IT技術とCoqの世界

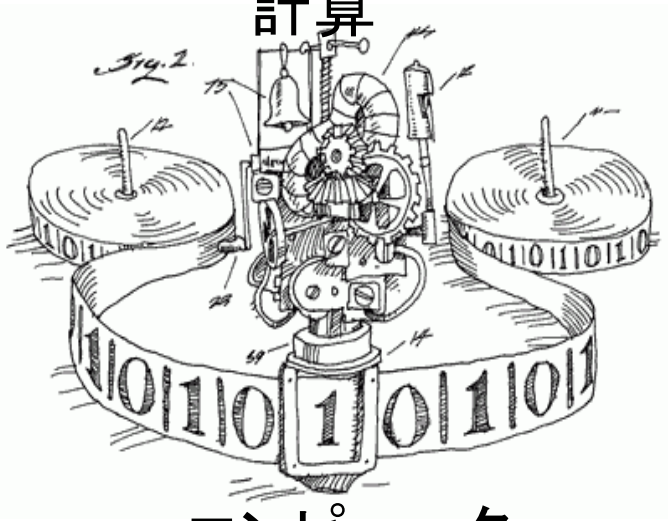
「証明」＝「プログラム」＝「計算」の意味を考える

数学者

プログラム

プログラマ

計算



コンピュータ

When exhaustive testing is impossible —i.e., almost always— our trust can only be based on proof (be it mechanized or not).

-- Dijkstra

Given the proof, deriving a program justified by it, is much easier than, given the program, constructing a proof justifying it.

-- Dijkstra

Most working engineers view machine-checked mathematical proofs as an academic curiosity, if they have ever heard of the concept at all. In contrast, activities like testing, debugging, and code review are accepted as essential. They are woven into the lives of nearly all developers

-- Adam Chlipala

Coming Soon: Machine-Checked Mathematical Proofs in Everyday Software and Hardware Development

-- *Adam Chlipala*

IT技術とCoqの世界
Agenda

- なぜ、開発は人間によって担われているのか？
- 機械は論理的＝数学的推論能力を持つ
  - 「証明」＝「計算」
  - 「証明」＝「プログラム」
  - 「プログラム」＝「計算」
- 人間と機械の関係を考える

  人工知能論の未来

はじめに

- 筆者は、これからは、開発者にとってCoqを学ぶことが重要になるだろうと考えている。小論はその理由と背景を、まとめたものである。
- 冒頭の「なぜ、開発は人間によって担われているのか？」は、人間中心の複雑な構造を持つ開発の世界は、コンピュータの論理的推論能力の利用で、大きく変わるだろうという展望を述べている。
- 現在では、多くの人にとって、それは非現実的なビジョンだと思われるかもしれない。ただ、それは、バグのないセキュアなソフトを作るという社会的に重要な課題に対して、もっとも根本的な解答を与えるものである。同時に、それは、開発コストの大幅な削減という経済合理性を備えている。Coqは、そうした見通しを実現する上で欠かせないツールになろうとしている。

はじめに

- 次章の「機械は論理的＝数学的推論能力を持つ」は、先に見た開発過程の刷新の鍵であるコンピュータの「論理的＝数学的推論能力」をテーマにしている。「機械が論理的に推論する」とか「機械が数学的証明を行う」という言葉が腑に落ちない人は多いと思う。
- 「計算」「証明」「プログラム」という三つの言葉は、もともと別の意味を持っている。それは今でも同じだ。この章は機械の能力を人間がどう認識してきたかを、この三つの概念の関係の認識の歴史をメイン・ストーリーにして述べたものである。
- まず、「計算」と「証明」が、本質的には同じものであるという認識が生まれ、ついで、「プログラム」は「証明」とみなせるという認識が生まれる。理論的には、この二つの発見で、三項の関係は明確になった。「コンピュータは、プログラムの実行という形で、証明を計算する」のである。

はじめに

- さいわいなことに、現代人は、「プログラム」と「計算」の関係については、十分な直感を持っている。この直感に依拠すれば、「チャーチ＝チューリングのテーゼ」や「カリー＝ハワード対応」に遡及しなくても、三項の関係が示す、「機械が論理的・数学的推論能力を持つ」という直感が得られるのではというのが、僕が期待しているところである。
- 残念ながら、「プログラム」と「計算」の関係だけでは、「証明」へのリンクが欠けている。僕のもう一つの期待は、Coqでの証明構成の経験が、このリンクを作ってくれるだろうということである。
- これまで見てきたような、人間には何ができ、機械には何ができるのかという問題は、人工知能論の大きなテーマである。ただ、今回は、こうした問題について触れることができなかった。いずれ、機会を改めて、筆者の考えを述べたいと思う。

