2015 年度第 10 回物学研究会レポート

「コミュニケーション手段としてのプログラミングとデザイン」

清水　亮 氏

（株式会社 UEI 代表取締役社長兼 CEO）

2016 年 1 月 20 日

プログラミングとデザイン、日本語では一見、なんの関係もなさそうに思える 2 つの概念ですが、その根底は同じ所にあります。それはそのどちらも、言葉以外のものを使ったコミュニケーションであるということです。デザイナーと呼ばれる人にあまり馴染みがないと思われがちですが、実はプログラミングは「プログラムを通じたコミュニケーション環境のデザイン」と言えます。

そしてプログラミングとは、文字や音、映像はもちろんのこと、手触りや生理感覚といった、言語化も視覚化も不能な領域にまで神経を研ぎ澄ませて、緻密なデザインを行う手段でもあります。

本講演ではプログラミングとデザインの意外な共通点の紹介から、プログラマーから見たデザインのあり方についてお話しいただきました。

以下、サマリーです。

「コミュニケーション手段としてのプログラミングとデザイン」

清水　亮氏

(株式会社 UEI 代表取締役社長兼 CEO)

01：**清水　亮** 氏

坂井　清水さんには成蹊大学経済学部の大学生に対するプログラミング講座を担当してもらったのですが、ものすごく分かりやすく、プログラミングの魅力的な部分を話してくれました。デザイナーにとってエンジニアは相棒だと思いますし、プログラミングのことも知っておいたほうがいいと思い、今日の会を企画しました。では、清水さん、よろしくお願いします。

清水　こんばんは。清水と申します。今、ご紹介いただいたように、坂井先生の経済学部の授業をやらせていただき、プログラミングに縁のない人たちに教えた経験は僕自身にも大きな刺激になり、いろいろなことを考えるきっかけにもなりました。今日は、「プログラミングはなぜ必要なのか」、そして、「なぜデザインの文脈にプログラミングがでてくるのか」についてお話ししたいと思います。

■プログラミングはなぜ、必要なのか

まず、「プログラミングとは何か」と質問されたら、僕は、「プログラミングができるようになることは、新しい表現の道具を手に入れることだ」と答えています。

例えば、僕が社員に「こんなことはできないか？」「もっとこういうことをしてほしい」と指示するとき、絵やビデオ、アニメーションを使って説明しますが、どうしても説明しきれないことがあります。そんなとき、「清水さんが言っていることを理解するために、プログラムを書いてほしい」と言われることがよくあります。

具体的に言うと、時計のイラストをもとに製品化するとき、イラスト上の１本の線が部品なのか、表面処理の線なのかは図面に起こせば分かります。プログラミングにも図面に似た機能があって動きを説明しやすく、プログラマーと話すときにこれほど伝わる方法はありません。

もうひとつ例を挙げましょう。以前、「enchantMOON（エンチャントムーン)」というプログラミング端末をつくったときのことです。適当に手書きした丸をきれいな円に描きなおすという機能がありますが、実はこの機能はけっこうややこしいアルゴリズムを解かないといけないので簡単につくれません。

「どういう動きがほしいのか書いてほしい」と言われて、僕は最初、「P(t)=(P'-P)×0.1×t」という公式を書きました。でも、プログラマーは、「公式だけでは分からないから、実際に動くものをつくってほしい」というのでつくってみせたところ、やっと僕が表現したいことを伝えることができました。あとはプログラマーが別の言語で書き直して、この端末は完成しました。

■プログラミングと宇宙

次もプログラムでないと説明できないという例です。それは、「月はなぜ地球に落ちてこないのか」という質問に対する答えです。月が地球の周りを回っていることは、ほぼ誰でも知っていますが、この質問についてちゃんと説明できる人を僕はこれまで見たことがありません。

よく聞くのは、「月が落ちようとしたとき、地球はもうそこにないから」という説明です。でも、これは正しくありません。地球のほうが圧倒的に大きいし、地球自体も太陽の周りを回っているので、それほど自由に動けません。

また、「万有引力によって互いに引き合っているから」という説明もあります。これは、地球と月の間に働く引力を F、地球の質量を M、月の質量を m、地球と月の距離を r、万有引力定数を G とすると、F＝GMm/r^2 という式で表せます。でも、この式を使って普通に考えると、「月は地球に落ちてくる」という結論しか導きだせません。

