ものづくりのための「ものグラミング」と実践的教育環境の構築

大野 浩之[1]　森 祥寛[1]　北口 善明[1]　中村 和敬[2]　松浦 智之[2]　石山 雅三[2]　當仲 寛哲[2]

概要：近年，自らの手で何かを作る「ものづくり」を愛好する者は増加を続け，その風潮は 「メイカー・ムーブメント」とも称されている．ものづくり愛好者のものづくりの対象・手法は多岐に及ぶが，ものづくりに際してマイコン（マイクロコントローラ）を使う場合，これまでの主流だった PIC マイコンや Arduino に加えて Raspberry Pi に代表される UNIX 系 OS を搭載した高機能マイコンを採用する事例が増えている．しかし，高機能マイコンを採用した多くの事例で，これまでのマイコンよりはるかに高機能であることを十分に生かし切れていない．そこで，マルチユーザ・マルチタスクの高機能 OS である UNIX 系 OS が提供する優れた基本機能と，多種多様な POSIX コマンド群を積極的に利用してものづくりを支援する手法を提唱し，これを「ものづくりのためのプログラミング」という意味を込めて「ものグラミング」と名付けた．本論文では「ものグラミング」の考え方，ものグラミングのためのものづくり環境，ものグラミングに基づく教育環境について述べ，その有効性を示す．

The Mono-Gramming Approach for MAKERs and its Educational Environment

Hiroyuki OHNO[1]　Yoshihiro MORI[1]　Yoshiaki KITAGUCHI[1]　Kazutaka NAKAMURA[2]
Tomoyuki MATSUURA[2]　Masami ISHIYAMA[2]　Nobuaki TOUNAKA[2]

1. はじめに

近年，マイコン（マイクロコントローラ）を使った計測制御系を組み込んだ機器を個人が気軽に設計製作する環境が整ってきている．個人向けの 3D プリンタやレーザーカッターの普及が始まったのと時期を同じくしたこともあり，電子回路を活用した「ものづくり」を趣味としてあるいは小規模事業として行う者は増加を続けており，彼らの製作物も多様化している．このような，ものづくりの当事者を Maker と呼ぶが [*1]，Maker による新たな「ものづくり」の潮流は新たな文化であるとして，クリス・アンダーソンによって「メイカー・ムーブメント (Maker Movement)」と名付けられた [1].

Maker が，作品に利用するマイコンは多種多様であるが，すでに普及し広く使われていた PIC マイコンや，AVR マイコンを搭載した Arduino などに加え，2012 年頃からは Raspberry Pi[2] に代表される「Linux を搭載する小型で高機能なマイコン」（以下 Linux マイコン）が安価に供給されるようになったため，Linux マイコンを積極的に採用する傾向がみられる．しかし，現状ではマルチユーザ・マルチタスクの本格的な Linux OS が利用になったことを十分に生かし切れておらず，「OS が提供する多様な基本機能」や「OS インストール時点で多数の基本コマンド群が利用可能」といった特徴を積極的に活用すれば簡単に目的を達成できる局面であっても，旧来の手法，すなわち C や C++ などの言語を用いたプログラムですべてを記述してコンパイルしたり，Ruby や Python のようなスクリプト言語を追加インストールした上でこれらの動作環境を常に最新状態に更新しつつ利用したりする手法が未だに主流である．

これに対し著者らは，「UNIX 系 OS が提供する優れた基

[1] 金沢大学総合メディア基盤センター
Information Media Center, Kanazawa University
[2] ユニバーサル・シェル・プログラミング研究所
Universal Shell Programming Laboratory Ltd.

[*1] Maker を「メーカー」と表記すると，白物家電製造業者をさす「(家電) メーカー」と同じ表記になるため「メイカー」と記して区別する事例もあるが，本報告では Maker と表記する．

本機能」と「UNIX 哲学 [3]」に基づいて作られた，「それぞれは単機能だが組み合わせることでさまざまな機能を提供できる多種多様な POSIX コマンド群」を積極的に活用したシェルスクリプトでものづくりを推進することにし，この手法を「ものづくりに適したシンプルなプログラミング」という意味を込めて「ものグラミング」と名付けた．英語では mono-gramming と表記することにしたが，これは日本語の「ものグラミング」の音訳であるとともに mono という音節に「単純かつ単一なプログラミングスタイル」というニュアンスを含ませている．

