具体化と抽象化の狭間で

— デバッグのエッセンス —

結城浩

2014 年 1 月

次の式を見て、何か気づくことはありますか。

$$1 + 8 + 27 + 64 = 100$$

多くの人が $1, 8, 27, 64$ に何か**規則性**がありそうだと思うでしょう。そして注意深い人はこの 4 個の数がすべて 3 乗数になっていることに気づきます。つまり、上の式は次のように書けるということです。

$$1^3 + 2^3 + 3^3 + 4^3 = 100$$

さらに、左辺で気づいたことを右辺に**適用**するとどうなりますか。右辺の 100 は 3 乗数ではありませんが 2 乗数です。ですから冒頭の式は、

$$1^3 + 2^3 + 3^3 + 4^3 = 10^2$$

と書けることになります。

もしかしたら、**一般化**して、こんな仮説が成り立つのではないでしょうか。

$$1^3 + 2^3 + 3^3 + \cdots + n^3 = N^2 \qquad \textbf{(仮説)}$$

ここには発想の飛躍があります。しかし、小さな n で**実験**すると確かに成り立つことがわかります。$n = 1, 2, 3, 4$ で実験すると、それぞれ $N = 1, 3, 6, 10$ で仮説が成り立つのです。

ここで出てきた $N = 1, 3, 6, 10$ はどんな数列でしょう。数列の謎を解くときの基本的な**道具**は、隣り合った数同士の差（階差）をとることです。

これで、階差は 1 ずつ増えているらしいことが予想されます。仮説の式で n が 1 増えるとき N の階差も 1 増えるので、実は次のようなきれいな式が成り立つのです。これは簡単に証明することもできます。

$$1^3 + 2^3 + 3^3 + \cdots + n^3 = (1 + 2 + 3 + \cdots + n)^2$$

実は、プログラミングにおける「デバッグのエッセンス」がここまで考えてきたことの中に示されています。

あなたがデバッグを行うとき、いままでお話しした要素に出会うはずです。目の前に現れる現象の**規則性**に気づき、**一般化**し、**仮説**を立てること。そして、仮説を立てるために**道具**も知らなければいけません。

現象をよく把握するためには、意識して規則性を見つけようとする態度が必要です。そのためには現象の細かいところまで注意深く見ることが大切になるでしょう。

規則性を見抜くためには構成要素に対する知識も必要です。ちょうど 27 を 3^3 だと見抜くように、あなたが扱っているプログラムの基本的な構成要素の理解が大切です。

規則性からさらに一般化する大胆な飛躍も必要です。その際には具体的な数から離れ、変数や文字を使った別の表現が大切になってくるかもしれません。それは仮説となって表されます。

一般化して得られた仮説は、検証が必要です。小さな数で仮説をためしたように、自分のプログラムが自分の仮説通りになっているかどうかの検証が大切になります。この検証がなければ単なる推測で終わってしまうからです。

デバッグは、**問題解決**にほかなりません。あなたは毎日、多くの問題解決を行っているでしょう。それをいきあたりばったりの活動にしないためには、規則性を見つける、仮説を立てる、検証する……といった個々の活動を意識することが大切です。

問題解決には、一見矛盾する二つの態度が必要です。一つは、具体的な問題にどっぷりつかり、細かいところまで注意を払うこと。もう一つは、細かいところに惑わされず、抽象化した本質を見抜くこと。

すなわち、具体化と抽象化の狭間を意識的に行き来することが、問題解決には不可欠なのです。

あなたの問題解決が実り豊かなものになりますように。

http://www.hyuki.com/
初出: 技術評論社 Software Design 2014 年 2 月号