なぜ、開発は人間によって担われているのか？

人間がしていることと、コンピュータがしていること

ITビジネスとは、単純化していうと、ソフトウェアを
ハードウェア上で動かして、それをサービスとして
顧客に提供するビジネスである

ITビジネス

ソフトウェア　　ハードウェア　　サービス

ITビジネスの中核部分は、ソフトとハードからなる
IT技術である。

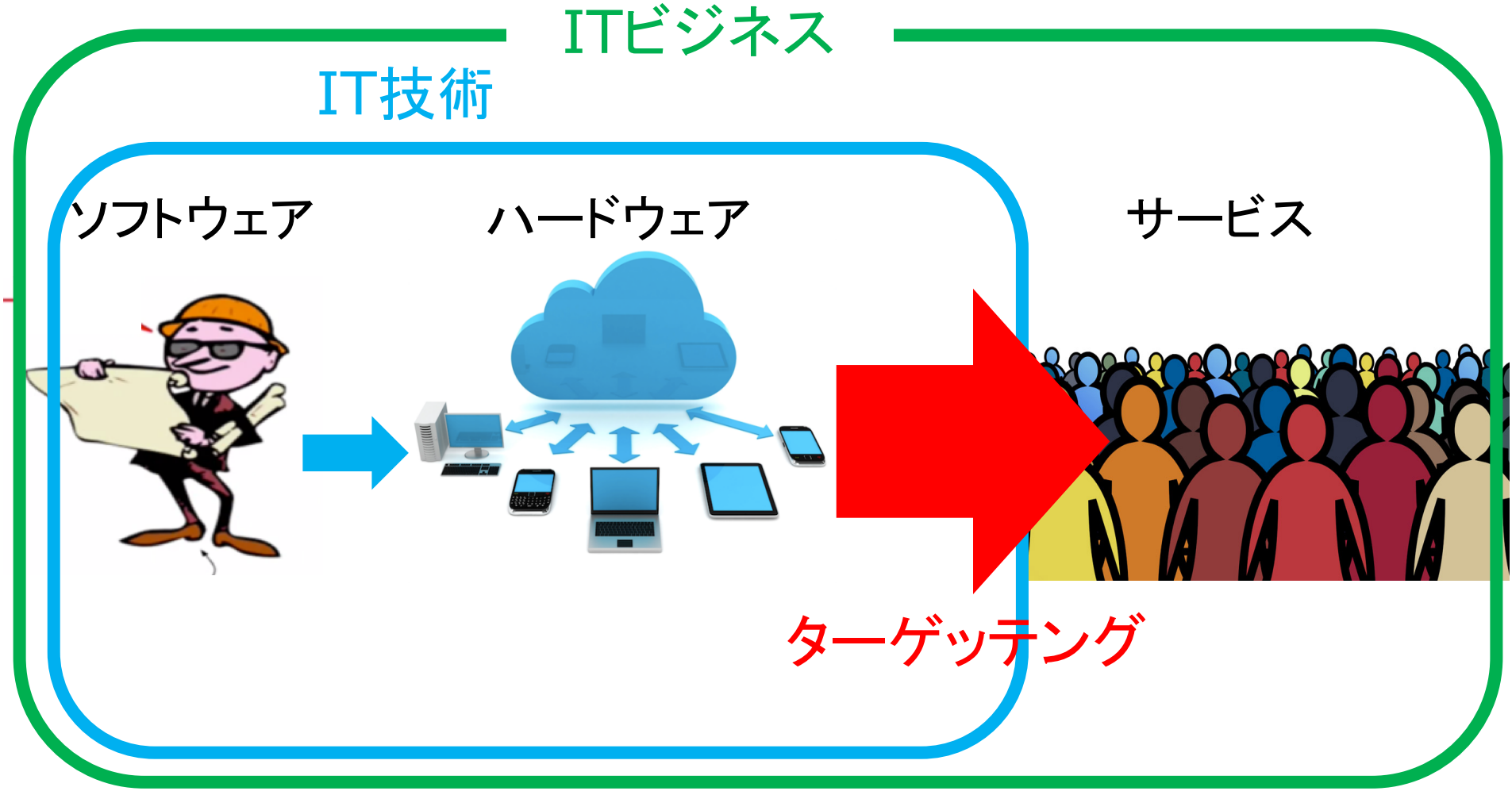
ITビジネス

IT技術

ソフトウェア　　ハードウェア　　サービス

プログラマ　　コンピュータ

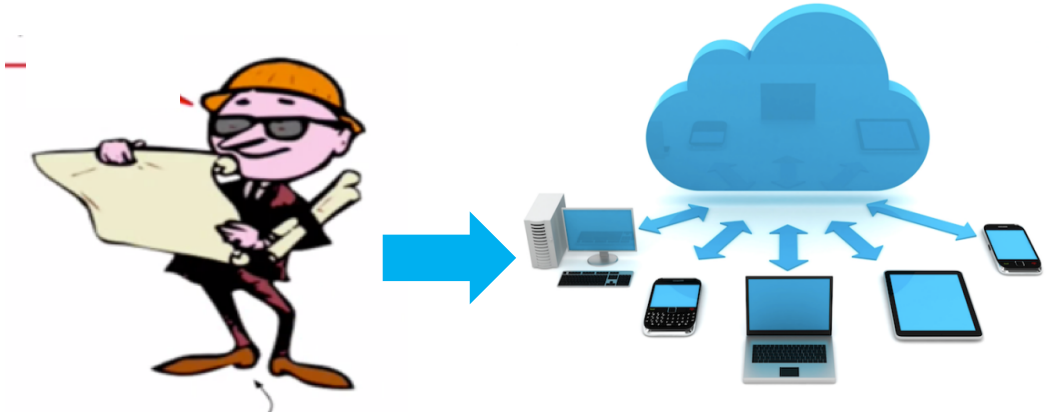
現代のITビジネスの最も大きな関心の一つは、
消費者へのサービスのターゲッテングに、IT技術（例えば、「AI技術」）を利用することなのだが、それについては別に述べる。



ここでは、ソフトウェアとハードウェアからなるIT技術の
単純なモデルから出発して、人間がしていることと
コンピュータがしていることを考えよう

IT技術

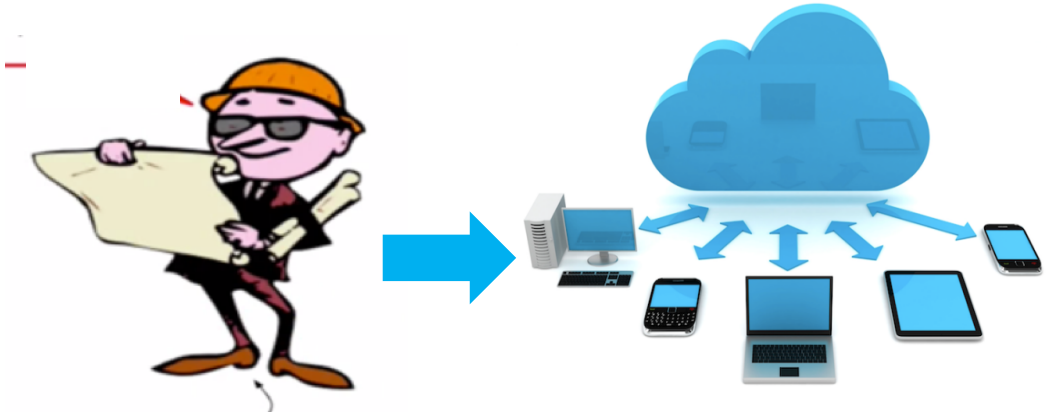
ソフトウェア　　ハードウェア



単純に考えれば、プログラマがプロプラムを作成し、
コンピュータがそれを実行する。人間がソフトを作り
コンピュータがそれを実行する。それは間違っていない。

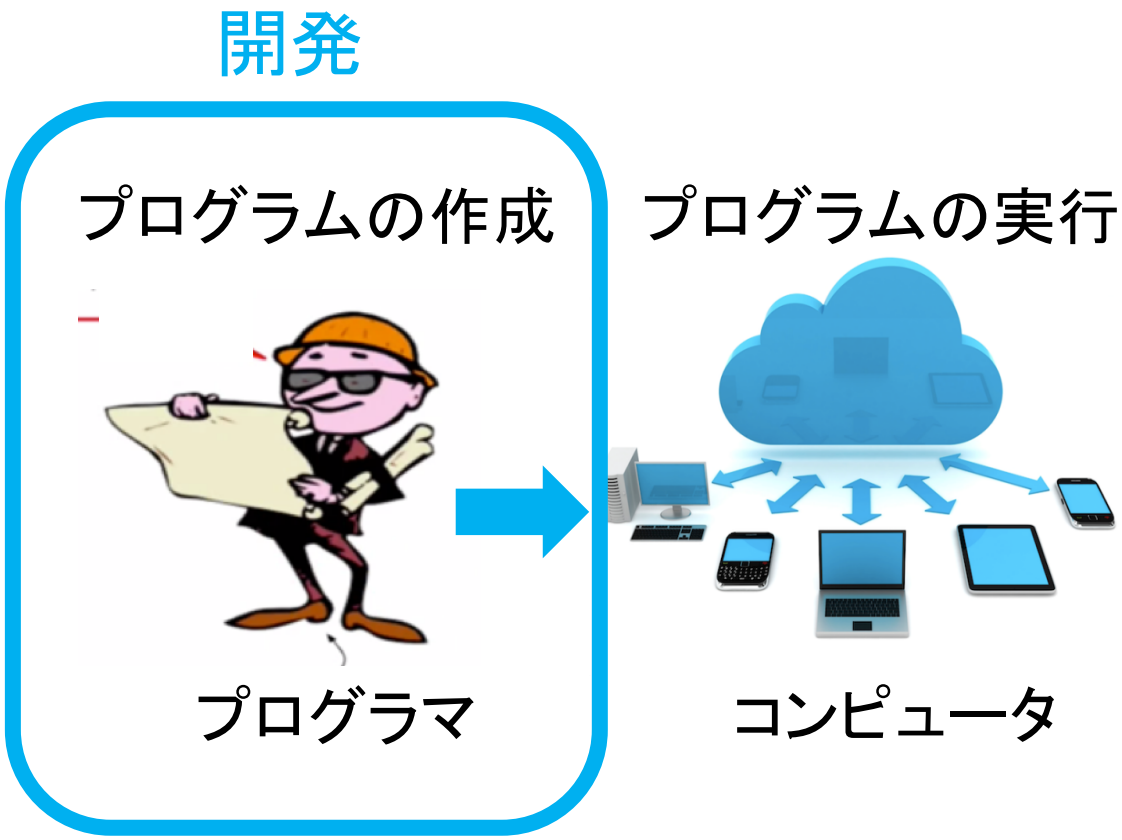
IT技術の単純なモデル

プログラムの作成　　プログラムの実行



プログラマ　　コンピュータ

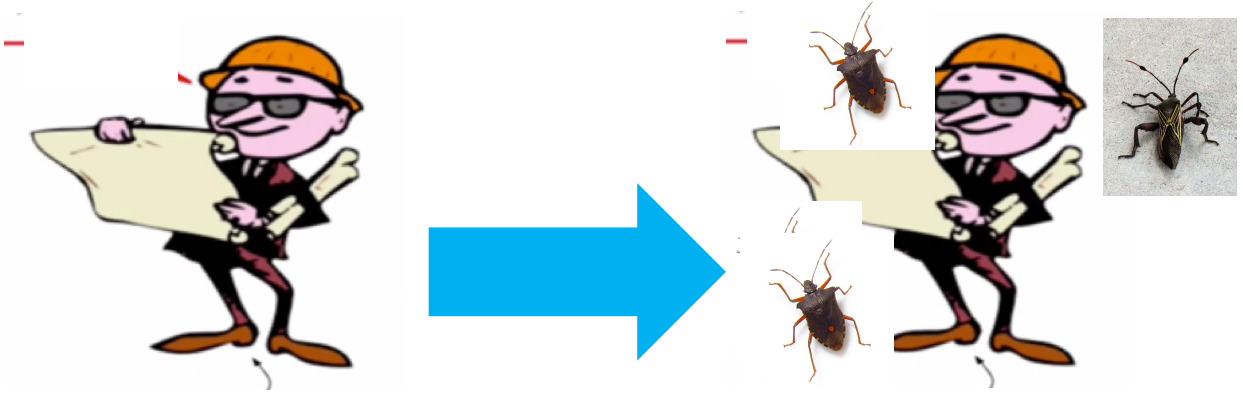
コンピュータとプログラムの関係は、シンプルである。規模の違いはあっても、それは、均一にスケールする。
それに対して、プログラマによるプログラム作成の過程（開発過程）は、複雑である。以下、それを見ていこう。