高校物理では３つの方程式を習います。ひとつが先ほどの引力の式で、もうひとつは熱力学の方程式。そして、もっとも重要な運動方程式、すなわち、力ベクトル（F）は質量（m）と加速度ベクトル（a）の積で求められる F＝ma、それを変形すると、加速度（a）力（F）を質量（m）で割ったものになるという、a=F/m になります。

これらの式を使って考えると、地球のほうが重いので、実際に動く移動量は地球のほうがはるかに少ないことが分かります。つまり、同じ力をかけたときは重さが重いほどスピードは遅くなるので、地球のほうが動きは遅くなるわけです。

ところが、プログラムを書くとちゃんと説明できます。r の二乗で距離を求め、力ベクトル F を x と y に分解し、それぞれを月の質量（m）で割り、加速度（a）を足していくという単純な計算をします。この２つの方程式をプログラム言語に翻訳して実行してみると、ほとんどの天体は地球に衝突したり、どこかに飛んでいってしまったりして、地球の周りを回るようになる天体は中心にあるいくつかだけしかないと分かります。そのいくつかも、実際の月のようにあんなにきれいな軌道で回るものはありません。

つまり、月が地球の周りをきれいに回っているのは、ものすごく奇蹟的なことだと分かります。そういった説明をせずに、現実の宇宙空間のデータだけで答えを導き出そうとするから、「どうもこうらしい」と考えるしかない。だから、うまく答えられないし、プログラムを知らないと絶対に説明できません。

それに、このケースは地球が固定された「２体問題」なのでひとつの線形方程式で解けますが、太陽を含む「３体問題」になると普通の数学では解けなくなります。そして結局、分からないけど分かったことにしようとなってしまうのです

僕はシミュレーションを重視しています。ある仮説に対して、次に起こることはこういうことだとモデル化し、それぞれの関係性を仮説に基づいてプログラムで定義します。そして、実行してどうなるか推移をみることがシミュレーションです。

インフルエンザの流行やパンデミック発生の予測、災害の予測などにも使われます。シミュレーションは仮説を進めて検証することであり、プログラムのいい点は、そういう繰り返しによるシミュレーションをしやすいところです。

シミュレーションはまた、月と地球の関係や太陽系全体のこと、または、宇宙空間に物体を投入したら、ちゃんと静止軌道に乗るのか、人工衛星を打ち上げるときはどのくらいの初速なら周回軌道に乗るのかなど、いろいろな仮説の検証に使うことができます。宇宙計画はコンピュータがないとできないと言われるのは、そういう理由があるからです。

この５０年はコンピュータが普及した５０年です。この間、我々のライフスタイルにとってはインターネットや携帯電話ができたくらいで大きな変化はあまり思いつきませんが、科学の世界は確実に変わりました。それはプログラミングができるようになったことであり、おかげで人間の進化のスピードが速まったのは間違いありません。

■表現手段としてのプログラミング

先ほども言いましたが、言葉で説明しきれないこともプログラム言語で表現すると伝えられます。現在はデザイナーが表現したいものをプログラマーが翻訳しています。でも、プログラミングはそれほど難しくないので、デザイナーがプログラミングを学び、自分でやればいいのにと僕は思います。とはいっても、プログラミングだけを勉強してもうまく使えません。学ぼうとする目的が必要です。これは英語教育と同じだと僕は思っています。

最近、「社内の公用語を英語に」という会社が出てきていますが、僕は反対論者です。なぜかというと、たとえ TOEIC のスコアが高くてもその人が英語を使えるとは限らないからです。海外の会社と仕事をする場合、一定レベルの英語力は必要だと思いますが、それ以上は英語力よりもむしろ「何を話すか」のほうがもっと大事だからです。例えば、僕は本格的な英語教育を受けずに渡米し、マイクロソフト社で働き始めたころ、「誰よりもよく話している」と言われました。自分が表現したいことがあれば、英語が拙くても伝わるものです。

また、僕は高校受験のとき、先生に言われて 1 万語の英単語を覚えましたが、そんなに必要なのかと疑問に思っていました。そこで最近、僕は 20 年間書きつづけてきた自分のブログの文章、約 200 万字の中で使われている文字の種類を調べてみたんです。すると、数字、英語、記号、カタカナと、ひらがな、漢字、全部合わせて 3000 種類で、単語レベルになると 6000 単語しかありませんでした。

