ビジネス系学部におけるプログラミング言語教育の授業デザイン

木下和也

Email: xkino@nakamura-u.ac.jp

中村学園大学流通科学部

◎Key Words　授業デザイン　行動主義　社会構成主義　認知的徒弟制　アクティブラーニング

1.　はじめに

大学におけるビジネス系学部での授業では、プログラミング言語を学び、それをどう活用するかを考えさせるべきであろう。アルゴリズムを重視した数値計算や検索手法などを学び、同様の例題を解くことに集中するだけでは、ビジネス系学部の専門教育の内容とは乖離してしまう。たとえば、いわゆる現代の経営情報システムである ERP に代表されるように、全社的にデータを共有し業務の効率化を支援するシステムなどを引き合いに出し、ビジネスと情報システム及びそれを構築するために必要なプログラミング言語との関係などを示すべきであろう。また、グループウェアや社内 SNS が同様にビジネスには必携のツールであることや、ビジネスそのものを支援するシステムとして通販で用いられるカートの仕組みなど、学生にはある程度想像できるシステムを念頭にプログラミング言語の重要性を示すべきである。

なぜなら、後述するように知識と（それが有効に使われる）状況が分離された状態では、使えない知識として定着してしまい、そのことによってプログラミング言語を学ぶ面白さが失われることに繋がりかねないからである。彼らは大学にプログラミング言語を学びに来たのではなく、体系的にビジネスを学びに来ていることを忘れてはならない。

では、そのような学生にプログラミング言語教育を行うべきか、学習に関する理論と。それに基づく筆者が行ってきた実践を例に論じたい。

2.　第一歩となる言語の基本的知識

様々なプログラミング言語が存在するが、どのような言語も、判で押したかのように最初に学ぶ内容は "Hello world"である。つまり、print 文などによる画面出力である。言語にもよるが、キーボード入力と画面出力、またはプログラム中にデータを記述するところから変数を学び、合計や平均値などを例に for 文や while 文などの構文を使った繰り返し処理を学ぶ。また if 文、if - else といった構文を使った条件分岐も最大値と最小値の求め方などを例に学ぶ。繰り返しと条件分岐を完璧に理解することがアルゴリズムの第一歩と考えてよいであろう。

第 2 に 1 次元及び 2 次元配列へと変数の扱いを拡張することで、より多様な処理を可能とするアルゴリズムを身につけることとなる。

筆者がこれまでに担当してきた学生（平均的な学生と考えている)に関しては現実には、15 回の授業で 2 次元配列までを完全に理解して応用できるようになる学生は少数派であり、全体を対象に理解させることを考えれば、1 次元配列を用いたバブルソート程度が限界であろう。しかし、このレベルまでのアルゴリズムを身につけていれば、さらに継続して履修するプログラミング言語の授業では、ビジネス系学部らしいシステム開発を目的とした授業を展開できると考えている。

3.　繰り返し学習の効果

基本となるアルゴリズムの学習に効果が期待できるのは繰り返し学習である。繰り返し学習にはいくつかのタイプが考えられ、そのひとつは、同じ内容を複数回繰り返すことで、知識を定着させるというものである。しかし、この方法は授業内容を薄くしてしまい、教える知識量が少なくなるというデメリットがある。

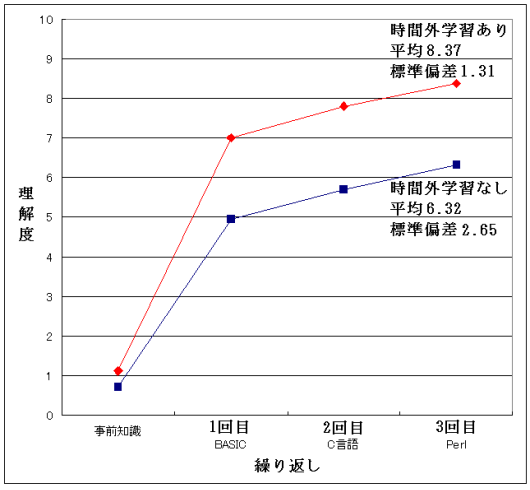

図 1　繰り返し学習による理解度の変化[3]