バグのないプログラムが、すぐにできるわけではない。
開発では、プログラムのテストは欠かせない。

開発

プログラムの作成　　プログラムのテスト

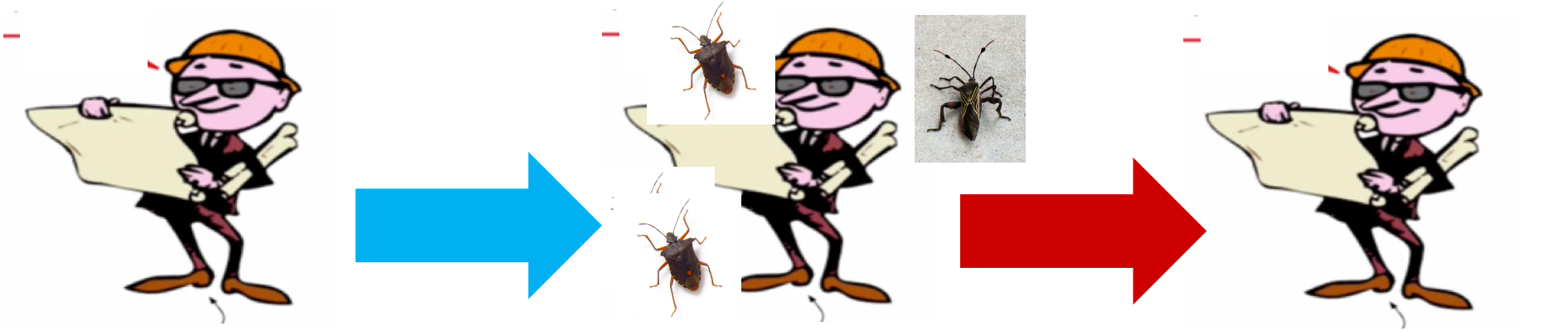


プログラマ　　プログラマ

テストでバグが見つかれば、デバッグをする

開発

プログラムの作成　　プログラムのテスト　　プログラムのデバッグ

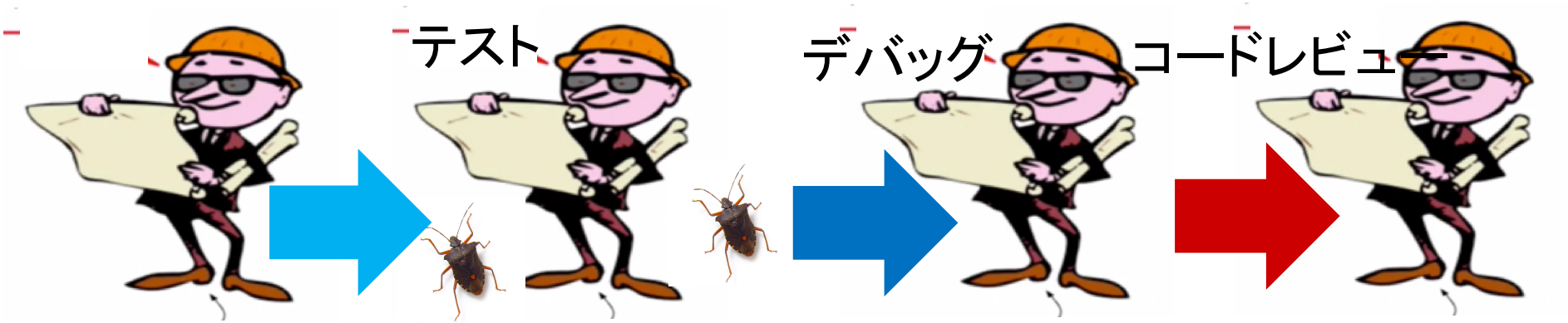


プログラマ　　プログラマ　　プログラマ

コードを見直しをする