同様に著書でも試しました。1 冊あたり約 15 万字の本を 3 冊調べたら、1 冊分が約 4000 語から 5000 語でした。それくらいの単語の組み合わせだけで、自分の言いたいことがすべて表現できていたわけです。どんな表現手段をとったとしても、自分が表現したいものがあるか、表現したいものが何か分かっているかどうかが重要なのです。

表現手段という意味ではもうひとつ、プログラム言語と英語は類似性があると思っています。僕はアメリカから帰国したとき、「一番英語ができる」と評価されました。外資系の会社なので帰国子女や TOEIC900 点以上の人もたくさんいましたが、彼らは海外のプログラマーとは話ができなかったのです。プログラマーと話すのに必要なのは英語力でなく、数学やプログラミングができるかどうかです。つまり、英語と同様に、プログラム言語も世界共通語であり、世界中の人とコミュニケーションできる強力な言語なのです。

■デザインの文脈に、なぜプログラミングがでてくるのか

次に、プログラミングとデザインとの関わりについて話します。僕は、言葉を説明するときに単語から説明します。「design（デザイン）」は、「de」と「sign」に分けられます。

似たような言葉には、「define（定義する）」「description（説明する）」などがありますが、ラテン語系の言葉のつくりは共通しているので、「de」は「決める」という意味だと分かります。「sign」は「signature」など「しるし」「意図」といった意味があります。つまり、「design」とは「意図を決めるもの」だと分かります。

ここから分かるように、「design」という言葉を「ものの見方や形を決める」という意味に捉えているのはたぶん日本だけで、海外の英語圏ではそれに限らず、プログラミングもデザインだし、設計者やディレクターなども含んでいます。

先ほど僕は、英語公用語制には反対だと言いましたが、実は明治時代にも日本の国力を上げるために英語が必要なのではないかと、漢字を廃止してすべてローマ字にする英語公用語化の議論がありました。でも結局、現在のかたちに落ち着いたのは、「日本という国の文明のデザイン」であり、とても素晴らしいことだと僕は思っています。

例えば、英語をそのまま使うのではなく、新しい日本語がつくられました。philosophy（フィロソフィー）＝哲学、mathematics （マセマティクス）＝数学などは２語で意味を表現しています。それに数学のなかには algebra(アルゲブラ)、geometry（ジオメトリー）があり、それぞれ代数学、幾何学と訳されています。どちらも「学」がついているので学問だと分かりますが、英語そのものでは分かりにくいです。

また、「museum（ミュージアム）」も世界中で使われている言葉ですが、日本語では美術館と博物館の 2 つに訳されているので、日本語をみれば、その施設がどんなミュージアムか分かります。ルーブル美術館は、「きれいなものがたくさんあるだろう」、大英博物館は、「珍しいものを集めた場所なのだろう」と予測できるわけです。

英語から日本語になって文字数は減っているのに、情報は減っていません。こうした例は、明治時代の人たちが頑張ってつくってくれたタグであり、今、僕らがこうした言葉を享受できているのは非常にありがたい話だと思います。

■「マインクラフト＝Minecraft」の流行

「マインクラフト＝Minecraft」というゲームソフトは今日のテーマにぴったりかと思います。全世界で何千人もの人がプレイし、日本では小中学生にも流行っているゲームソフトです。ある世界を舞台に、そこにあるモノを叩いて破壊し、そこから新たなモノをつくりだして、サバイブしていくゲームです。例えば、壊した木を元に角材や板材をつくり、それからスコップといった道具や家をつくるのです。

これが驚くほど流行っています。実はゲームというより、プログラミングツールでもある点が人気なのです。プレイヤー自身が自分でプログラムを書き、マインクラフトというインタフェイス上で実行し、自分の表現手段として使っています。最近は、自宅にマインクラフトのサーバーを立ち上げるのが一種のステータスにもなっているようです。

すごいのはマイクロソフト社が買収したことです。同社は普通のゲーム会社には興味がないのですが、マインクラフトがただのゲームでなく、プログラミング環境だと考え、そこにコミュニティと可能性を見出したから買収したのだと思います。おそらくすごい高額で買っているはずです。

