

3

ヒューマンコンピュータ インタラクションとしてのプログラミング

HCI におけるプログラミング研究の過去、現在、そして未来



+ + + + + 鈴木 遼 (University of Colorado Boulder)

プログラミング研究の歴史

本稿では、ヒューマンコンピュータインタラクション (Human-Computer Interaction, 以下 HCI) における、プログラミング研究の過去、現在、そして未来について、述べる。まずは、コンピュータにおいては、プログラミングの歴史を紐解きながら、なぜ HCI の分野でプログラミングの研究が行われてきたのか、どのように HCI という分野がプログラミング研究に貢献してきたのか、という点から議論していきたい。

インタラクティブなプログラミング

1950 年代、コンピュータは現在のようなインタラクティブなツールではなく、一方向的な計算機であった。そこでのプログラミングとは「パンチカードと機械語を使い、計算機に命令を与える作業」としてみなされていた。しかし、1960 年代に入り、Doug Engelbart や Alan Kay などの先駆的な研究者によって、コンピュータは単なる計算機から、インタラクティブな思考・表現のツールに変わっていった。時を同じくして、プログラミングも、「一方向的に計算機に命令を与える作業」から、「コンピュータと対話をしながらインタラクティブにコンピュータの振舞いを記述する作業」へと変化してきた。この新たなプログラミングパラダイムは、Interlisp のような対話型開発ツールの研究や、BASIC のような、ソースコードエディタ、コンパイラ、デバッガがひとまとまりで提供される「統合開発環境」という概念を生み出し、そこで生まれたプログラミングの在り方や思想は、Eclipse などを通じて現在にも引き継がれている。

エンドユーザプログラミングと ビジュアルプログラミング言語

一方、インタラクティブなコンピュータや開発環境の登場により、プログラミングの敷居が下がり、「エンドユーザプログラミング」という新たなプログラミングスタイルが生まれた。エンドユーザプログラミングとは、プロフェッショナルな職業プログラマとは対照的に、プログラミングの初学者や知識を持たないユーザが、自らのニーズを満たすために、簡単なスクリプトなどを使ってプログラムすることを指す。たとえば、エクセルを使った複雑なデータの操作や、R を使った統計の計算、HTML や JavaScript を使った Web ページの作成などが挙げられる。こうしたエンドユーザプログラマは、アメリカにおいて約 9,000 万人以上いると言われており¹⁾、どのようにこうした初学者をサポートするかが大きな研究課題であった。1つのアプローチとして、ビジュアルプログラミング言語と呼ばれる視覚的な操作を使ったプログラミングが提案された。しかし、ビジュアルプログラミング言語は、Scratch や Matlab, Grasshopper などに少数の成功例が見られるが、多くのプログラミング言語を置き換えるほどには普及せず、現在に至っている。それはなぜか。

ヒューマンコンピュータインタラクション としてのプログラミングへ

❖ 人間中心のアプローチへ

ビジュアルプログラミング言語に対する 1つの批判として、人間がどのようにプログラミングをしているか、ということを深く理解せずに、単純に

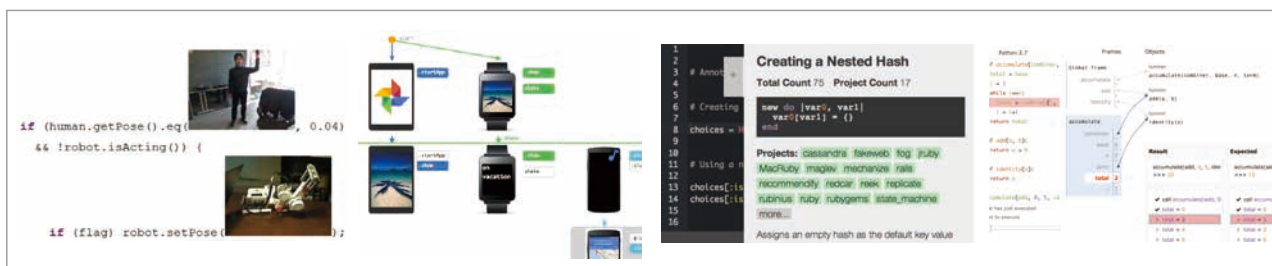


図-1 HCIにおけるプログラミング研究の例：左から、プログラム内に画像を埋め込めるPicode、ストーリーボード形式でアプリをデザインができるDemoScript、クラウドソーシングを活用してコードの推薦を行うCodeX、間違っているコードが本来どう動くべきか可視化するTraceDiff（画像は著者の許可を得て掲載）。

「視覚的 = 分かりやすい」という仮説に則ったシステムが多く提案されてしまった、という点が指摘されている^{2), 3)}。こうした批判から、一度原点に立ち返り、プログラマはどのような問題に直面しているのか、その問題を解くためにはどのようなプログラミング言語やツールを提供するべきなのか、という観点から再度プログラミングを考え直す「人間中心（Human-Centered）」のアプローチが1990年代から2000年代にかけてHCIの新たな潮流になった。その中心となったのが、カーネギーメロン大学のBrad Myersとその教え子であるワシントン大学のAndrew Koである。