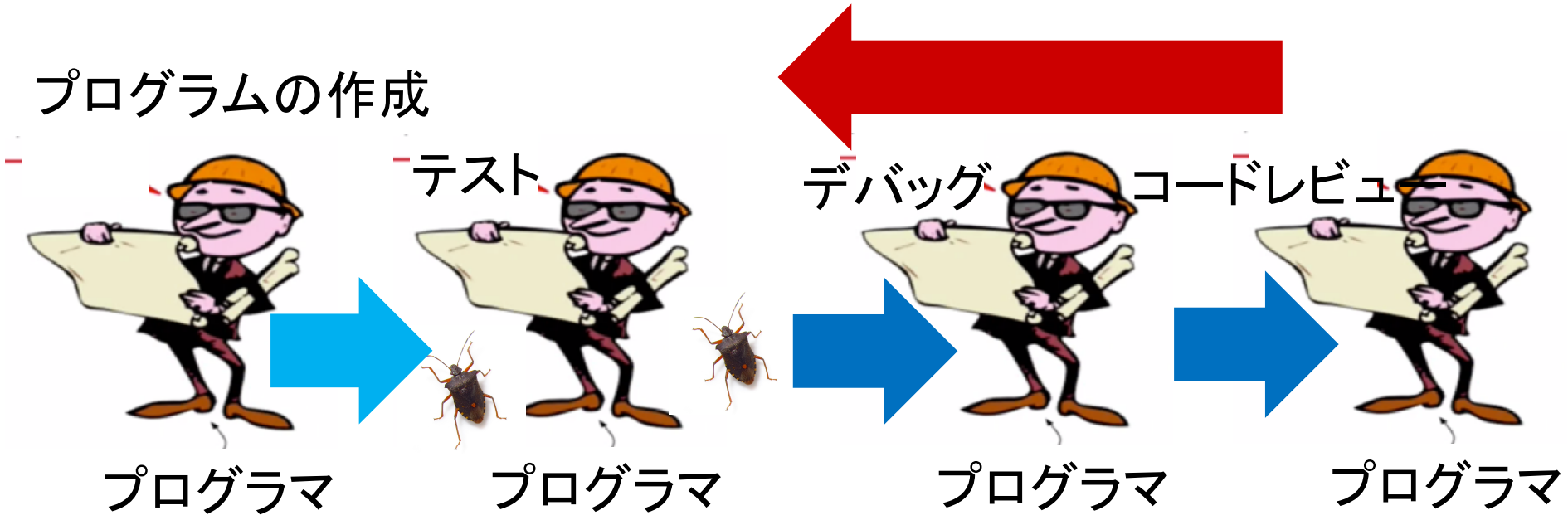
開発

プログラムの作成



コードを見直したら、またテストをする

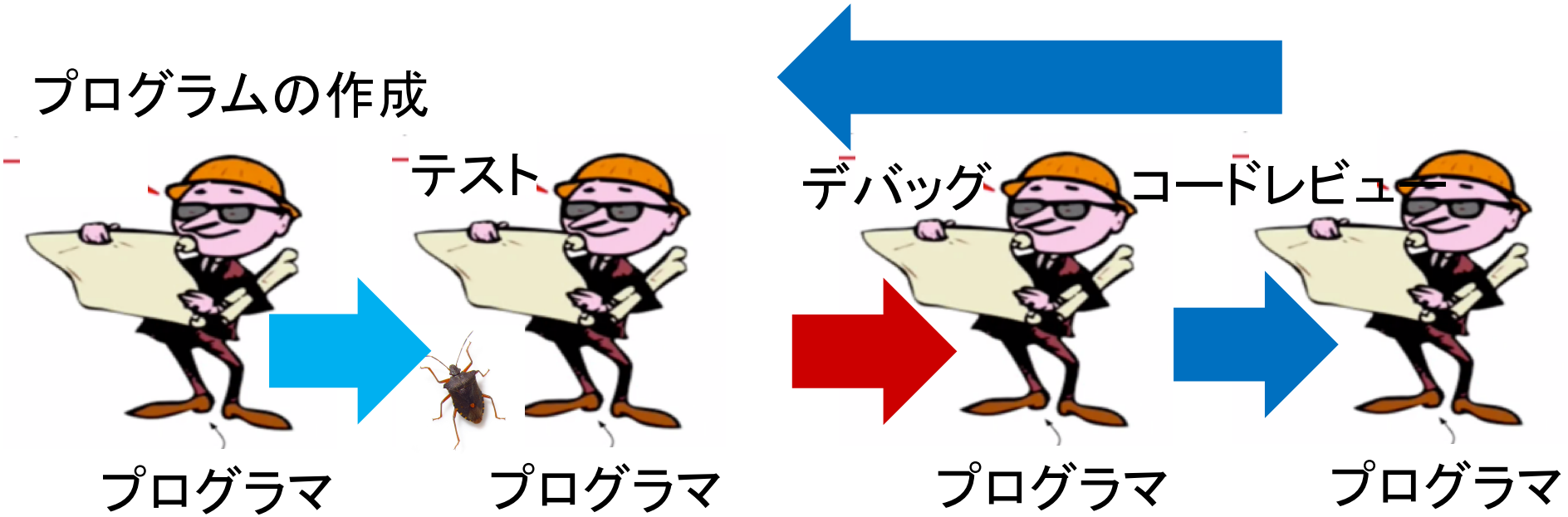
開発



その作業は、バグがなくなるまで繰り返される

開発

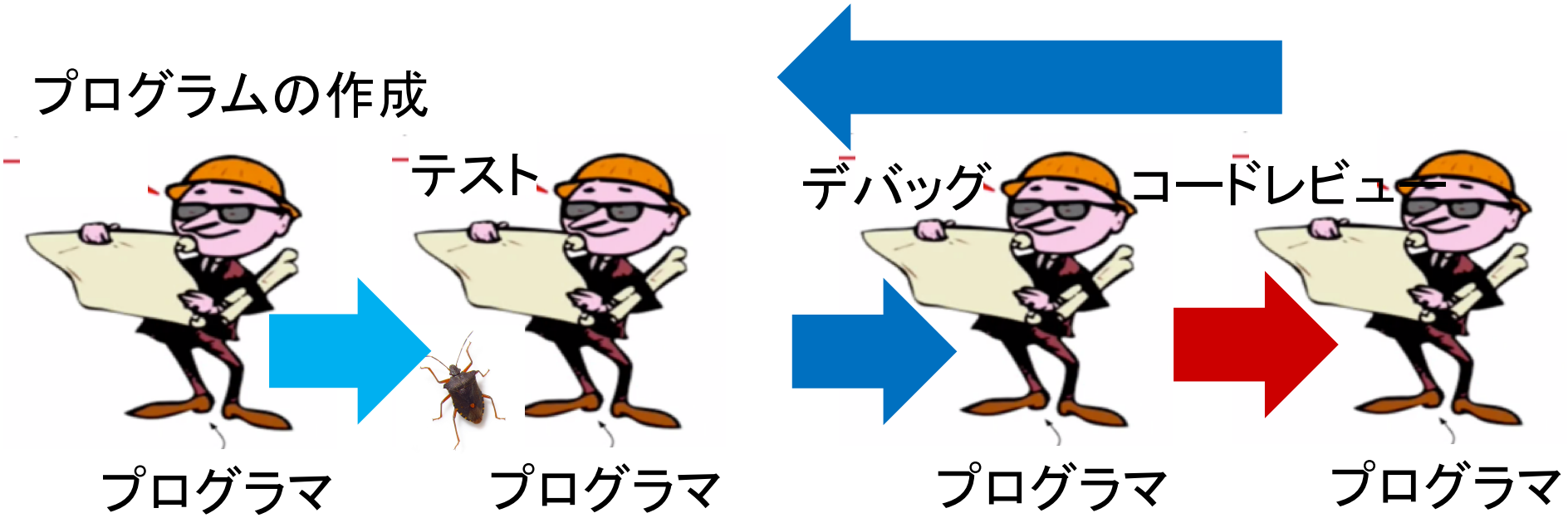
プログラムの作成



その作業は、バグがなくなるまで繰り返される

開発

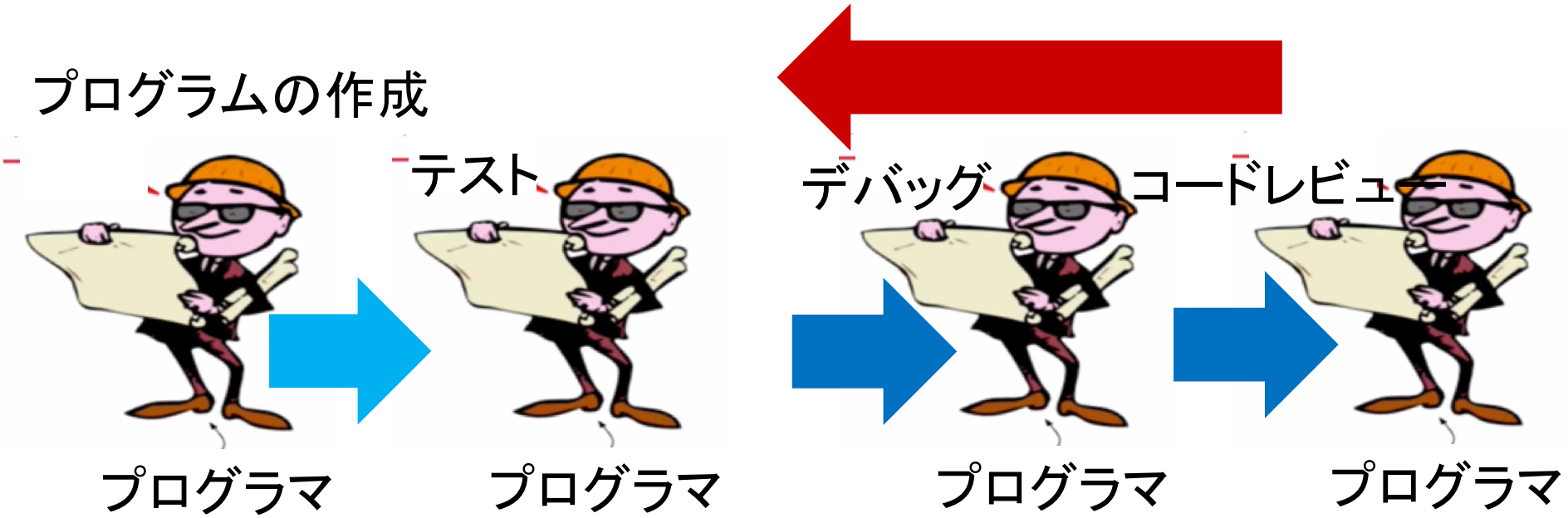
プログラムの作成



その作業は、バグがなくなるまで繰り返される