2. ものグラミング

2003 年の “Wiring” から発展を続け [4]，ものづくりの現場において現在でも数多く用いられている Arduino[5] は，Arduino UNO のような主要機種におけるプログラム格納領域は 32kB に過ぎず，購入時点ではその先頭部分にブートローダが書き込まれているだけで OS と呼べるものは存在しない．数 100 種類の多種多様なライブラリがオープンソースで提供されてはいるものの，C++言語をベースにしたパソコン上のクロスコンパイル環境（Arduino IDE）で必要とする機能の全てをユーザが記述してコンパイルし，これを実機に書き込むのが基本である．ブートローダが一定の領域を占めるため，ユーザが記述したスケッチ（Arduino ではユーザが書くプログラムを「スケッチ」と呼ぶ）をクロスコンパイルしたバイナリを格納可能な領域はさらに小さく約 30KB となる．そのため，たとえば DHCP クライアント機能と DNS による名前解決機能を持つスケッチを書くと，それだけでプログラム格納領域をほぼ使い切ってしまい肝心の機能をほとんど記述できない状態になる．したがって，ネットワークアクセスを伴う実用的なプログラムを書くのは容易ではない．ヒープとスタックに用いられる RAM も Arduino では 2kB である場合が多い．このサイズでは，仮に IPv6 パケットを扱うスケッチを書くと，変数に IPv6 アドレスを 8 つ取り込んだだけで RAM 領域の半分を消費してしまう．すなわち，インターネットに接続する用途に供するには RAM 領域も全く不足している．このような状況から，Arduino によるものづくりが普及しても，その作品がインターネット接続性を直接持つことはあまりなかった．

一方，2012 年に登場した，UNIX 系 OS が動く Raspberry Pi では，0.5TB ないし 1GB のメインメモリが用意されており，ファイルシステムも SD カードや USB ストレージ上に作るため，個々のプログラムの大きさについては制約を受けることは事実上なく，さらに UNIX 系 OS が提供する基本機能や多種多様な POSIX コマンド群が，OS インストール直後から利用できるという利点がある．インターネットへの接続性確保も問題なく実現できる．このような Raspberry Pi 上でものづくりをする場合，ものグラミングのための汎用的なコマンドを新たにいくつか追加する必要はあるものの，少数の新規追加コマンドと既存の POSIX コマンド群を効率よく連携させるシェルスクリプトを書けば，C 言語等を駆使して作品を手がけるたびにいちいち最初から全てをプログラミングせずとも，ものづくりに必要なプログラムは簡単にできると考えた．すなわち，著者らが提唱する「ものグラミング」では，OS の基本機能や既存のコマンドを最大限に生かしてシェルスクリプトを書くことがほぼ全てであり，コンパイルを伴う言語で個別の作品ごとにプログラムをゼロから開発する作業は原則として行わない．

ここで，「LED を点滅させる」という基本動作をどう実現するかを従来の方法と比較すると以下のようになる．なお，この動作は「LED をチカチカと点滅させる」ということから「L チカ」と呼ばれ，プログラミング言語の特徴を述べる際に，まず「”Hello, World!” を画面に表示させるプログラム」を記述してこれを他の言語と比較検討することから開始するのに相当する．

まず，Arduino で L チカを記述すると以下のようになる．このコードでは，Arduino のデジタル入出力ピンの一つ（13 番ピン）を setup() 関数で出力モードに初期化し，loop() 関数で 1 秒毎に HIGH と LOW を交互に書き込んでいる [*2]．多くの Arduino では 13 番ピンに LED が取付けられているので，このスケッチで LED の点滅が実現する．

```
// 初期化
void setup() {
  pinMode(13, OUTPUT);
}
// 点滅
void loop() {
  digitalWrite(13, HIGH); // LED 点灯
  delay(1000);            // 1 秒待つ
  digitalWrite(13, LOW);  // LED 消
  delay(1000);            // 1 秒待つ
}
```

Arduino 言語 [*3] の規定により，setup() と loop() の名前は特定の目的に使われることになっており，前者はプログラム実行開始時に一度だけ呼び出され主として初期化を行い，後者は setup() の後に自動挿入される無限ループ内で繰り返し呼び出される．なお，main() は自動生成され

