1 ライブプログラミングによる滑らかなプログラミング体験

応用 一般

Sean McDirmid（Y Combinator Research）
翻訳：加藤　淳（産業技術総合研究所）

映画「アイアンマン」の中で Tony Stark は，パワードスーツのモデルをインタラクティブなホログラム環境で操作することによって，パワードスーツを設計している．人がプログラムを作る様子を見るのはこれに比べてたいていつまらないものであって，その理由はほとんどの「動作」がプログラマの頭の中で起こっていて見えないことに起因している．そうした「動作」の内訳は，問題を解いたり，コンピュータで扱えるように抽象化したり，それをコードに落とし込んだりといったことである．ライブプログラミングは，こうした「つまらない」プログラミングを解決するために，頭の中で起こることをコンピュータの中に移していこうという試みである．ライブプログラミングでは，まず，従来は別々の動作であったコード編集とデバッグを区別せず行えるようにする．そうすると，プログラマはコードの編集結果を即座にプログラムの新たな振舞いとして観察できるようになるため，頭の中で編集結果を想像する必要がなくなる．結果として，より滑らかなプログラミング体験が実現する．ライブプログラミング環境の基礎的な実装は簡単で，コードが編集されるたびにプログラム全体を再実行し，結果を継続的に提示すればよい．次なる技術的な課題は，これをより大規模なプログラム開発にも適用できるよう「スケール」させるところにある．たとえば，プログラム全体を再実行せず差分だけ実行していくといった方法が考えられる．しかし，ここで注意しておきたいのは，ライブプログラミングの成功は，実装の洗練度合いよりも，いかにプログラミング体験をデザインできるかにかかっているということである．

先に述べたとおり，ライブプログラミングは，コード編集とデバッグの境界をなくして，プログラムの実行時の振舞いについてプログラマに継続的にフィードバックを与えてくれるものである．そう一口にいっても，フィードバックがどんなかたちを取るべきかというのはまったく自明ではない．極端なことをいえば，ユーザの指定した引数を与えたプログラムの出力結果をコード編集のたびにアップデートして見せるだけでも，継続的なフィードバックたり得る．しかし，そんな出力ではプログラムの振舞いはほとんど明らかにならない．たとえば，複雑な計算をした上で最終的な解だけを出力するプログラムのデバッグでは，解だけアップデートされても役に立たないだろう．役に立つライブプログラミングを実現するには，プログラムの振舞いのうち，解こうとしている問題に関連のある内容を適切に可視化する必要があるのだ．それを実現するには，printf 文で標準出力に関連する情報を出力させるような明示的挿入（explicit instrumentation）か，行ごとの状態変化を一覧表示するような普遍的可視化（pervasive visualization）を行えばよい．この 2 つのアプローチの間は，利便性を取るか（普遍的可視化），大量の情報に圧倒されないようにするか（明示的挿入）というトレードオフの関係がある．たとえば DejaVu [1] という統合開発環境は，Kinect（通常のカラー画像に加え深度情報と人の姿勢情報を取得できるカメラ）のプログラム開発において，フレームごとの情報可視化を行うことでライブプログラミングを可能にしたものであるが，プログラマが可視化したい内容を自由に選べるようになっている．Bret Victor もエッセイ「Learnable Programming」[2] の中で，プログラムの状態情報，データ，そして処理の流れを可視化するためのさまざまな方法を紹介している．

さて，プログラムの振舞いを適切に可視化できたとして，次なる問題は，それがコード編集のたびに「ぎくしゃく」してしまいかねないことである．たとえばコード中の定数 200 を 175 に変更するためにキーボードを使うなら，値の推移は 20, 2, 1, 17, そして最後に 175 となるだろう．この値が画面上の矩形の位置を表しているなら，矩形の位置は乱雑に飛び回って，ライブプロ

グラミングのフィードバックを邪魔で無益なものにしてしまう．Chris Hancock は，ライブプログラミングは水道のホースで的に水をかけるようなものになるべきだと提案している[3]．弓矢を使うのと比べて，水道のホースで的に水をかけるのは簡単だ．なぜなら，射手はとにかく水流を見て，水がかかるまで狙いを少しずつずらしていけばよいからである．ライブプログラミングにおいては，小さなコード編集の結果はプログラムの出力における小さな変化に対応しているべきだといえる．そのとき，プログラミングという行為は，さながら狙いを探る連続関数のようになるだろう．これを実現する 1 つの方法は，値の編集を滑らせて行うこと（scrubbing）である．たとえば定数 200 を選択したときにスライダーを表示すれば，値を編集する際に 175 へ至る編集中の中間値が 199 から 176 まで連続的に変化するため，それに応じて矩形の位置も滑らかに移動する．