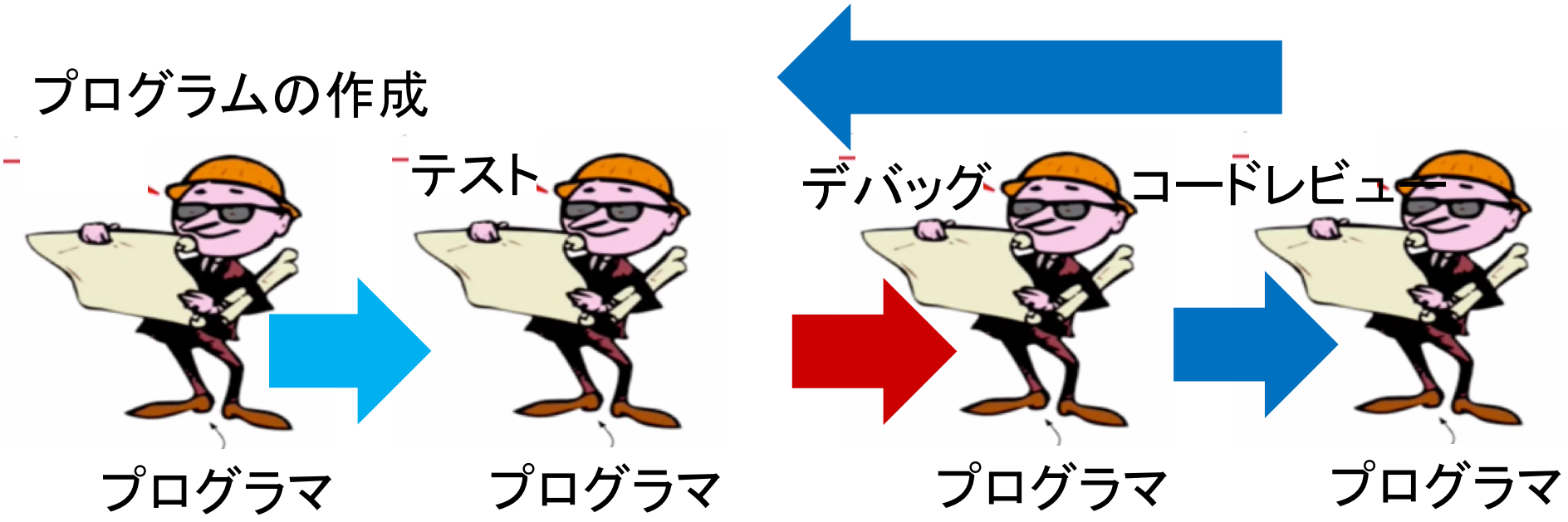
開発

プログラムの作成



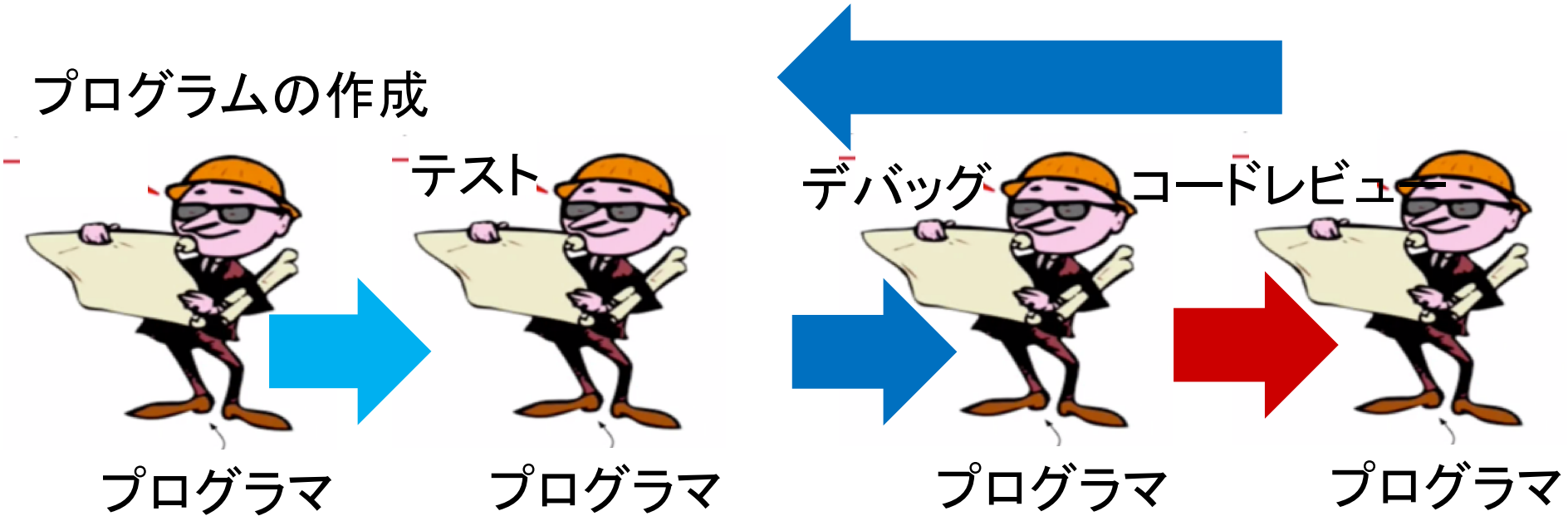
その作業は、バグがなくなるまで繰り返される

開発



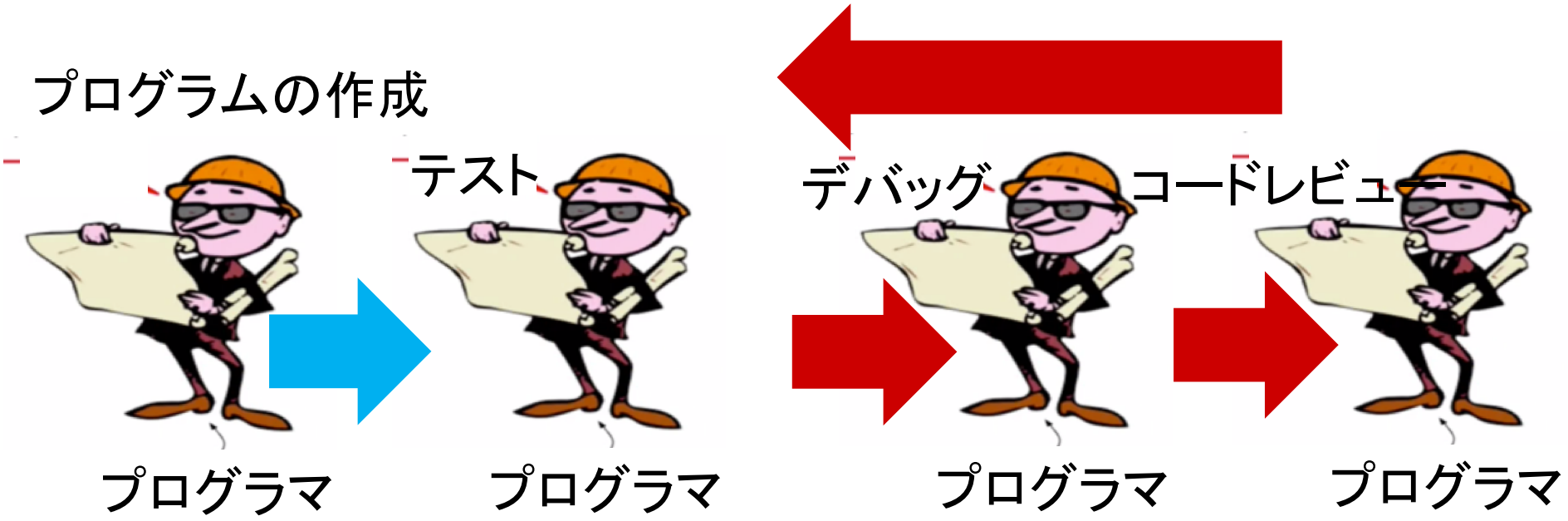
その作業は、バグがなくなるまで繰り返される

開発

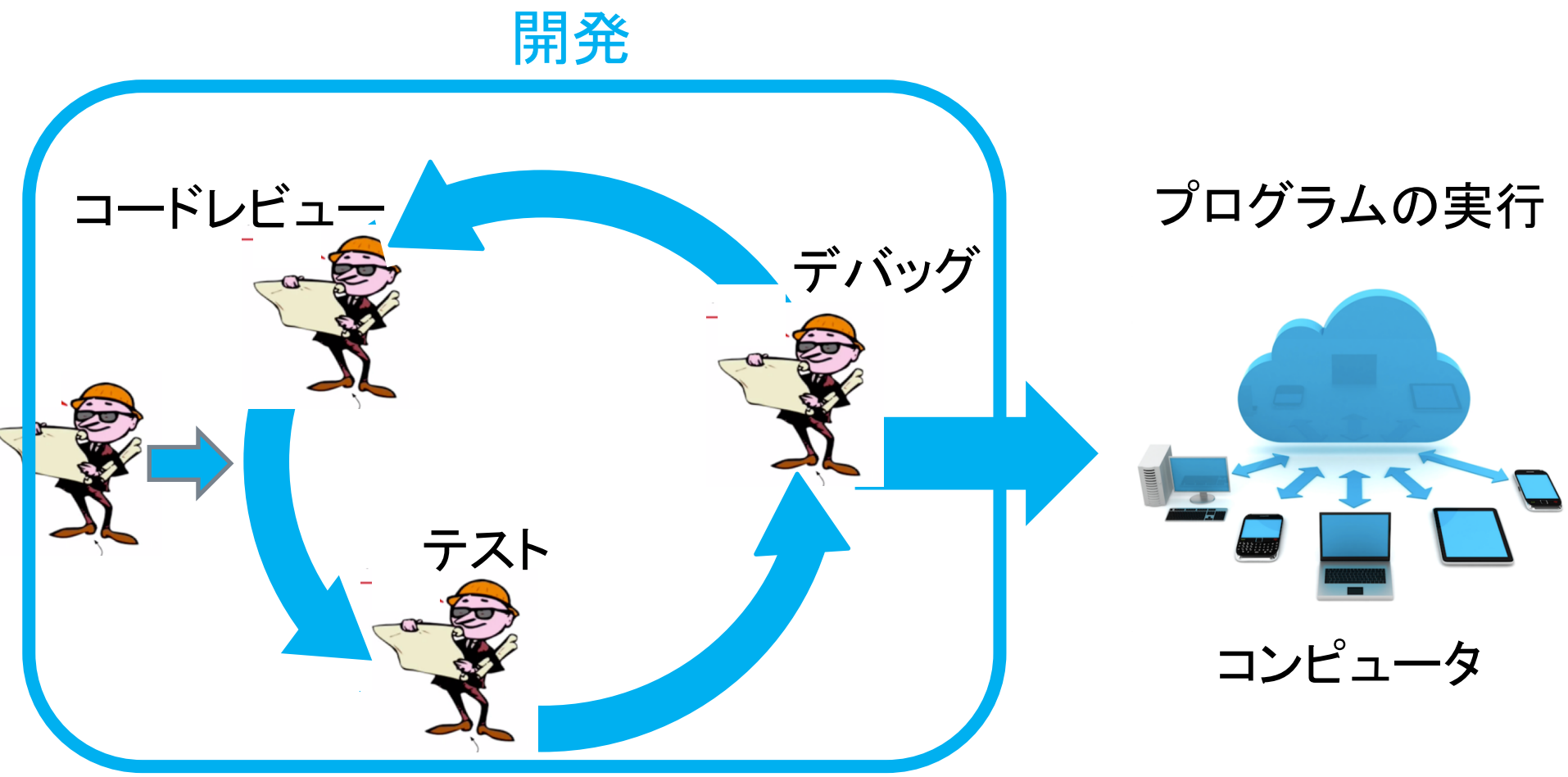


その作業は、バグがなくなるまで繰り返される

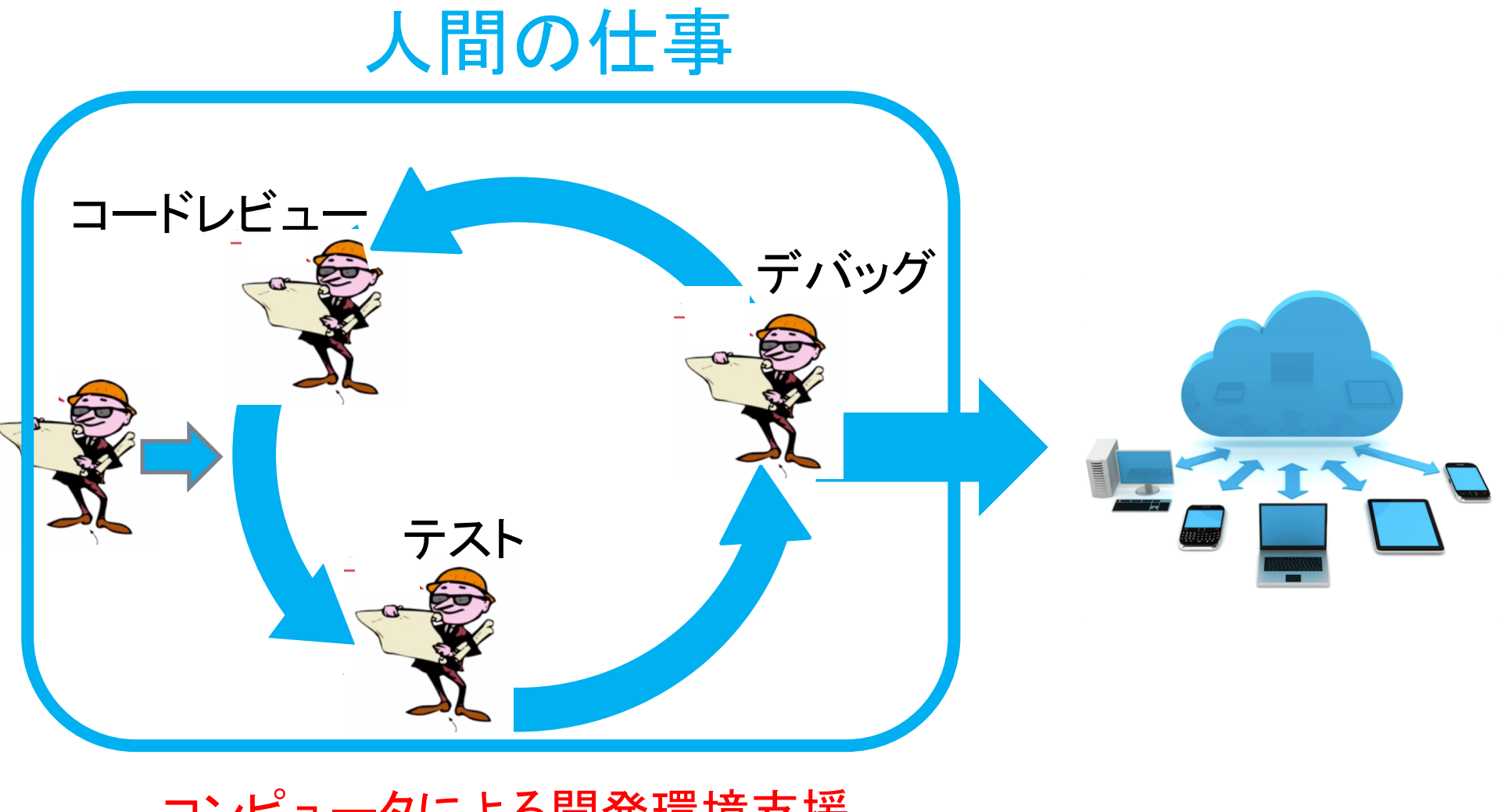