これと同じような例が以前にもあって、「スーパーマリオ」のステージをつくるゲーム、「マリオメーカー」で計算機をつくることが流行ったことがあります。マリオが近づいてくると勝手に弾を打つというアイテム、「キラー砲台」の性質を使い、自動的に足し算をさせるのです。この例で重要なのは、「モチベーションは何か」ということ。計算なら電卓ですればいいだけの話ですが、世の中には与えられた道具で、他の使い方をしようという人たちがいる点です。

最近の流れは明らかにこういう流れです。プログラミングが簡単になりプログラムができる人が増えたので、アイデア次第で思いついたものは何でもすぐにつくれるようになっています。実際に目に見えるものとしてのデザインと、論理回路や論理的な考え方のデザインの組み合わせはある程度一体化してきているのだと思います。

自分でプログラムを書けるようになることはまた、プログラマーとうまくコミュニケーションするための一番簡単で効率的な方法でもあります。プログラミングというとみんな身構えますが、実際はそうたいしたことではありません。簡単なプログラムを使って、どんな遊び方ができるだろうと考えればいいのです。

僕は優秀なプログラマーというのは、あまりプログラムを書かない人だと思っています。なるべく書かず、書いても 10 行程度の短いものです。なぜかというと、プログラムの美徳とは、同じことをやるのに短く表現できること。それが美しさにつながるからです。

プログラム言語をつくりだすことで重要なのは、自分がやりたいことや表現したいことは何か、表現したいことに対していかに最小限の努力でたどり着くかです。できるだけお手軽につくって目に見える結果をインスタントに求めようというのは最近の主流です。それって実は、いわゆる「デザイン」に近いことではないかなと思っています。

■言葉を覚えると、できることが増える

プログラミングの面白いところは、新しい言語をひとつ覚えると、できることがひとつ増えることです。例えば PHP を覚えるとウェブサーバーを書けるようになり、Javascript を覚えるとブラウザ上でゲームソフトが書けるようになり、C#を覚えると Xbox のゲームをつくれるようになる。つまり、簡単に自分の能力を拡張できるというわけです。

この「自己拡張感」こそ、コンピュータ研究の大きなテーマであり、「ヒューマン・エンハンスメント」と言います。現在はまだ複雑な状態ですが、将来的にはもっと簡単になって、自分のやりたいことができるシンプルな言語が見つかったり、より発展性があるだろうと思っています。

プログラミングはコンピュータとコミュニケーションするための重要な方法だと思います。人工知能技術もでてきていますが、まだ人間が言いたいことの本質やロジックを理解できるほどコンピュータが人間のことを理解できていないので、もう少し時間が必要でしょう。

また、プログラミングを通して世界中の人とコミュニケーションできると僕は思っています。プログラムは、ノンバ―バルなコミュニケーションであると同時に、ノンバーバルな論理表現の共有手段でもあるからです。

例えば、自動翻訳は自然言語のあいまいさを吸収させる必要があって非常に難しいですが、プログラミングの場合は論理構造を表現する言葉はひとつしかなく、最初から自然言語のあいまいさが排除されているので、英語でも中国語でも変換は簡単です。

だから、プログラミングで表現する世界が進んでいけば、言語や国境の壁を越えて、いろいろな人とコミュニケーションが取れるようになるのではないかと思います。

ここからは今日のまとめの部分になりますが、特に言いたかったことは 2 つです。ひとつは、言葉は表現手段のひとつでしかなく、プログラミングもまた、いろいろな可能性をもった新しい表現手段であること。

もうひとつは、プログラミングは表現の手段ですが、重要なのはそこにデザインする「サイン」をもっていなければいけないということ。たくさんのサインがある中でどれにフォーカスし、デファインするのか。僕は、それが「デザイン」なのだと思っています。ありがとうございました。

Q&A

Q1：関　清水さんは子ども時代からプログラミングされていたそうですが、プログラムに興味をもったきっかけは何でしょうか？　また、どんなところから勉強を始めたのでしょうか？

A：　僕が子どもだった当時、コンピュータはプログラムを書く以外に遊び方がなかったからです。ソフトウェアも高価だったので、ゲームも自分でつくるしかありませんでした。そして、プログラミングは専門書で学びました。昔は、自分が作りたいとイメージするプログラムをつくるのに、１年くらいかかりましたが、今は初心者でも１時間ほどの勉強で自分が表現したいものをつくれるようになり、環境が全く変わりましたね。