[*2] より実用的なプログラミングでは，ピン番号は直接記述せずに変数に格納するかクラス内に閉じ込めるか名前で記述してプリプロセッサに処理させるかして抽象化し，ピン番号を必要とする関数等では，この例のように数値を直接指定するのではなく，抽象化した名前で参照するように記述すべきであるが，この例ではそうなってはいない．その是非についてはここでは議論しない

[*3] Arduino IDE で記述する言語には正式名称はないようだが，C++ との違いを明確にする必要がある場合には Arduino 言語と呼ぶことがある

るため記述する必要はない．Arduino IDE 上でこのコードを記述した後，クロスコンパイルし，さらにターゲットとなる Arduino に書き込んで動作させる．一連の動作は，Arduino IDE 上のボタンを一度クリックするだけでつなぎ目なく行え，この点が Arduino をそれまでのマイコン開発から一歩抜き出た存在にしたが，クロスコンパイルしてからターゲットに書き込むという流れは，従来のマイコンで行われて来た方式を踏襲している．

次に，同等の内容を実行するシェルスクリプトを Raspberry Pi 上で作成して実行すると以下のようになる．なお，この例では，Raspberry Pi の GPIO13 に 1 と 0 を交互に書き込んでいるが Arduino と違って LED は装着されてはいないので，LED を点滅させるためには GPIO13 に電流制限抵抗を直列接続した LED を装着する必要がある．

```
#! /bin/sh
# 初期化：
gpio -g mode 13 out
#点滅：
while [ 1 ]; do
 gpio -g write 13 1; sleep 1;
 gpio -g write 13 0; sleep 1;
done
```

Raspberry Pi は，Linux や BSD 等の UNIX 系 OS が動くので，新たなプログラムを C や C++などの高級言語で記述してセルフコンパイルすれば，得られたバイナリはそのまま実行できるが，上記のコードはシェルスクリプトなので，コンパイルの必要もない．さらにこの程度であれば，わざわざシェルスクリプトにしなくても Raspberry Pi のシェル上で 1 行のコマンドライン（ワンライナー）として直接実行できる．たとえば，上記のシェルスクリプトは以下のようにして実行できる．

```
$ gpio -g mode 13 out; while [ 1 ]; do gpio
-g write 13 1; sleep 1; gpio -g write 13 0;
sleep 1; done
```

なお，上記のシェルスクリプトやワンライナーで用いている gpio コマンドは，Raspberry Pi のために書かれた WiringPi ライブラリ [6] のパッケージに含まれるコマンドである．POSIX では規定されていないため OS インストール直後には存在しきないが，git コマンドでソースコードをダウンロードしてビルドするだけの簡単な操作でインストールできる．

シェルスクリプトを活用するものグラミングでは，上述のような単純動作に限らず，ネットワーク上のリソースと連携した動作を行う際にも，ほとんどの場合は新たなプログラミングを行う必要はなく，既存のコマンドを呼び出すことで目的を達成できる．これは大きな利点があるが，上記の例で言えばデジタル入出力を行うたびに gpio コマンドを一つ実行するため，利用目的によっては実行速度が問題となる場合がある，この点については考察で述べるが，秒単位の間欠的な計測や制御で済む場合であれば，シェルスクリプトであることは問題にはならない．

3. ものグラミング実践環境

ものグラミング手法の有効性を確認するため，著者らは以下の環境を準備して自らで使いつつ，情報科学を専門としない大学学部生向けの授業にこの環境を導入し，ものグラミングの有効性を確認する評価を行っている．

3.1 ハードウェア：Raspberry Pi

ものグラミングで利用するマイコンは UNIX 系 OS が利用でき，汎用デジタル I/O (GPIO) ポートがユーザ空間のアプリケーションから利用できることが必須となる．

この条件を満たすマイコンボードは，Raspberry Pi 以外にも Beagle Bone シリーズ，Intel Edison を始めとして数多く存在し，それぞれに特徴があるが，今回は Raspberry Pi を選んだ．選択の理由は 1 台 35 ドルという低い価格とその結果もたらされた普及度合いである．したがって，他の同様のマイコンボードでも，ものグラミングは可能であり，今後 Raspberry Pi 以外のマイコンでも実施する可能性はあるが，以下では特記ない場合は Raspberry Pi を前提としている．