開発



開発では、こうしたテスト・デバッグ・コードレビューのサイクルが繰り返され、その後、コンピュータ上にデプロイされる。

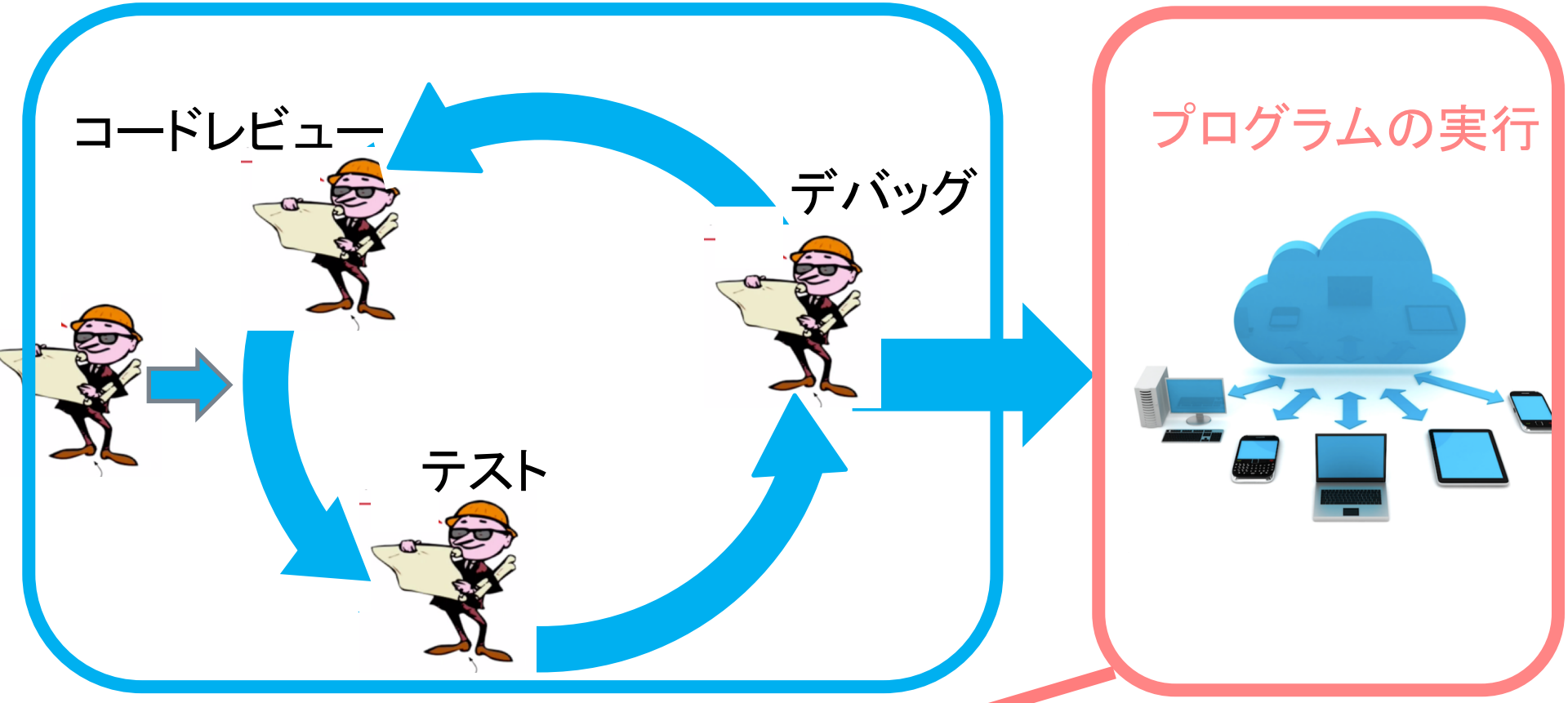


統合エディター等、コンピュータによる開発支援環境は、現代の開発には欠かせないのだが、開発の過程は、基本的には、すべて人間の能力によって担われている。



デプロイされたプログラムは、コンピュータによって実行される。システム・エラーでも起きない限り、この局面では、人間が介在する余地はない。それはコンピュータの仕事である。