ライブプログラミングの水道ホースの原則は，しかしながら，一般化がとても難しい．プログラム中に何らかの関数呼び出しを加えると実行結果に劇的な変化が起き得るし，それだけで劇的な変化が起きなかったとしても，ほかの関数呼び出しとの兼ね合いで「小さな連続的変化を起こす小さなコード編集」の単位を探ることは非常に困難である．では，コード編集のたびに実行結果をスムーズにアップデートできないなら，因果関係を逆転して，実行結果をスムーズに編集することでコードの側を書き換えるというのはどうだろう？　実行結果を直接操作（direct manipulation）で編集することは，編集操作によって対象のオブジェクトを即座に変更することにほかならず，まさに「ライブ」である．こうした体験を実現するには，プログラムの出力が直接操作に向くように仕立てる必要がある．たとえば，グラフィカルなプログラムでは，図形を描いたり編集したりできればよい（文献 2）の “create by reacting” を参照）．また，プログラムの動作した履歴を操作できるようにすることも考えられる．

直接操作の結果を汎用的なプログラムにするには，操作対象を段階的に抽象化（gradual abstraction）する必要がある．これは Pygmalion や Tinker[4] のような例示プログラミング（programming by demonstration）に似ているが，同時に，文献 2）の “create by abstracting” にも似ている．たとえば，直接操作で 2 つの矩形が水平方向に整列したら，段階的抽象化によってその水平方向の座標は 1 つの変数で表され，結果として矩形は常に整列した状態を保つようになる．さらに，ライブプログラミング環境は，プログラマが直接指定した出力結果を参考にして，現在のプログラムに正しく追加できる抽象化はどのようなものか，プログラマに提示する．今日ではまだ夢物語であるライブプログラミング環境において，プログラマはプログラムの出力を操作し，抽象化し，操作し，抽象化し，という段階を繰り返していく．この過程でプログラマの思考はもっとコンピュータに対して細切れに伝わるようになり，プログラマの認知的負荷が削減できる．そして何よりも，そうしたプログラミングは見ていてもっとわくわくする行為になるだろう．

ライブプログラミングは，プログラミングとプログラムの実行の間の溝を取り払うことで，プログラミングをもっと重層的な体験にしようという試みである．そのためには，プログラミング体験（PX）の課題を数多く解決していかなくてはならない．たとえば，プログラムの出力をプログラマにとって有用なフィードバックにするにはどうしたらいいか，コードの変更とプログラムの実行結果の変化をどう滑らかに関連付けて見せるか，プログラムの出力を直接操作した結果をどう段階的にコードへ抽象化していくか，といった課題である．課題は山積しているが，これらを解いていく中で，映画の中にしかないプログラミングのサイエンスフィクションが，本当に現実のものになるかもしれないのだ．

参考文献

1）Kato, J., McDirmid, S., Cao, Xi. : DejaVu : Integrated Support for Developing Interactive Camera-based Programs, UIST 2012.
2）Victor, B. : Learnable Programming(2012), http://worrydream.com/LearnableProgramming/
3）Hancock, C. : Real-time Programming and The Big Ideas of Computational Literacy, Doctoral Dissertation, MIT Media Lab, Cambridge, MA (2003).
4）Watch What I do, edited by Allen Cypher, MIT Press, Cambridge, MA (1993).

（2017 年 7 月 30 日受付）

Sean McDirmid　sean.mcdirmid@ycr.org

2005 年ユタ大学博士課程修了．博士（コンピュータ科学）．EPFL で Scala に関する研究に従事（2005 ～ 07），Microsoft China にプロトタイプ開発者・研究者として勤務（2007 ～ 16）した後，現職（2016 ～ 17）．ライブプログラミングと，よりよいプログラミング体験の研究を専門とする．

加藤　淳（正会員）　jun.kato@aist.go.jp

2014 年東京大学大学院情報理工学系研究科博士課程修了．博士（情報理工学）．産業技術総合研究所研究員．SIGPX 発起人・幹事．プログラマと多様な人々が協力できる環境の研究に従事．https://junkato.jp/ja