彼らは、人間中心のアプローチをもとに、HCIの手法や考え方を応用したプログラミング研究を進めた。すなわち、「プログラミング言語や開発環境は、開発者である人間を中心にデザインされるべきである」という思想のもとプログラミング支援ツールの新たな方向性を模索したのである。「人間を中心にデザインする」ということは、新しいシステムを提案する前に、開発者がどのような課題に直面しているかを深く理解することが必要不可欠である。彼らが行った重要な貢献の1つは、プログラミング初学者が直面する課題を調査し、6種類に分類して議論したことである⁴⁾。

1. ある機能を実装しているコードを特定する際の、選抜の壁（selection barrier）
2. シンタックスやライブラリの使い方などを学ぶ際の、使い方の壁（use barrier）
3. 複数のプログラムやライブラリがどのように組み合わせられているかを理解する際の、組合せの壁（coordination barrier）

壁（coordination barrier）

4. エラーが起こった際に、予期しない挙動を正しく理解しなければならない、理解の壁（understanding barrier）
5. デバッグの際に内部の挙動が見えないために起こる、情報の壁（information barrier）
6. アルゴリズムやシステムのアーキテクチャをデザインする際の、設計の壁（design barrier）

Koらは、これらの課題が、今後のプログラミングの研究によって解決されるべき問題であると議論した。実際、これらの知見は、その後の研究者が新たなシステムや支援ツールを設計する際の指針となっている（図-1）。たとえば、厳密なシンタックスや型が分からなくてもプログラムできるようにしたAliceやScratch [TOCE'10] といったブロック型プログラミング言語（使い方の壁）や、デバッグの際になぜエラーが出ているのかをGUIから辿れるようにしたWhyline [CHI'04, ICSE'08]^{☆1}（選抜の壁）、プログラミングが実行される際に内部の状態を可視化したPython Tutor [SIGCSE'13]^{☆2}やTimelapse [UIST'13]（情報の壁）、エラーが起こった際に正しい場合と間違っている場合の違いを可視化するTraceDiff [VL/HCC'17]^{☆3}（理解の壁）、プログラム内に画像や姿勢などを直接埋め込めるようにしたSikuli [UIST'09]^{☆4}やPicode [CHI'13]^{☆5}（組合せの壁）、そしてストーリーボード形式で複数デバイ

☆1 <https://www.cs.cmu.edu/~NatProg/whyline.html>

☆2 <http://www.pythontutor.com/>

☆3 <https://github.com/ryosuzuki/trace-diff>

☆4 <http://www.sikuli.org/>

☆5 <https://junkato.jp/picode/>

スのアプリケーションのデザインができるようにした DemoScript [CHI'16] (設計の壁) など、多くのツールがこうした人間中心のアプローチを元にデザインされ実装されてきている。

Brad Myers の Natural Programming 研究グループから始まった人間中心のアプローチは、こうして 2000 年から 2010 年代における、HCI の重要な研究成果として認識されている (ちなみに、彼は HCI 全体の中でも、最も論文が引用されている研究者の 1 人であり、また彼の教え子には、Andrew Ko を始め Jacob O. Wobbrock, James Landay, Rob Miller など、その後 HCI を代表する研究者になった者も多く^{☆6}, 教え子を通じてこの分野を大きく発展させてきた点も特筆すべきである)。

生産性の向上を超えて

上で述べたような研究の 1 つの大きな目標は、開発者の生産性を向上させることであった。そして、2000 年代から 2010 年代前半におけるこうした研究の発展によって、プログラミングを取り巻く環境は大きく変わってきた。事実、これらの研究成果はアカデミックのみにとどまらず、実際のプロダクトにも反映されてきている。たとえば、Microsoft Visual Studio や Swift Playground, Scratch, Arduino, Google Chrome Inspector, Jupyter などのプロダクトを始め、LightTable や Kite といったスタートアップ企業にも、直接的あるいは間接的に影響を与えている。

今後の展望: 3 つの新たなプログラミングパラダイム

こうした背景を受け、HCI における研究の焦点も、開発者の生産性を上げることだけでなく、新たなプログラミングパラダイムを提案するような研究も少しずつ出てきている。ここでは、最近の HCI における、新たな萌芽的なプログラミングの研究を紹介