筆者の授業で測定した結果によれば、ある程度の効果は確認されている。図 1 に描かれているように、言語を替えながら、同じアルゴリズムを学習させたところ、自己申告ではあるが、時間外学習の有無により程度は異なるものの、双方とも理解度は上がっている。

第二の方法は時間外に繰り返し学習させることである。これは類似の多数の課題を出題し、解かせることで実現する。いわゆるドリル型の学習であり、CAI の利用などにより、一定の効果が期待できる。これら繰り返し学習は、いわゆる行動主義的学習観に基づく授

業デザインと捉えることができ、これまでの実践の経験から、基本的なアルゴリズムの教育に関しては効果が期待できると考えている。

4. 互恵的教授法によるグループ学習の効果

互いに教え合う行動が、互いの実力を向上させることは、経験から想像しやすい。この行動を構造化した学習法が互恵的教授法である[2]。

互恵的教授法は予測、明確化、質問、要約という4つの活動で構成されている。ただし、これらの活動は段階や順序を示すものではなく、それぞれが相互に作用しながら学習内容の理解を促していくものである。これをプログラミング言語教育に応用できないかを模索し試みている。それぞれの活動をプログラミング言語特有の学習内容と場面に当てはめてみたのが次の例である。

予測とは、これまでに学んできたアルゴリズムや記述方法を手がかりに、どのようにプログラムを作っていこうかと考え合う活動である。

明確化は、わからない処理やアルゴリズム、記述方法、さらにはその組み合わせ方など、曖昧な部分をはっきりとさせる活動である。プログラムは曖昧さがなく明確なコンピュータへの指示であるから、この活動はプログラミング言語を学ぶ上では特に重要だと思われる。

質問は、教材や教科書にあるプログラムの例を元に、類似する問題を作り、出題し合う活動である。質問を行うには自身が理解している必要があり、この活動を通して、自身の知識を確認することができる。人に教えられるくらい内容を理解しなさいと指示するのは、こういった効果を期待しての発言である。

要約は、内容をまとめることであり、プログラミング言語学習であれば、プログラム全体が何を意味するのか全体の流れを把握して、一言でその処理概要を説明できるようになることといえる。プログラムを上から下へと記述されている順番にステートメント毎にしか説明できない学生は、実際にはそのプログラムが何を処理するのかを明確には理解していないことが多い。繰り返し処理や関数毎に処理を把握し全体としてどのような処理を行っているのかを明確に説明できるようにしなければ、相手に伝わる説明はできない

教師が学習者に教える姿をモデルとして、それを自身でも実践しようとすることは、学習者同士で相互に作用し合う活動を通して自立した学習を目指すことと同義であり、プログラミング言語学習には適した学習スタイルであると思われる。

もちろん授業時間だけでは知識もスキルも定着しない、どうしても時間外の学習に期待せざるを得ないという性質の授業内容であるから、学生間での相互作用は必要であるともいえる。

筆者の実践では、授業中の私語を認めることであり、さらに私語を奨励することに効果があった。とは言うものの、授業と関係のない話を奨励しているわけではなく、学習者相互で声を出して教えあう時間を認めているという意味である。

現実問題として、一人で黙々と学習してスキルアップできる学習者とそうではない学習者がいる。
しかし、一人で学習することで効果が出てこないような学習者であっても、他人の助けを借りながらであれば、しっかりとした知識とスキルを身に付け、同じ水準にまで達することができるということは経験からも納得できる教員は少なくないと思われる。

