バグの闇を照らす ソフトウェア工学の光

計算工学専攻
権藤 克彦 研究室

権藤 克彦 教授　1966年兵庫県生まれ。東京工業大学大学院理工学研究科情報工学専攻博士課程修了。2011年より、学術国際情報センター教授。

現代社会はコンピュータによって支えられている。そのコンピュータはソフトウェアに支えられて動いている。では、ソフトウェアを開発する人間は一体何に支えられているのだろう。それこそがソフトウェア工学である。権藤先生は、ソフトウェア開発者がバグを生み出さないようにするため、開発者を支えるさまざまなツールの開発を行なっている。世界を支えるソフトウェア工学の世界へ、ようこそ。

ソフトウェア工学とは

私たちが暮らしている現代社会では、さまざまな場面でソフトウェアが利用されている。携帯電話やパソコンといった身近なものだけに限らず、証券取引所など、社会を支える重要な場所においても利用されているのだ。ソフトウェアは、機械が行う動作についての判断や思考を担っており、動物の脳に相当する重要な役割をもっている。携帯電話やコンピュータなど、複雑な動作をおこなっている機械に頼る現代社会は、ソフトウェアなしでは成り立たないのだ。

しかし、ソフトウェアは人間が作るものである以上、見落としや入力間違いなどによるミスが必ず発生してしまう。このミスにより意図していない動作、いわゆるバグが混入してしまうことが多々あるのだ。この動作によって、さまざまな問題が発生してしまう。

例えば、携帯電話に搭載されているソフトウェア中のバグによって、ある特定の条件下でその携帯電話を利用すると、電話帳などのすべてのデータが消えるという事態があった。この事態を受けて、携帯電話のメーカーは約23万台もの端末を回収し交換せざるを得なくなり、約30億円もの損害を受けてしまった。また、証券取引所に用いられているソフトウェアにバグが発生し、すべての動作が2回以上行われてしまう状態になったこともあった。このバグを修正するために取引が長時間停止してしまい、400億円を超える損害が発生したと言われている。

このような事態は頻繁に起こっており、社会にとって大きな損害となっている。これらはすべてソフトウェア中のバグに起因するものである。バグをなくすことができれば、ユーザはソフトウェアのバグで困惑することもなくなり、また社会が損害を受けるような状況もなくなる。つまり、バグの殲滅が社会全体に利益をもたらすといっても過言ではない。

そこで、先生は世の中からバグをなくすことができないかと考え、ソフトウェア工学を用い始めた。このような理念のもとで、先生はソフトウェア工学の研究を行なっている。

ソフトウェア工学とは、一般のプログラマでも効率よく、バグのない高品質なソフトウェアを作れるようにすることを目標とした学問分野である。この学問分野はさまざまな分野をカバーしている。例えば、開発自体に用いるプログラミング言語や手法といったソフトウェア開発に直接関係する分野に加え、ソフトウェアの設計や品質管理といった開発を間接的に支援するような分野も含まれている。このように、ソフトウェア工学はさまざまな分野を含み、そこにはそれぞれの目標に対応するアプローチが存在しているのだ。

その中でも、先生はツールの作成と活用という、間接的に開発者の手助けとなるようなアプローチを取ることでバグの殲滅を狙っている。ツールとは、エディタやバージョン管理ソフト、統合開発環境などのように、ソフトウェア開発をさまざまな方面から助けてくれるソフトウェアのことである。さまざまなツールを作成し、その活用手段を提案することで、どんな人でもバグのない高品質なソフトウェアを作れるようになるのではないか、と先生は考えたのだ。

では、具体的に先生がどのようなツールの作成や活用について研究しているのかを見ていこう。

足跡を残すTBCppA

ソフトウェア開発において、開発しているプログラムの構造を解析し、把握することは重要である。構造を把握することで、意図した通りの動作をするか、また意図しない動作が発生したときにどこを直せば良いのかがわかるからである。この解析を行うツールを解析用ツールと呼ぶ。

その中でも先生は、C言語のプリプロセッサという機能に関する解析用ツールを開発した。C言語とは、ソフトウェア開発に用いられるプログラミング言語の一つである。現存するソフトウェアは、この言語を用いて作成されているものも多い。

C言語を用いるとき、プリプロセッサという機能は欠かすことができない。プリプロセッサとは、人間が書いたプログラムのソースコードを定められたルールに従って置換することで、機械が構造を把握しやすくなるような下地を作り出すものである(**図1**)。例えば、使用しているOSがWindowsのときはWindowsで処理できるように、Macのときは Macで処理できるように置換する。