Raspberry Pi のハードウェア構成の詳細についてはインターネット上に広く公表されているのでここでは割愛するが，USB ポート，HDMI ポート，アナログ AV 入出力，有線 LAN 接続以外の周辺機器との接続は，P1 ヘッダと呼ばれるピンヘッダに集中している．gpio コマンドはこのピンヘッダの状態をユーザが操作するためのコマンドであると言える．

Raspberry Pi を利用する場合，著者らは，キーボード，マウス，ディスプレイをつながず，LAN ケーブルでパソコンと直接あるいはスイッチングハブ経由で接続し，SSH でアクセスする方式を主たる利用方式としている．学生向け講義でも基本はこの方式である．ただし，必要に応じて，キーボード，マウス，ディスプレイを接続して利用したり，シリアルポートにパソコン等を接続し，ターミナルエミュレータを介して操作する方法も適宜併用している．

3.2 ソフトウェア：UNIX 系 OS と POSIX コマンド群とシェルスクリプト

ハードウェアが UNIX 系 OS を利用できる能力を持つことが前提なので，ソフトウェアは，(1) UNIX 系 OS, (2) OS 上で動くコマンド，アプリケーションおよびライブラリ，さらに (3) これらを活用するユーザ作成のプログラム

という3つのカテゴリからなる一般的な構成になる．

まずOSにはDebian LinuxのRaspberry Pi向けディストリビューションであるRaspbian[7]を採用した．Raspbianの採用により，POSIXで規定される全てのコマンドはRaspbianのインストール直後から利用可能になる．

Raspbianは，Debian LinuxのRaspberry Pi向けディストリビューションで，2016年5月の時点ではDebian 8.0 Jessieに対応している．このため，ソフトウェア管理システムのapt-getコマンドによってapt-getが管理するソフトウェアは容易に導入できる．さらに，gitコマンドによって，オープンソースソフトウェアとして公開されたプログラムやライブラリも容易に導入できる．このような環境の上でユーザは必要なプログラミングを行うことになり，C, C++などのコンパイラ言語，Ruby, Pythonなどのスクリプト言語，さらにコマンドの組み合わせを記述するシェルスクリプトなどが利用可能になる．

前述のように，ものグラミングにおいては，コンパイルを伴う高級言語でのプログラミングだけでなく，Perl, Ruby, Pythonのようなスクリプト言語も使わず，POSIXに規定されるコマンドに最小限のコマンドを追加した環境で，もっぱらシェルスクリプトを活用するソフトウエア開発体制を整えている[*4]．この方針は，著者らが別途研究を進めている「POSIX中心主義」[8]に準拠したためであるが，gpioコマンドやi2cset/i2cget/i2cdetectコマンド（後述）の「交換可能性担保」が確保できていない．したがって，現時点では，ものグラミングはPOSIX中心主義から逸脱している部分があるが，現時点で交換可能性担保が確保できていないこれらのコマンドはオープンソースであり，それぞれが提供する機能はUNIX哲学に基づいたシンプルなものであるので，交換可能性担保はいつでも確保できる．交換可能性担保は，当該環境を10年あるいは20年間先でも利用可能にするために必要な条件であるが，今は将来よりも現在のものグラミング環境を発展させることを優先する時期なので，交換可能性担保の確保の優先度は下げている．

3.3 周辺回路：I2Cバスの活用

Raspberry Piと周辺回路を相互接続する際に利用するP1ヘッダは当初は26ピンであったがRaspberry Pi B+から40ピンになった．電源とGNDを除くピンには複数の役目を切り替えて割り当てることができるが，標準提供の機能と数さらに消費ピン数は以下のようになる．

Raspberry Pi B+ 以前
- UART × 1 (2ピン)
- SPI × 1 (5ピン)
- I2C × 1 (2ピン)
- Digital I/O × 8 (8ピン)

Raspberry Pi B+ 以降
- UART × 1 (2ピン)
- SPI × 2 (9ピン)
- I2C × 2 (4ピン)
- Digital I/O × 13 (13ピン)