大学の場合、オフィスアワーを設定し質問などの学生対応を行っている。プログラミング言語学習に関しても、その時間を利用して欠席者や学習が遅れている学生の対応を行っている。この時間を利用する学生によれば、「授業ではよくわからなかったが、少人数で話を聴き、間に質疑応答を入れながらの学習であれば、理解しやすい」という感想をもつようである。また、例題の解答やその後の実習状況および試験結果から、オフィスアワーで学んだ学生については、授業だけで理解できた学生と同じ水準に達していることは確認できる。つまり、人の手を借りながらではあるが、最終的には自分の知識として定着させていると判断できる。

しかし、オフィスアワーに質問に来る学生は多数派とはいえず、同じ能力はありながらも、その機会を失い、内容を理解できないままに終わる学生がいることも事実である。

そこで、筆者の授業では、このオフィスアワーの効果を授業中に取り入れ、一定時間、声を出して周囲の学生と教え合うことを促している。また、座席を離れて質問したり教えに行ったりしてもよいとも伝えている。学習者への聞き取り調査によれば、そのような時間に意味があるか（とくに学習効果があったのか）との問いに、「ある」と答える割合がきわめて多かったことは印象に残る。

5. グループ学習によるシステムの構築

プログラミング言語の学習を専門科目として捉えるのであれば、それはビジネスに関連したシステム構築を意識したものになろう。

筆者の実践例では、グループ学習によるシステム構築に一定の効果があったと考えている。基本的なプログラミング言語の知識を持った学習者に模擬的な会社としてグループを作らせ、ファシリテータとしての教員から出題される課題をグループで取り組ませる方法である。

前述の繰り返し学習によるプログラミング言語の習得は行動主義的な学習観に基づく授業デザインといえる。それに対し、学習者がグループ学習のように他の学習者との関係を含む様々な環境との相互作用の中で知識を定着させることを念頭に学習を行う場合、それはヴィゴツキーの主張する社会構成主義的な学習観に基づく授業デザインといえる。

ピアジェによる構成主義は、人間の学習および発達を個人と環境との相互作用の中で捉えており、あくまでも他者は想定されていないが、これに対して、社会構成主義は社会的文化的な背景、他人との相互作用を取り込んで、社会的な相互交渉の過程でこれらを捉えている。つまり、周囲の学習者との相互作用を重視して授業を行うことはヴィゴツキーの提唱する社会構成主義に基づく授業デザインの特徴となる。

もともと、プログラミング言語の学習は知識の積み上げをしながら段階的に進めていく授業であるため、オムニバス形式で進めていく講義式の授業と異なり、途中で内容について行けなくなる学生が現れることがしばしばあり、その学生達にどのように対処すべきかが大きな問題であった。このような学生への対応策として他者との相互作用の中で学ぶ効果が期待できる授業デザインこそ、社会構成主義的学習観に基づく授業だと考えられる。

グループ学習が効果的である理由は、ヴィゴツキーの提唱する最近接発達領域の理論でも説明できると考えている。これは、学習内容が周囲からの支援を必要とせず自身で解決できる水準と、その学習内容の理解過程に教師や仲間といった周囲の他者からの援助が介在することで達成される水準との間に存在する領域のことである。つまり、プログラミング言語に関してついて行けなくなる学生にとっては現時点でのつまずきが真に難解な内容というわけではなく、手が届きそうなところにある内容だという意味である。これを学習者と教員の双方が放置すれば、つまずき以降の授業には全くついていけなくなるが、何らかの形でその知識を補完する活動があれば、さほど負担なく内容を理解し知識が定着していくのである。プログラミング言語学習にはこのような効果を期待してグループ学習を取り入れるべきであろう。

6. 認知的徒弟制によるグループ学習の効果

プログラミング言語学習をアルゴリズムの学習から一歩踏み出し、実用的なソフトウェアやシステム開発といった目標に置き換えることは、より効果的な言語学習に繋がると考えている。前述のように、アルゴリズム学習だけでは、それが何の役に立つのかが認識できず、興味を失ってしまう可能性もある。かといって実務的なソフトウェアを簡単に構築することは難しい。それは、アルゴリズム学習で得た知識がソフトウェアとして機能することとは分離されて知識として定着した状態のままであるからである。知識と状況が分離された状態では、現実には実践的なシステムを構築できない。それを解決する手法が認知的徒弟制といえる。つまり、職人の修行課程と同じで、知識を状況に埋め込まれた状態にして学習させることである[(1)]。