^{☆6} <http://fredhohman.com/brad-myers-advisee-tree/>

しつつ、今後の展望として特に以下の 3 つの方向性について議論したい。

❖ 直接操作でのプログラミング

1 つ目は、コーディングではなく、オブジェクトなどを直接操作 (direct manipulation) することでプログラムするプログラミングパラダイムである。こうした方向性は特に、データ可視化や、インフォグラフィクス、Web およびモバイルアプリ開発、イラストレーション、3D モデリングといった GUI のアプリケーションにおいていくつかの研究が発表されている。たとえば、Bret Victor の Drawing Dynamic Visualizations^{☆7} や Lyra [EuroVis'14]^{☆8}, Apparatus^{☆9}, Active Drawing [CHI'17]^{☆10} といったプロジェクトは、コーディングではなく、マウスを使い直接 GUI を操作し、データの可視化やインタラクティブなダイアグラムをプログラムする方法を示している。また、別の例として、コーディングをせずにスプレッドシートからドラッグドロップで Web サイトを生成する Gneiss [UIST'14, CHI'15]^{☆11} といった研究が発表されている。ここで重要なのは、テンプレートなどからグラフや Web ページを生成するのとは違い、プログラミングで作るような柔軟性を失わずに作ることが可能な点である。今後は、こうしたプログラミングパラダイムが、AR/VR のアプリの開発や、IoT やロボットの開発といった実世界上でのプログラミングに波及していくと考えられる。たとえば、現在の AR や VR のアプリ開発において、コンピュータ上でプログラムし、それを実世界上でテストし、それを再度コンピュータ上でデバッグする、というように入力と出力に著しい乖離 (gulf of execution and evaluation) がある。こうした非直感的なワークフローが、実世界上でオブジェクトを直接操作するようなプログラミングパラダイムによって解決されることが期待される。

^{☆7} <https://vimeo.com/66085662>

^{☆8} <http://idl.cs.washington.edu/projects/lyra/>

^{☆9} <http://aprt.us/>

^{☆10} <https://www.media.mit.edu/projects/active-drawing/>

^{☆11} <http://www.cs.cmu.edu/~shihpinc/gneiss.html>

❖ クラウドソーシングによるプログラミング

2つ目の方向性は、クラウドソーシングを活用したプログラミングのパラダイムである。クラウドソーシングを活用したプログラミング支援ツールには、クラウドの知見を使ったデバッグのサポートする HelpMeOut [CHI'10] や、コードの推薦を行う CodeX [CHI'14] などがある。しかし、近年では、クラウドソーシングをプログラミングの支援に使うだけでなく、プログラミング自体を行えないか、というチャレンジングな課題に取り組む研究が出てきている。たとえば、Micro-task Programming [UIST'14, ICSE'15] ^{☆12} や FlashTeams/Organizations [UIST'14, CHI'17] ^{☆13} などは、クラウドソーシングをうまくデザインすることで、Eコマースのサイトや、病院の予約管理アプリといった高度で複雑なデザイン/プログラミングが要求されるタスクを、早く、かつ高い質でプロダクトを完成できることを示した。このようなクラウドソーシングを使ったプログラミングの研究はまだチャレンジングな課題が多く残されているが、それだけに今後の研究が期待されている。

❖ データドリブンによるプログラミング

3つ目の方向性は、データドリブンなプログラミングツールである。データドリブンなプログラミングパラダイムは、古くから、ユーザが例を与え、そこから推論してプログラムを生成するといった、例示プログラミング (Programming by Example) が提案されているが、近年では、こうした方向性を発展させた、より高度なコードの推薦・生成が可能になっている。たとえば、既存の Web デザインを組み合わせて生成する Bricolage [CHI'11] ^{☆14}、Microsoft Excel に実際に導入された Flash Fill [POPL'11]、プログラミングの課題を自動で添削し間違いを正す

AutoGrader [PLDI'13] ^{☆15} や OverCode [TOCHI'15] などが挙げられる。一方で、こうした既存のアプローチは、例外的な状況に弱かったり、あらかじめルールを決めなければいけないといった点で課題がある。これらの課題を解決する上で、今後注目を集めるであろうアプローチが機械学習、特にディープラーニングを使ったアプローチである。たとえば、pix2code ^{☆16} はニューラルネットワークを使い、GUI のモックアップから、コードを生成することができるツールである。今後はさらに、ユーザが抽象的な言葉や概念図で説明するだけで、複雑なプログラムやアプリケーションを生成することが可能になると考えられる。その場合、エンドユーザプログラミングという行為そのものが再定義されるような、現在とはまったく違うプログラミングパラダイムになる可能性がある。こうした新たなプログラミングパラダイムを発展させるような研究が今後期待される。

以上、HCI におけるプログラミング研究を、俯瞰的に紹介した。本稿が、HCI におけるプログラミング研究を知るきっかけとなり、今後の日本におけるプログラミング体験の研究の発展の一助になれば幸いである。

参考文献

- 1) Scaffidi, C. et al. : Estimating the Numbers of End Users and End User Programmers, VL/HCC (2005).
- 2) Green, T. et al. : Comprehensibility of Visual and Textual programs, ESP (1991).
- 3) Nardi, B. A. : A Small Matter of Programming : Perspectives on End User Computing, MIT press (1993).
- 4) Ko, A. J. et al. : Six Learning Barriers in End-user Programming Systems, VL/HCC (2004).

(2017年7月30日受付)

^{☆15} <http://sketch2.csail.mit.edu/python-autofeedback/>

^{☆16} <https://github.com/tonybeltramelli/pix2code>

^{☆12} <https://cs.gmu.edu/~tlatoza/>

^{☆13} <http://stanfordhci.github.io/flash-teams/>

^{☆14} <http://hci.stanford.edu/research/bricolage/>

鈴木 遼 ryos.suzuki@colorado.edu

University of Colorado Boulder
<http://ryosuzuki.org/>