プログラムの解析のためには、プリプロセッサにより置換されたソースコードを解析することに加えて、置換されたソースコードを元に戻すための履歴が必要だ。なぜなら、プリプロセッサで置換される前のコードでは、置換後の変化を予測できないため、解析ができないからだ。また、置換

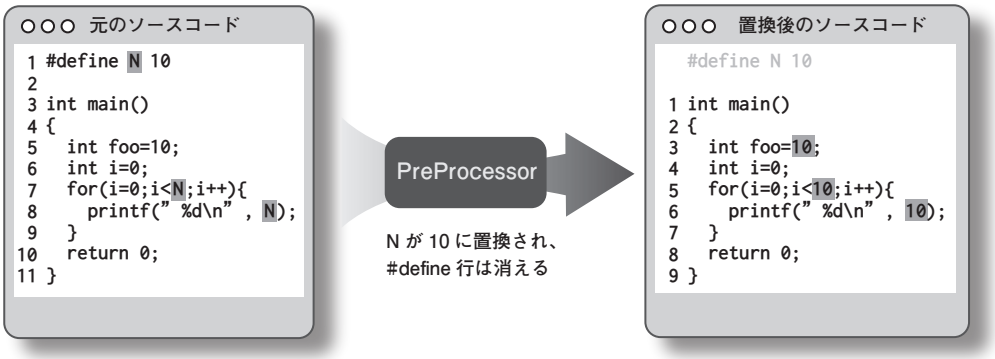

図1　プリプロセッサの置換機能

プリプロセッサの機能により、ソースコードのNという箇所がすべて10に置換される。しかし、置換後のソースコードに現れる10が元々あったものなのか、置換されて生じたものなのか判別できない。

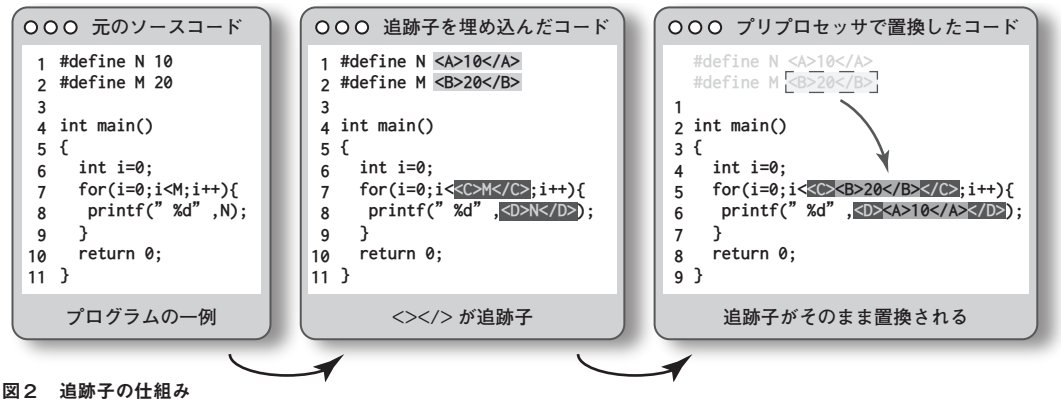

図2　追跡子の仕組み

追跡子を埋め込んでから置換することで、どの部分が何と置換されたかという履歴を残すことができる。

されたソースコードを解析しても、置換されたソースコードは元のソースコードから大幅に変更されているため、元のソースコードとの関連性をもったデータを得ることができない。つまり、置換されたコードを解析して得られるデータと、元に戻すための履歴を合わせて、初めてプログラムの構造を解析することができるのだ。

しかし、覆水盆に返らずという言葉もあるように、一度置換されたものを元に戻すのは難しい。プリプロセッサはどこがどのように置換されたかについての履歴をまったく残さないからである。そこで、先生はプリプロセッサによる置換の履歴を取得する、TBCppAというツールを開発した。

TBCppAを開発する際、先生はプリプロセッサによる置換の動きを追うため、追跡子というアイデアを用いた。追跡子とは、生物分野の実験における染色のように、ある部分をプログラムの動作を変えることなく目立たせ、その動きを追うために使うものである。TBCppAでは、ソースコード中に追跡子を埋め込み、実際にプリプロセッサにかけることで、追跡子からどのように置換が行われたか履歴を得ることができるようにした(**図2**)。

このTBCppAによる置換の履歴のデータと置換後のソースコードを用いることで、プログラムの構造を解析することができる。その一例として、コールグラフを作ることがある。コールグラフとは、プログラム中の呼び出し関係を可視化したものである(**図3**)。この情報を参考にすることで、呼び出し忘れている部分を発見したり、頻繁に使われている部分を高速化したりするなどして、ソフトウェアの改良を進めていくことができるのだ。