コンピュータの仕事



コンピュータによる開発環境支援

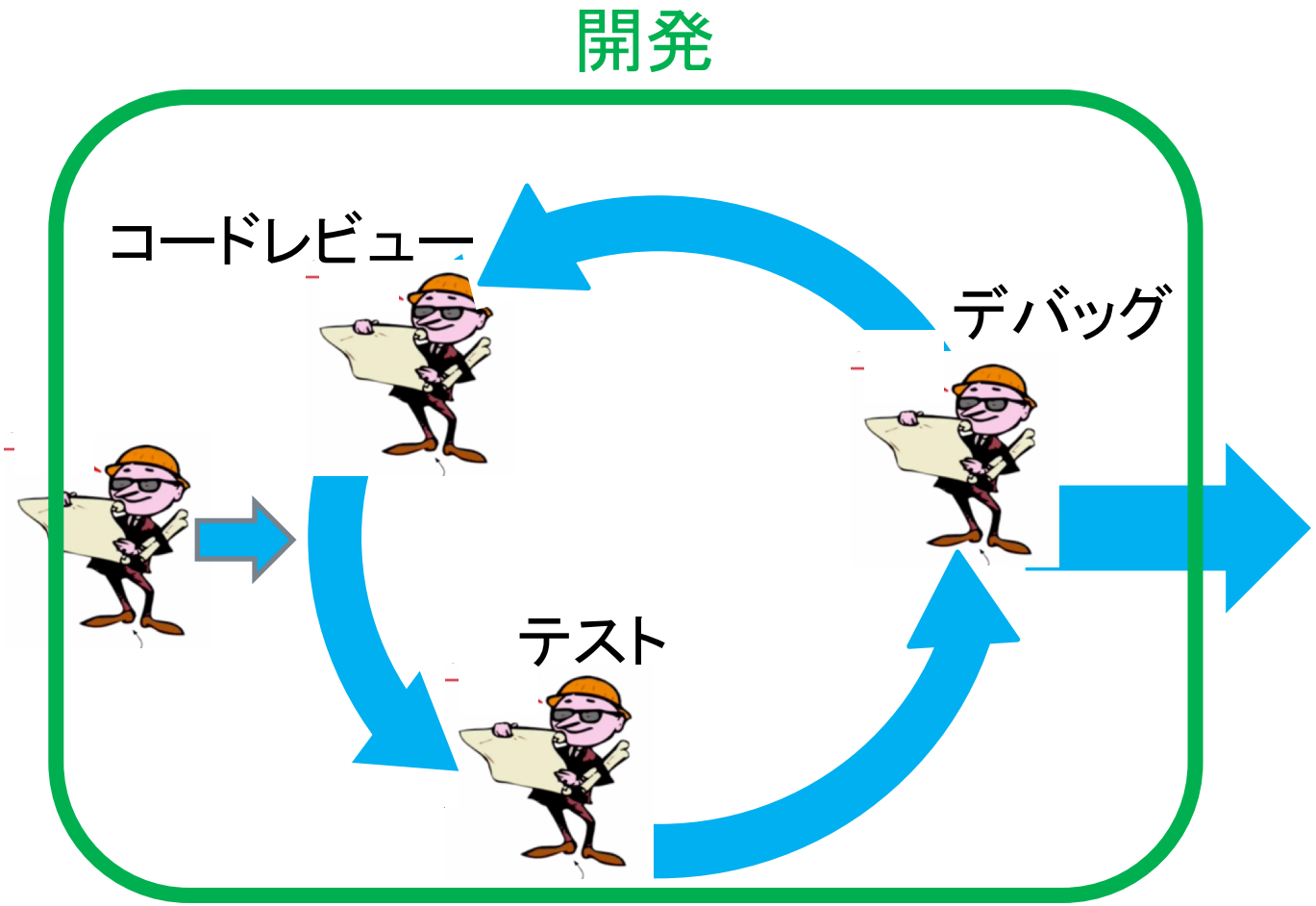
なぜ、開発は人間によって担われているのか？

- それは、ある目的を持ったプログラムを作成する能力が人間にしかないからである。人間は、機械にはない「創造的」な能力を持っている。
- ではなぜ、人間はバグを生み出すのか？
  それは、人間は「創造的」かもしれないが、「完全」ではないからである。
- ただし、人間はバグを修正できる。
  機械は、現在の技術では、バグの修正はできない。

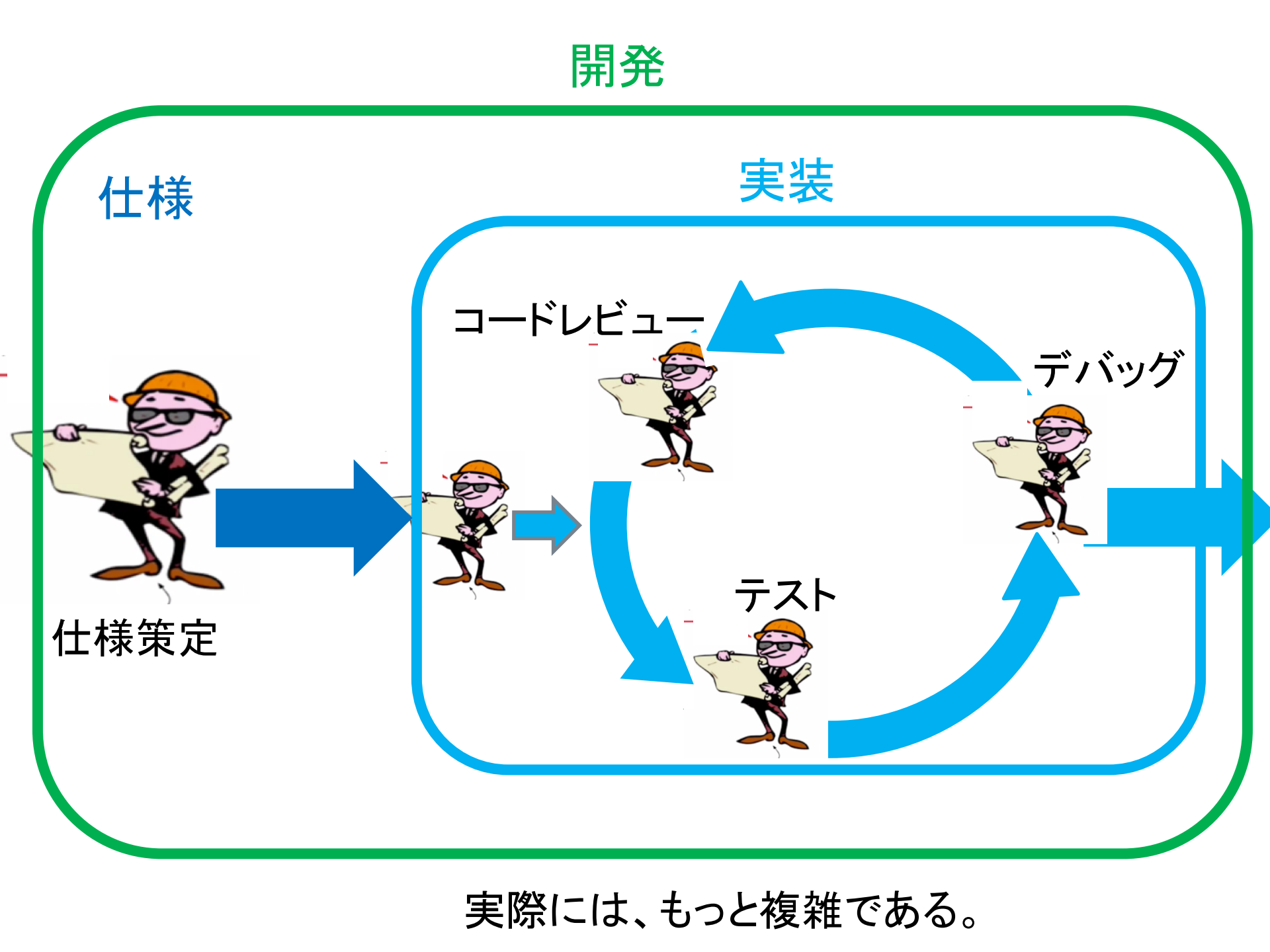
バグを修正する人間の能力はどのようなものか？

- それでは、バグを修正するのに、人間のどんな能力が使われているのだろうか？
- それはプログラムの目的に照らして、その振る舞いが正しいか否かyesかnoで判断し、正しくないなら、その原因を見つけだすという能力である。前者と後者では、多少、難しさは違う。ただ、両者とも、「論理的」に推論する能力と言っていいい。
- そこで問題となっているのは、プログラムの意図を記述する「仕様」と「実装」の関係である。
- 「仕様」の「意味」を理解することが、機械には難しいことが、ここで人間が必要とされる大きな背景になる。

