である。ここではさらに、教育システムとしての追加要件について検討する。

本論文では、教育システムで教育をする対象は、プログラミングに最初から否定的感情を抱いている者あるいは、否定的感情を抱いてしまった者もその範疇に入れる。しかし、その対象が現代の学生層であれば、彼らは、いわゆるテレビゲームに対しては興味を示す年代であるといえる。

したがって、ここで提案する教育システムは、テレビゲーム様の操作を導入する。

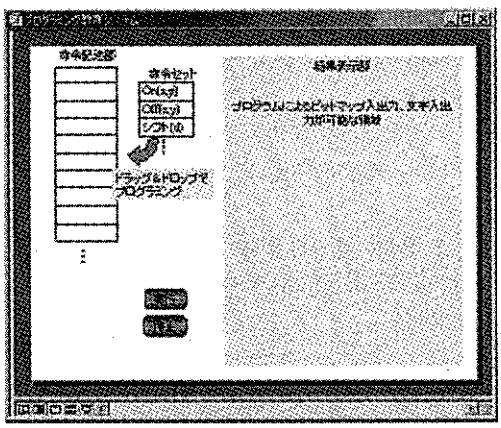

図2　教育システムの外観

具体的には図2に示すように、ウィンドウ内左側にコーディング領域を設け、命令などの記述は、選択肢を表示して選ばせることで行う。関数の記述では、引き数記述の位置にカーソルを合わせると、引き数として正しい型の変数名の入力や、整数型の場合では、数字キーの直接入力だけを許し、他は表示や音でエラーであることを知らせる。また、ウィンドウ右側はビットマップ及び文字の入出力が可能な領域とする。

学習者が興味をもち、かつ、プログラムの特徴である手順的な自動処理のテーマの例としては、図3に示すような、ゲーム様のものが考えられる。

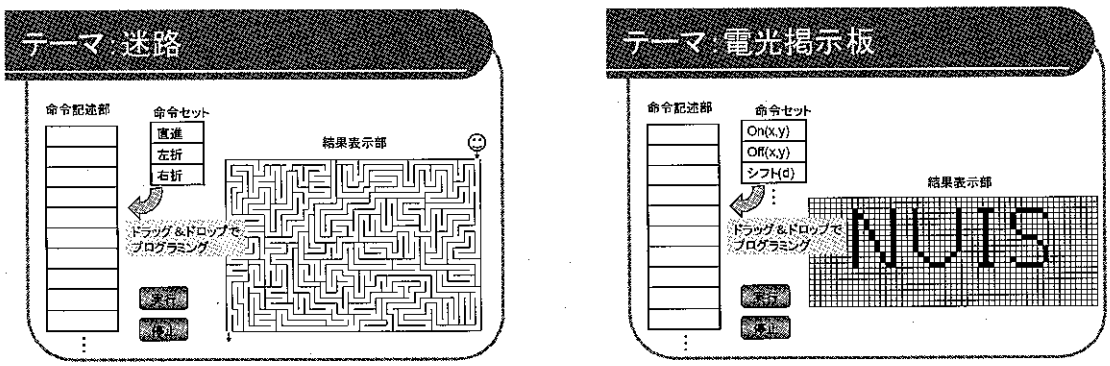

図3　教育システム上での個別のテーマの実現

6．2　内部設計

この教育システムは、学生など教育対象となる人間が使う個別テーマ層、そこにさまざまなテーマを実現するための命令セットと入出力を処理するテーマ記述層、これら2層の記述を実行するためのプラットフォームの3層で構成する。この教育システムの論理的構造を図4に示す。