TBCppAのような、置換されたものを元に戻そうとするツールは数多く存在する。例えば、置換の履歴を残すようにプリプロセッサ自体を改造し、その履歴を利用することで元に戻すことができるツールがある。しかし、このようなツールは、元のプリプロセッサが更新された場合、それに合わせて更新する必要があり多大な労力が必要である。一方で、TBCppAは、置換という基本的な機能を利用して履歴を残すため、プリプロセッサが変更されてもTBCppAを大幅に変更せずに利用することができる、実用的なツールなのだ。

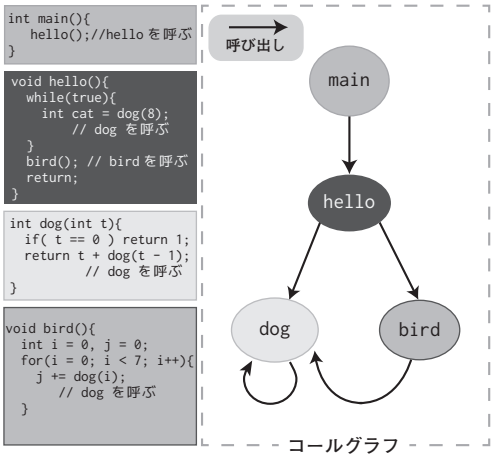

図3　コールグラフ

コールグラフに図式化することで、呼び出し関係を直感的に把握することができる。

つながりが見えるMieruCompiler

先生は解析を助けるツールの開発に加え、教育のためのツールの開発も行なっている。

コンパイラは、ソフトウェア開発において開発者が書いたソースコードを機械語というコンピュータが理解できる言語に変換するツールであり、いわば開発者とコンピュータの橋渡しという重要な役割をもつ。このツールについての理解を深めることは、ソフトウェア開発をより効率的に行うには欠かせない。実際に、多くの大学の情報系の学部ではコンパイラの教育が行われている。

コンパイラは大きく2つの機能をもつ。ソースコードを機械語に変換する機能と、作られた機械語を組み替えることにより、高速化を目指す機能である。学部でのコンパイラの学習において重きを置かれている機能は前者である。コンパイラがコードをどのような機械語に変換するかを知ることによって、コードを書く段階でそのコードがどのように変換されるかを予測することができる。これにより実行速度の速い、コンピュータにフレンドリーなコードを書くことができるのだ。

しかし、コンパイラの学習は非常に難しい。実際のソフトウェア開発に用いられるコンパイラは、学部でのコンパイラ学習においては重視されない、実行を高速化する機能がその構造の多くを占めているため、構造が非常に複雑になっている。そのため、ソースコードから機械語への変換を行っている部分を見つけることが難しいのだ。それに加え、ソフトウェア開発において実際に使われる言語は文法が非常に複雑であるため、コードの解析が難しく、コンパイラ内の機械語へ変換する部分がさらに複雑になってしまう。

この難しさを解決するため、複雑な部分を排除し、仮想的な環境で動かすことを目標とした教育向けのコンパイラが存在する。だが、このようなコンパイラから作り出された機械語は実際のコンピュータ上で直接実行することができないため、実践的な理解の妨げとなってしまう。また、本物の環境と異なるため、学習の動機が弱くなってしまうことにもつながる。

先生はこれらのデメリットを解決する手段を考えた。実在する言語を簡略化した言語XCと、それに対応し、実際にコンピュータで動かせるような機械語を作り出すコンパイラXCC、およびコンパイラがどのような動きのもとでソースコードを変換するかを可視化するMieruCompilerという一連のツールを作り出したのだ。

XCとは、実際のソフトウェア開発で広く用いられているC言語というプログラミング言語から主要な機能のみを抜き出し、シンプルにした言語である。ソースコードの解析を難しくしてしまう機能を排除することで、コンパイラ学習者にとって理解しやすいものとなっている。また、XCCはXCを変換し、実際のコンピュータ上で動く機械語を出力するコンパイラである。XCCは一般に用いられているコンパイラとは違い、ソースコードから機械語へ変換する部分に重点を置き、実行を高速化するための部分は複雑化しないように配慮されている。このことにより、コンパイラ学習の際、XCC中のソースコードを機械語に変換する部分を簡単に見つけ出せるのだ。

コンパイラがソースコードを機械語に変換するときに使う情報を読み解くことは、変換の過程を理解する際に重要である。しかし、この情報はツールを用いて取得する必要があるが、量が膨大であるため、読み解くことが難しい。そのような苦労を減らすために登場したのがMieruCompilerである。MieruCompilerはXCCがソースコードを変換する際に使った情報を取得し、その情報をわかりやすく一画面に表示する。