認知的徒弟制には、4段階の学びがあり、それぞれ、弟子が親方の仕事ぶりを見て学ぶモデリング、親方が弟子に技を指導するコーチング、弟子がスキルを身に付けたかどうかを確認し自立させるスキャフォルディング、そして親方が手を退くというフェーディング、という段階である。

アルゴリズム学習は、いわば様々な処理のエッセンスを抜き出して抽象化した知識であり、そのままでは何に使えば良いのか理解しにくい。これはいわゆる脱文脈化された状況であり、その意味で真正さ、つまり本物らしさが感じられない状況である。そのため、簡単ではあるが、実用的と思われるシステムを開発するという「仕事」を与える。この学びは具体的なシステムであれば何でもよいわけではなく、ビジネス系の学部にふさわしい例にするべきである。たとえば、簡単な在庫管理システムや、顧客管理システムといったものはビジネス系学部の学生にとってより真正なものと感じられるであろう。

図2　開発した模擬システムの発表（1）

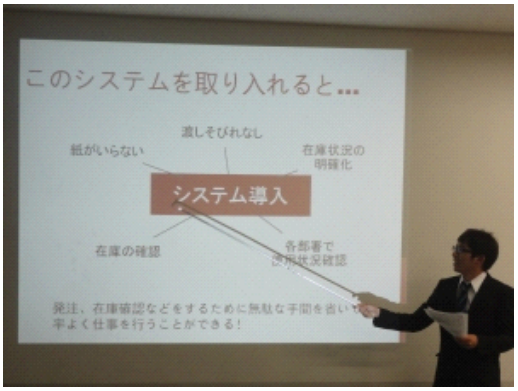

図3　開発した模擬システムの発表（2）

図4　開発した模擬システムの発表（3）

筆者は段階的に難度の高い課題を与え、実践共同体たる学生の学習グループの実力を段階的に向上させようと実践を行ってきた。その具体例は、先に挙げた在庫管理システム、顧客管理システムの他、それらを組み合わせたレンタルビデオ店の業務システム、簿記の知識を具体化する会計ソフトなどの課題である。これは他の専門科目との関連を想起させる意味でも真正さをより明確にしていると考えられる。また、開発に先立って、業務分析と設計、および開発マネジメントに関する知識に触れ、単にプログラムを作ればよいのではなく、業務プロセスを理解した上で、プログラミングスキルを活用するというビジネス系学部ならではのプログラミング言語の捉え方を強調して伝えている。

さらに、完成したシステムは図２，図３および図４のように、グループ毎にコンペ形式で発表させ、相互に競争意識を持たせている。筆者はこのグループを「会社」と称して実施している。学生はそれぞれの会社に所属するエンジニア兼営業担当であり、各グループは相互に競争相手であると認識して学ぶのである。

7. 課外活動の活用

ここまでに述べてきた学習方法は、いわばアクティブラーニングの一形態をなしている。アクティブラーニングをその構造の自由度と活動範囲で分類すると、知識の定着を目指すための活動であれば、構造の自由度は低く、活動範囲は狭い。繰り返し学習などはこれに相当するだろう。それに対してグループ学習方式は構造の自由度が高く活動範囲もある程度広くなる応用志向の授業といえる。しかし、さらにプログラミング言語知識の活用を目指すまでに高めるのであれば、より活動範囲を広げた実践が必要となる。筆者がアクティブラーニングとして最終的に目指すのは、実習やフィールドワーク形式をさらに超えた、より構造の自由度が高く活動範囲の広いプロジェクト学習に相当するものである。