このように，Raspberry PiはA/D変換器やD/A変換器を持たず，GPIOポートにも数の限りがある．この制限を解除するためにはUART, SPI, I2Cなどを活用することになるが，接続するデバイスの種類の豊富さや価格をから，I2Cバスに着目してこれをシェルスクリプトから利用する環境を構築した．このために，まずI2Cバスに対して情報を送受信するi2cset, i2cgetコマンド等を整備した．さらに，Raspberry PiのI2Cバスを絶縁した上で延長し，センサやI/Oポートを小型のケース（実体はペットボトルキャップ）に納め，I2Cモジュールを簡単に着脱するしくみ（I2Cブロック）を新規に開発し，利便性を大きく改善させた．

3.3.1 I2Cバスを操作するコマンドの追加

Raspberry PiからI2Cバスを操作するためにまず以下のコマンドを追加した．

- i2cset - 指定したアドレスを持つI2Cデバイスに指定した値を書き出す．
- i2cget - 指定したアドレスを持つI2Cデバイスからデータを読み出す．
- i2cdetect - 利用可能なI2Cデバイスのアドレスを表示する．

これらのコマンドは，現時点では交換可能性を担保する代替コマンドがないので，これらのコマンドを使うことはPOSIX中心主義準拠からは逸脱するが，本研究はPOSIX中心主義遵守が目的ではなく，単純な機能を提供するこれらのコマンドの追加で シェルスクリプトからI2Cデバイスへのアクセスが可能になるため，ものグラミングでは全面的に採用している．

3.3.2 I2Cバスの操作

デジタルI/Oが可能なI2Cデバイスを操作してLEDの点滅を行うには以下の操作が必要となる．この例では，Microchip社のI2C I/Oエキスパンダ MCP23017の8ビットのI/Oポートの全てにLEDを接続し，単なるLEDの点滅ではなく，8つ並んだLEDの一方の端から他の端に向かって順次点滅してゆく動作を繰り返すようにした[*5]．

I2Cインタフェースに対応した素子は一般に多機能なため特定の機能を用いるための初期化は，前述のgpioコマンドによる操作例より多少複雑になるが，実際の点滅操作全体としては大きな違いはない．

[*4] Perl, Ruby, Pythonは POSIXに規定が無いため利用しないが，awkと sedは規定があるので積極的に利用可能

[*5] Raspbianの sleepコマンドは1秒単位ではなくこの例のように1秒に満たない時間を指定できるが，この機能はPOSIX準拠ではない

```
#! /bin/sh
I2C_IODIRA=0x00
I2C_OLATA=0x14

i2cset -y 1 $I2C_DEV $I2C_IODIRA 0x00
export xcnt=1
while [ 1 ]; do
  i2cset -y 1 $I2C_DEV $I2C_OLATA $xcnt;
  sleep 0.1;
  i2cset -y 1 $I2C_DEV $I2C_OLATA 0;
  xcnt=‘expr $xcnt ’*’ 2‘;
  [ $xcnt -ge 256 ] && export xcnt=1;
done
```

4. 普及啓発活動

4.1 著者らの周辺での試験的運用

前節で述べたシステムを用意し，システムを開発した著者の一人（大野）が本共同研究参加者全員に実機を示しながら解説し，その有効性を説明した．これをもとに議論を行い，次項で述べる大学での教育で実践することが決まった [9].

4.2 大学教育での実践

これまでに，以下の講義で，ものグラミングのアプローチを取り入れた講義を行った．

- 計算科学特論（金沢大学，2015 年度後期）
- クラウド時代の「ものグラミング」概論（大学コンソーシアム石川いしかわシティカレッジ, 2016 年度前期）

前者の講義では，試行錯誤の要素が強く，gpio コマンドを用いた操作が中心で I2C については言及できなかった．

後者は現在開講中の講義であり，本稿執筆時点では Raspberry Pi の利用には到達しておらず，ESP-WROOM-02 を用いた従来型の電子工作を行っている．この後，2016 年 6 月中旬からいよいよ Raspberry Pi を用い，ものグラミングの講義と演習に入る．

どちらの講義でも，手元のマイコンの動作と連携させる WEB 側のサービスの最初の事例として，コマンドライン操作が可能な Twitter クライアント「kotoriotoko（小鳥男）」を用いた．kotoriotoko はすべて POSIX 中心主義に合致するシェルスクリプトである．