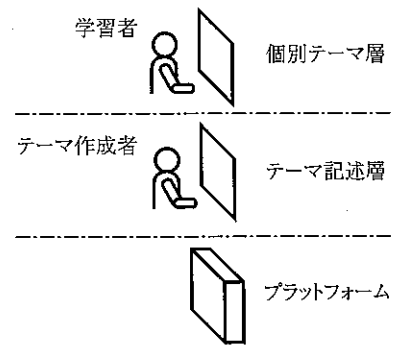

図4　論理構造

個別テーマ層は、前述のような個々のテーマが設定された、学習者が操作する。個別のテーマは、基本的には共通の複数のオブジェクトで構成される。文字の入出力、ビットマップ情報の入出力及び、入力された命令を解釈、実行する系である。

テーマ記述層では、個別のテーマを構成するための記述言語を導入し、これによる記述を行うことになる。この記述は、通常、教員など教育する側が、実現したいテーマごとに作成する。それらの記述は、たとえば、他の教育機関での利用も可能なように、テンプレートとして配布し、受け取った側のこの教育システムで実行することも可能とする。

この層を記述する言語を解釈、実行するために、プラットフォームの種類ごとに汎用プログラミング言語で作成されたサブシステムが存在することになる。

このように構成することで、前述のような一見まったく違う複数のテーマを実現するための多様な入出力と処理が、プラットフォームからの独立性をある程度保ち、また、個別のテーマを比較的簡単に作成でき、さらに、そのテーマは、テンプレートとして流通可能なものとすることができる。

図5, 6は実装構造である。実装構造1は、論理構造と同じ考え方である。テーマ記述層は、プラットフォームごとに作成する必要があるが、個別テーマ層は、プラットフォームから独立させることができる。実装構造2では、テーマ記述層と個別テーマ層の両方をプラットフォームごとに作成する必要がある。これは主に、個別テーマ層の処理速度を速くする効果があり、プラットフォームの性能により利用者に負担のかかる処理速度となる場合はこの構造を選択することとなる。また、実装構造2はライセンス管理をする場合などに適した構造

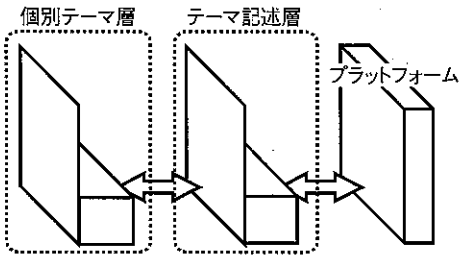

図5　実装構造1

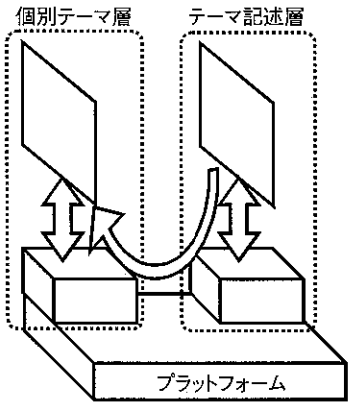

図6　実装構造2

である。

ライセンス管理を行う場合、その目的は次のとおりである。

(1) 個別テーマをテンプレートとして他者で利用する条件となる互換性を管理する。

(2) 既存の個別テーマをテンプレートとして利用するだけの環境とテンプレートを作成できる環境を分離し、それぞれを対価の異なった有償ソフトウェアとして、独立管理する。

(3) 優良テンプレートの創世と流布を促すため、テンプレートの作者が価値に応じた対価を設定することを可能とする。

7．今後の課題

本論分では、プログラミングの難しさについて考察し、その排除、軽減のための支援機能をもつプログラミング環境について検討した。また、プログラミングの本質とも理解できる、機械的には排除できない難しさについては、その教育に傾注できるシステムの構想を提案した。

今後は、このプログラミング環境の展開、新たな言語創設と既存言語の新たなプログラミング環境への適用について、詳細に検討する。また、教育システムについて、その詳細設計、ポートフォリオ機能やネットワーク接続に関する詳細の設計、作成を行い、実際の教育への適用に関して、計画と実施及び効果の測定を行う予定である。

本論文掲載の第10回記念号の紙面の都合上、参考文献の掲載を割愛させて頂きました。