筆者はこれを目指し実践に取り組んできた。その構造は図５のような課外学習や地域との連携を強化したものである。

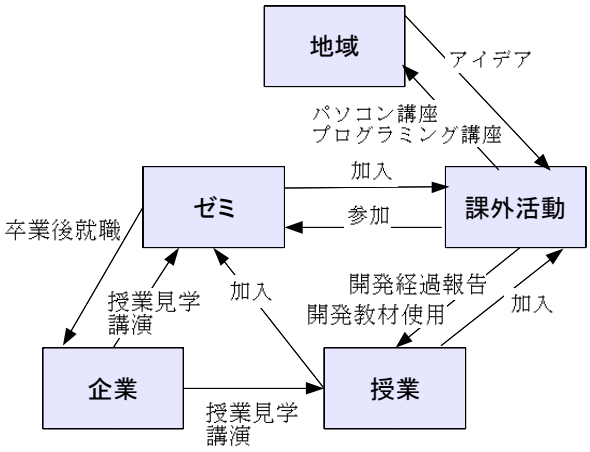

図５　授業を取り巻く学習環境とその相互作用(3)

授業を基本とするが、一般的には、そこでの学習はパソコン教室であるから、方法も限られてくる。先に述べた互恵的教授法や教室内でのグループ学習による効果を期待するにとどまる。

しかし社会的な実践共同体への参加の度合いを増すことが学習であるという正統的周辺参加の理論を適用すれば、プログラミング言語が社会でどのように用いられているのか、また学習者である学生自身がどのように役に立つことができるのかを経験させることによって、状況を認識したプログラミング言語学習ができると考えたのである(1)。授業と並行してこのような活動に参加している学生には、聞き取り調査や授業アンケートなどによって相応の効果があったとの確信を得ているが、授業のみに参加している学生にもその効果を波及させるべく、ゼミや課外活動で得られたノウハウを授業にフィードバックさせるように努めた。このような授業形態を多年度に渡り継続した結果、学生による課外活動への自発的な参加が増え、これまでビジネス系の専門科目とプログラミング言語学習との間に関連性を見出せなかったという学生の感想は減少している。

8. おわりに

ビジネス系学部でのプログラミング言語学習は、プログラミング言語が社会にどう役立っているのかという状況を把握させながら、ビジネス系の専門科目との関連を示しつつ進める必要がある。

しかし、筆者の経験ではこれを授業中に説明したとしても、その説得力はそれほど高くなく、単純なアルゴリズム学習の段階で挫折したり飽きられてしまったりすることが多かった。また、まじめに学ぼうとする学生でさえ、知識の積み上げ式であるプログラミング言語の習得は一度のつまずきが元で挫折してしまうことも多かった。

これらを改善させたのは、いわゆるアクティブラーニングであり、学習理論を取り入れた様々な工夫により授業は改善されている。そこで得られた知見は、次の２点である。（１）基本的なアルゴリズム学習では繰り返し学習をしながら、互恵的教授法や最近接発達領域の理論を取り入れ、相互に学び合うことで、中途の挫折者をできる限り防ぐことができるという者である。また、（２）ソフトウェアやシステム開発などを意識した応用的な言語学習では、認知的徒弟制や社会周辺参加の理論を取り入れ、より高度な構造と広範な活動を意図した社会との関わりを持った授業を展開することが有効であるということである。

今後は、より細かい学習環境、例えば教室設備、PC環境、UI、教材などとプログラミング言語学習との関わりについて研究を進めていき、ミクロ的な視点での学習効果の測定を行いたいと考えている。

参考文献

(1) Lave, J., Wenger, E.（佐伯 胖　訳）：“状況に埋め込まれた学習”，産業図書，(1993).
(2) 秋田喜代美：“学びの心理学　授業をデザインする”，142ページ，左右社（2012).
(3) 木下和也：“システム開発を意識したプログラミング教育の試み”，経営管理研究紀要，第16号, pp.11-20, 愛知学院大学経営管理研究所（2009)”
(4) 宮田仁，大隅紀和，林徳治：“プログラミングの指導方法と問題解決力育成との関連（３）”，日本教育情報学会第13回年会, pp.152-155 （1997）