4.3 電子工作愛好者へ向けた活動への展開

著者の一人は石川県金沢市周辺の電子工作愛好者の集まり「木いちごの会」を主催し，2014 年 6 月から月に 2 度のペースで技術的な話題を交換したり新しい技術のハンズオンを行う集い（木いちごの夜会）を開催してきた．ものグラミングのアプローチは，この会合でも紹介している．利用者が急増したという状況にはないが，今後理解者が増えることを期待している．

また，Maker 向けの作品公開・展示の場としては，日本では Maker Faire Tokyo,　NT 金沢，NT 京都などが開催されている．著者らはこのような場でも，ものグラミング方式を取り入れた作品を展示しており，今後も展開する予定である．

5. 考察

5.1 本方式の性能

本方式に対する予想される否定的見解の筆頭は，その処理速度についてである．

GPIO の処理速度比較については，WEB 上に [10] に興味深い比較結果が掲載されている．これによれば，gpio コマンドを用いたデジタル出力（遅い）と，カーネル空間にある GPIO ポートをユーザ空間から直接操作するデジタル出力（最速の方法）とでは 処理速度に 50 万倍近い差が見られた．すなわち，GPIO ポートを単にオン・オフした場合，前者では 40Hz 程度なのに対して，後者では 20MHz を越えている．したがって大量の入出力を高速で行う必要があるのであれば，ものグラミングが提唱する「gpio 等のコマンドと既存のコマンドを活用したシェルスクリプトを活用する方式」は不向きである．しかし，計測制御する間隔が秒の単位であるならこの速度差は問題にならない．現在，このような動作が比較的遅い用途をを中心にものグラミングの適用を始めている．

5.2 POSIX に準拠していない UNIX 系 OS の是非

Linux を搭載したマイコンボードの中には，当該 OS のもとになる版の OS からいくつかの機能を抜いて提供している（この OS をここでは「サブセット版」と呼ぶ）場合がある．サブセット版は OS のサイズを小さくし，動作速度もファイルシステムや使用メモリのサイズも「軽量」にするのが目的である．多くのサブセット版では，busybox 等を用い，/bin や /usr/bin の下にあるコマンドの多くは buspox へのシンボリックリンクになっており，busybox は自分がどの名前で呼ばれたかを検出して，呼ばれた名前に対応した機能を提供する．たくさんリンボリックリンクが貼られても実体は一つであるため，ファイルシステム上で占める割合も実行速度も節約できる．しかし，busybox で提供されるコマンドの多くは POSIX に準拠しきれていないため，POSIX 準拠を前提に作られたシェルスクリプトが正しく動作動作しないことがある．そのため，当面は，POSIX に準拠した OS については，ものグラミングの適用対象外とする．

5.3 IFTTT との比較

IFTTT[11] は，さまざまな既存の WEB サービス（facebook や twitter　など）を連携させる仕組みで，”if this then that” の頭文字を取って命名されている．IFTTT の利用者は，ifttt.com に用意された WEB インタフェースを用い，“this” と “that” の部分（前者をトリガー，後者をアクションと呼ぶ）に既存の WEB サービスを指定して連携を実現する．すでに，Raspberry Pi やネットワークインタフェースを追加した Arduino をトリガーやアクションに指定する機能（HTTP を用いたフック）が提供されているので，これを利用してマイコンと WEB サービスを連携させている利用者は多い．

一方，ものグラミングではシェルスクリプトを使うので，特定のサイト（この文脈では ifttt.com）に依存せず，以下のように初期化と実行を以下のように 1 行ずつで記述できる．この例では，(1) GPIO4 を入力モードにした上で内部抵抗でプルアップする初期化を行い，その後は (2) GPIO4 を監視し，(3) GPIO4 が HIGH から LOW に落ちるまで待ち続け，(4) HIGH から LOW に落ちたら指定したコマンド（この例では date コマンド）を実行し，(5) 0.1 秒待ってから (2) に戻るという動作をする．date コマンドの部分をメールを送出したりツイートするといったコマンド（あるいはシェルスクリプト）に置き換えれば IFTTT 以上に多彩な対応が自力で実現できる．

```
# 初期化（GPIO4 を入力モードにし，内部抵抗で
# プルアップ）
$ gpio -g mode 4 input; gpio -g mode 4 up

# GPIO4 を監視し，LOW に落ちたらコマンドを実行
# する動作を繰り返す
$ while [ 1 ]; do gpio -g wfi 4 falling &&
date; sleep 0.1; done
```