また、コンパイラの学習のためにはその情報の相互関係を知る必要がある。そこで、水平スライスという機能を提供するのがMieruCompilerだ。この機能は選択された情報と関係する部分をすべてハイライトして表示するというものだ。例えば、ソースコードのある部分を選択すると、それに対応する情報がすべてハイライトされる(**図4**)。

ソースコードの情報の関連を表示するツールは他にも存在するが、例えばソースコードの行単位でしか対応関係を調べることができないという欠点があった。つまり、対応するデータがどのような文脈で用いられたか知ることが難しかったのだ。

先生が開発したこのMieruCompilerは、ソース

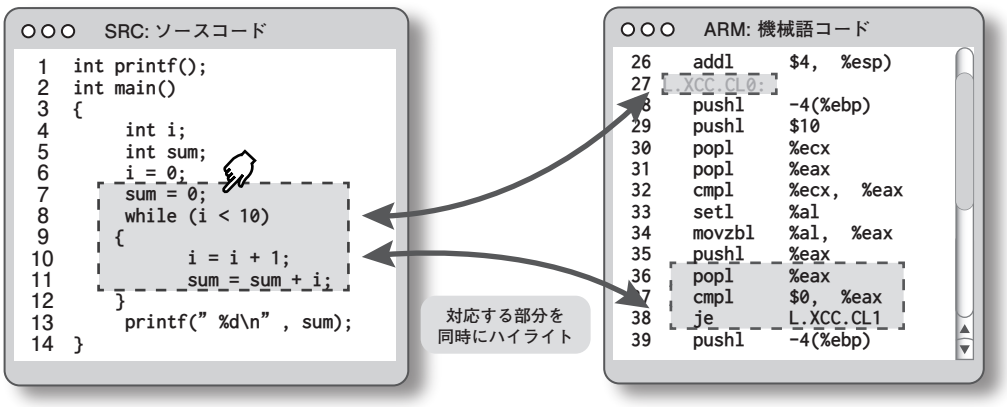

図4　水平スライス

ソースコードのある部分を選択することで、それに対応した情報をハイライトして表示することができる。

コードの段落単位で対応関係を表示するという、人間のソースコードのとらえ方に近い表示方法を取る。そのため、コンパイラがどのように情報を用いてソースコードを変換しているのかなどの、コンパイラの仕組みがより理解しやすくなるのだ。

水平スライスという機能は、過去に類似したアイデアはあったものの、実際に作るのは困難だった。世界で初めて水平スライスの機能を搭載したMieruCompilerがどれだけ凄いかがわかるだろう。

先生が作り出したこれらのツールは実際の教育現場でも利用され、一定の成果を得ている。さらに、講義を受けた学生からの評判も良い。非常に面倒になりやすいコンパイラの学習をより簡単に、より身近にすることに成功したのだ。

世界を救うソフトウェア工学

これまで見てきた通り、先生はソフトウェア開発技術の改良や、バグを生み出さないような技術をもつ人材の育成を目指し、日々ソフトウェア工学の研究を行なっている。しかし、プログラミング技術が進歩する速さに比べ、ソフトウェア工学の進歩はずっと遅い。その結果、ソフトウェア開発を支援するためのソフトウェア工学がプログラミング技術の進歩についていけず、支援するという目標を達成することが難しくなっているのだ。

先生はその理由として、多くの研究者がソフトウェア工学について、自然現象のように普遍的に適用可能な法則を探そうとしているからではないかと考えている。というのも、一般的に工学という分野は、自然法則にのっとった考えが基本となっているからだ。電気分野における電流の流れ方を決定する法則や、機械分野における運動を決める法則のようなものである。

しかし、先生は人間に関する、一見泥臭く注目を集めにくい方面もソフトウェア工学の重要な側面だと考えている。機械上で行なわれる動作にのみ着目するのではなく、人間自体にも着目することで、ソフトウェア開発の効率化ができるのではないかと考えているのだ。例えば、開発者がただずっと働いている組織に比べ、仕事の段階ごとに目標を定め、休息を与える組織の方が開発者の調子が良くなり、バグが少なくなる、ということが知られている。

ソフトウェア工学には、革命的な進化、いわゆる銀の弾丸が存在しないと言われている。しかし先生は地道な努力によりソフトウェア工学を進歩させていこうとしている。ソフトウェア工学という、世界を救う学問への先生の挑戦に期待したい。

執筆者より

取材ではわかりやすく研究を説明していただき、情報への興味がより膨らみました。お忙しい中、取材に応じて下さった権藤先生に心より御礼申し上げます。　（関谷 翠）