Q2： 今はハードウエアのデザインの世界では、ファブラボなど３Ｄコピーで気楽にものがつくられるようになり、モノづくりが民主化されてきたように、プロフェッショナルだったプログラミングの世界も、いわゆるDo it Yourself みたいな感じで楽しみ方が広がってきたということでしょうか？

A： その点に関しては、僕はもっと過激なことを考えていて、たぶんあと何十年かで、プログラムを書いてもお金をもらえる社会ではなくなります。プログラミングは特殊なスキルでなく、どんどん簡単になっているので、プロでなく普通の人が使いこなせる世界がくるだろうと思っています。

Q3： プログラミングで稼げなくなったら、清水さんはどうされますか？

A： さっきも言ったように、僕はいいプログラマーはプログラムを書かない人だと思っています。例えば、ビジネスや組織のプログラミングといったように、プログラミングの考え方は何にでも応用できるので、そういう人は生き残るように思います。ただし、数は多くないと思いますが。

Q4： 最近、ディープラーニングなどの可能性から人工知能ブームが再燃しています。このブーム、どう見られていますか？

A： 僕も子どものころから人工知能が好きで、プログラミングもしていましたが、なかなかものにならないという印象があり、また手書き認識や音声認識など限られた用途でなら少し使えるのかなという程度で、機械学習に対してはあまりいいイメージがありませんでした。でも、深層学習に関しては十分応用価値があり、非常に面白いなと思っています。昔は人工知能関連技術だったデータベースのように、ディープラーニングも今、そんな状況にあると思います。

Q5： 僕は８７年生まれの建築家です。建築の勉強はキャドから入ったので、手書きの図面では線をきれいに引けません。また、今の学生はスケッチアップのような３次元でモノをつくっているので、寸法に対して無感覚な気がします。そんなふうに、いつ生まれたのか、いつその世界に入ったかで、得る能力と失われる能力があると感じています。プログラミングの世界にもそうした能力はありますか？

A： 前提知識について世代間の差は大きいと思っています。例えば、1年ほど一緒に仕事をした子で、20代前半なのにプログラミング能力がとても高かったので、次に何を勉強したいか聞いたら、「アルゴリズムを勉強したい」と言うんです。アルゴリズムを知らずにプログラムを書いていたのかと驚きました。今は、言語環境がよくなっているので、知らなくてもできてしまうんですね。結果優先か、プロセス優先かの話なんでしょう。

また、寸法の感覚がなくなるという話はよく分かります。若いプログラマーにもメモリーやリソースについては緩い部分があって、「それじゃ、メモリーを無駄に食っちゃう」と僕らの世代は思ったりします。評価関数をどこに置くかですが、消費電力は絶対に無駄にしてはだめだと思います。

「僕たちとは違うな」と思うところもありますが、最初から３Ｄでやっているデザイナーには頭のなかに立体的なモノを完全にイメージできるわけで、それはそれでけっこう重要な感覚だろうと僕は思います。もちろん、手抜き建築や耐震設計をミスしたりするのは言語道断ですが、コンピュータの進化によって建築の仕方も変化しているということだと思います。

Q6：　「３０年後になくなる職業リスト」などがありますが、プログラマーの世界では、どんな人が生き残ると思いますか？

A：　僕は、プログラマーは誰も生き残らないと思っています。といっても、プログラマーがいなくなるのではなく、プログラミングの能力を誰もが持つようになると思うので、職業としてのプログラマーはいなくなるという意味です。

近年、ひとつの職業のライフタイムが短くなっています。特に先端的な仕事であればあるほど短くなっていて、人の一生のうちに何度もパラダイムシフトが起きている分野もあります。僕自身が一番体感したのは、家庭用ゲーム機ですね。１９８０年代にファミコンがでて、９２年にスーパーマリオ、９５年にプレイステーションが出ましたが、プレイステーションのブームは５年から１０年でした。

だから、「プログラミングというスキルをもって生きていく」というのはありですが、「プログラマーとして一生やっていく」という発想はもう古い。つまり、特定の職業を一生つづけていくということはもうないんじゃないかなと思っています。

以上

2015 年度第 10 回物学研究会レポート

「コミュニケーション手段としてのプログラミングとデザイン」

清水　亮 氏

（株式会社 UEI 代表取締役社長兼 CEO）

写真・図版提供

01；物学研究会

編集=物学研究会事務局
文責=関 康子