5.4 本方式の教育効果

POSIX コマンドと少数の新規コマンドを組み合わせて，ものづくりが行う方法が理解できれば，C や C++のようなコンパイルを必要とする高級言語や，軽量（Lightweight）と言っても，ものづくりには必要のない機能も盛りだくさんの ruby や python のような軽量スクリプト言語を学ぶよりものづくりを行いやすくなると予想している．

このことを客観的に確認するためには，同程度の経験を有する被験者を 3 組に分け，高級言語を用いたプログラミングを使ったものづくり，軽量スクリプト言語を使ったものづくり，ものグラミングスタイルのものづくりを実施し，技術習得の様子を観察する必要がある．ものグラミング教育はまだ試行段階なので，現時点ではこのような定量的な評価ができる状況にはないが，前述のように一連の準備を経て，2015 年 10 月から金沢大学において「ものグラミング」を学部生向け講義に試験的に取り入れ，さらに 2016 年 4 月からは「いしかわシティカレッジ」の講義での全面採用し，さらに石川県金沢市周辺の電子工作愛好家との情報交換会でも普及啓発を行っているので，およその定性的な傾向は今年度中に把握できることを期待している．

6. おわりに

著者らは，UNIX 系 OS が利用できるマイコンボードで，POSIX コマンド群，少数の新規作成コマンド，シェルスクリプトを駆使してものづくりを支援する環境について検討し，実際に講義等で利用してその有効性を確認する作業を続けている．引き続き I2C バスを活用する実験環境の拡充と，利用者がただちに利用できるスタイルに調整したシェルスクリプトの提供をさまざまな I2C デバイスに対して行い，使い勝手を向上する作業も継続する方針である．

謝辞

本研究は，金沢大学総合メディア基盤センターとユニバーサル・シェル・プログラミング研究所の共同研究の一環として現在も推進中である．本共同研究の円滑な遂行を支援していただいている両組織の関係者各位に感謝する．

参考文献

[1] Chris Anderson : ”Makers : the new industrial revolution”,”Random House Business Books”,2012

[2] Raspberry Pi : ”Teach, Learn, and Make with Raspberry Pi”, 入手先 ⟨https://www.raspberrypi.org/⟩ （参照 2016-05-20）.

[3] Mike Gancarz, 芳尾桂 (監訳) : ”UNIX という考え方 (The UNIX Philosophy)”, (オーム社, 2001 年)

[4] Hernando Barragn : ”The Untold History of Arduino”, 入手先 ⟨https://arduinohistory.github.io/⟩ （参照 2016-05-15）.

[5] Arduino : ”WHAT IS ARDUINO ?”, 入手先 ⟨http://www.arduino.org/⟩, （参照 2016-05-15）.

[6] Gordon Henderson : ”Wiring Pi”, 入手先 ⟨http://http://wiringpi.com/⟩ （参照 2016-05-15）.

[7] Raspberry Foundation: ”RASPBIAN”, 入手先 ⟨https://www.raspberrypi.org/downloads/raspbian/⟩, （参照 2016-05-15）.

[8] 松浦智之, 大野浩之, 當仲寛哲 : ”ソフトウェアの高い互換性と長い持続性を目指す POSIX 中心主義プログラミング”, 情報処理学会 DICOMO2016 予稿集 (掲載予定), 2016

[9] 中村 和敬, 大野 浩之 他 : ‘’IoT 時代に資する「ものグラミング」教育のための授業開発と実践’’, 教育システム情報学会 2015 年度第 6 回研究会, 2016.

[10] Joonas Pihlajamaa : ”Benchmarking Raspberry Pi GPIO Speed”, 入手先 ⟨http://codeandlife.com/2012/07/03/benchmarking-raspberry-pi-gpio-speed/⟩ （参照 2016-05-15）.

[11] IFTTT : ”Connect the apps you love.”, 入手先 ⟨https://ifttt.com/⟩ （参照 2016-05-15）.