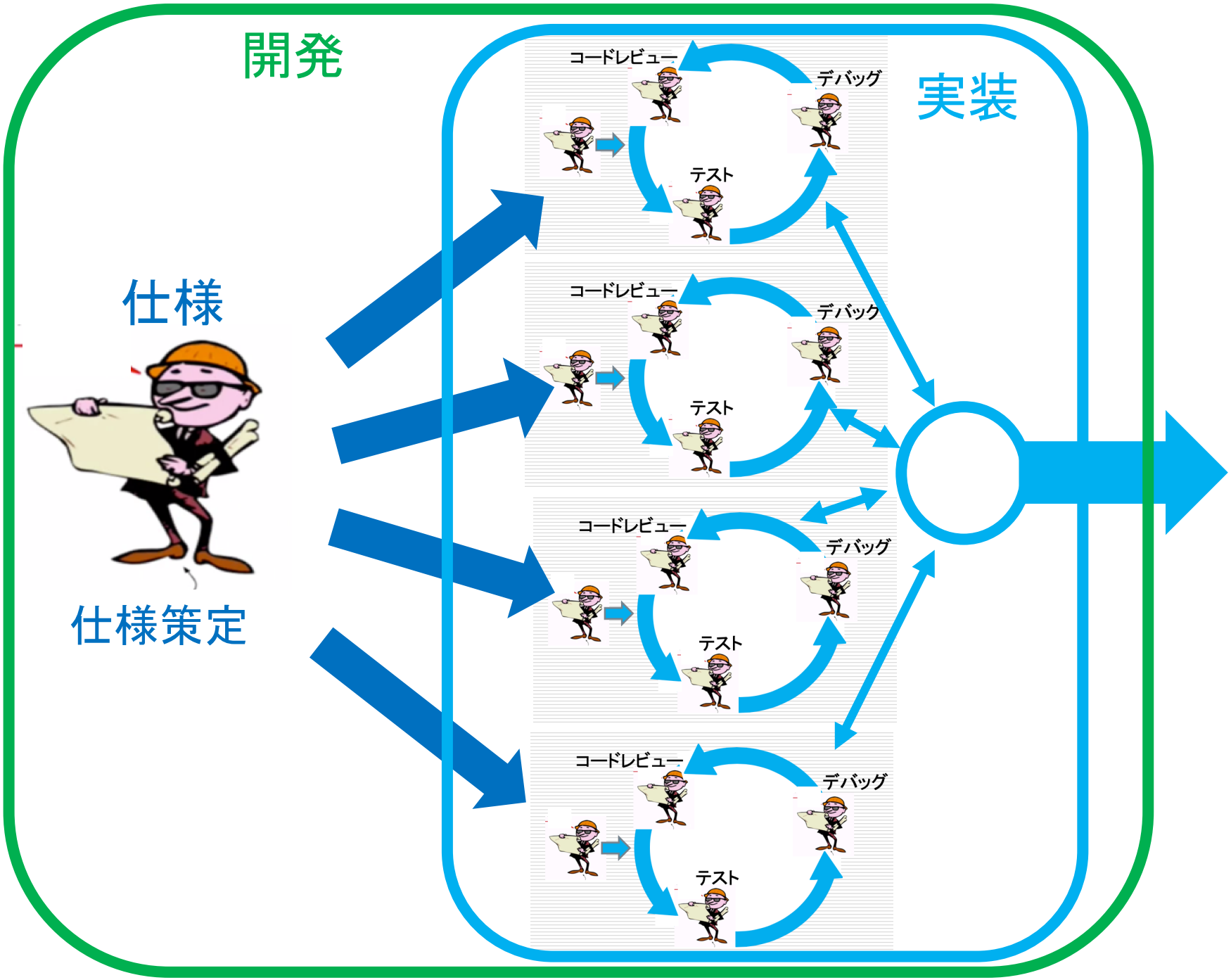
先に見た、次のような開発のモデルには、「仕様」のモメントが抜けている。個人が行う単純な開発にも、それがドキュメントにはなっていなくても、「仕様」にあたるものは必ず存在する。



次のようになる



実際には、もっと複雑である。


開発
仕様
仕様策定
実装
コードレビュー
デバッグ
テスト
コードレビュー
デバック
テスト
コードレビュー
デバッグ
テスト
コードレビュー
デバッグ
テスト

なぜ、開発は人間によって担われているのか？

- 開発は、単に、人間によって担われているというだけではなく、「非常にたくさんの人間」によって担われている。
- それは、そうした作業を、現状の技術では、機械によっては代替できないからである。
- ただ、それは本当だろうか？

コンピュータが論理的推論能力を持てば、開発はどう変わるか？

- 今回のセミナーは、コンピュータが論理的推論能力を持てば、開発がどのように変わるかを考えることをテーマにしている。

- 現在の開発作業の中核をなす「テスト・デバッグ・コードの見直し」というループの繰り返しをドライブしているのは、仕様との関係でのプログラムの見直しである。
- デバッグ作業でプログラマは、プログラムとにらめっこしているように見えるのだが、その時、無意識的にでも（「仕様」はドキュメントに書かれたものだけではない。「暗黙の常識」も「仕様」として機能する）、意識的にでも、常に意識しているのは、仕様とプログラムの関係である。

「仕様」と「実装」の関係は、論理的である

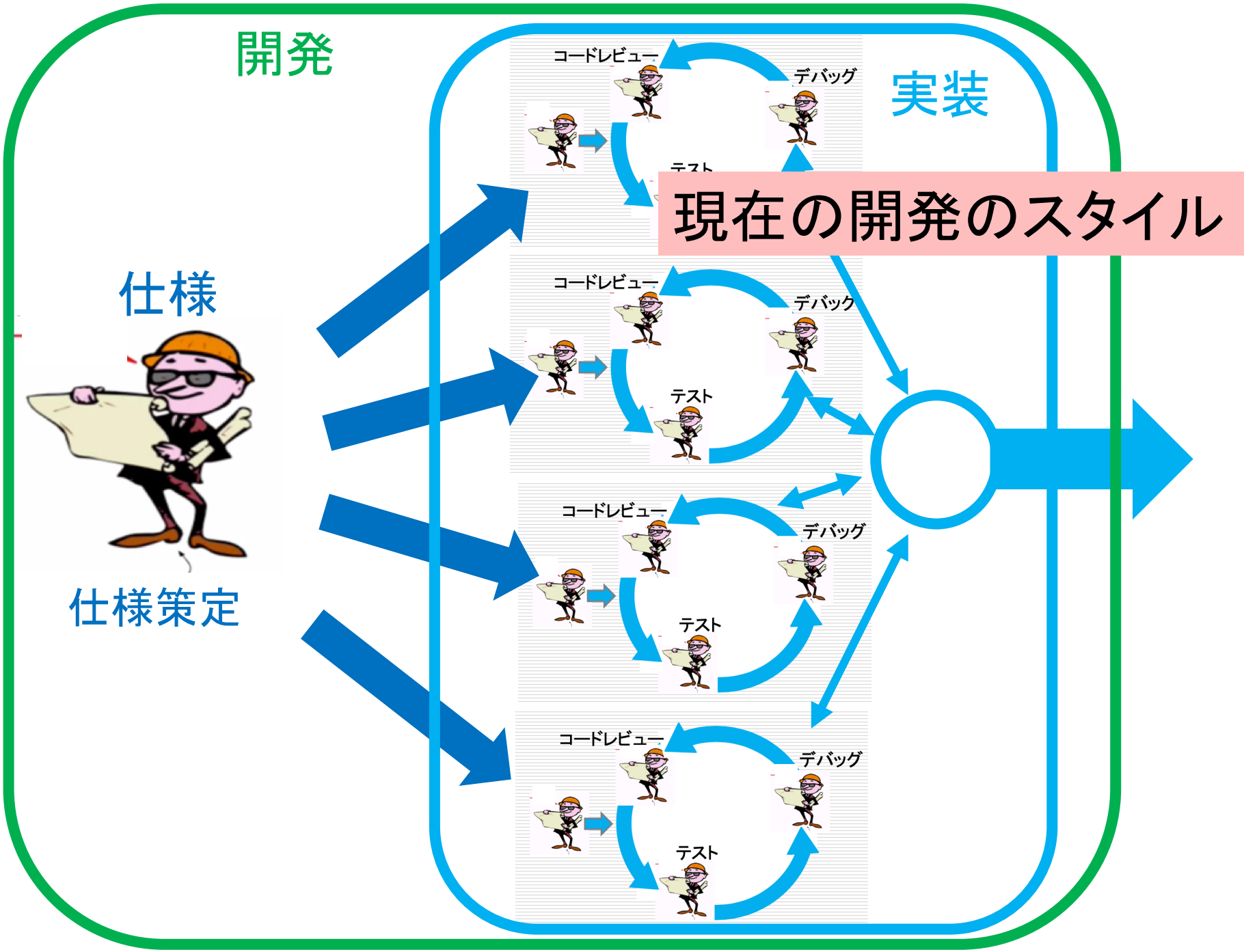
- ここでいう「仕様」は、あれこれのドキュメント化された「仕様」だけを意味するわけではない。コンピュータによって、実行さるべきものと人間が「意図」するものは、すべて「仕様」と見做して構わない。

- 重要なことは、このレベルでの「仕様」と実際の「実装」の関係は、論理的なものであるということである。
- もし、仕様が形式的に定義されているなら、ある実装がその形式的仕様を満たしているか否かは、形式的に証明可能である。
- もし、機械が論理的推論能力を持てば、そうした証明の実行によって、仕様と実装の一致をチェックするテスト・デバッグ作業に、人間は不要になる。

「実装」の正しさの鍵は、「仕様」が握っている

- 明らかに、「実装」が正しいか否かの判断の鍵は、「仕様」が握っている。
- ただ、「実装」が、どの実装でもコンピュータで受理されるプログラム言語の形式をとっているのに対して、「仕様」の記述のレベルは、まちまちである。仕様の策定者は、一般には、実装コードを書かない。
- もし、コンピュータが論理的推論を実行する能力を持ち、仕様と実装の一致を形式的に証明できるなら、ほとんど同じ技術を使って、形式的仕様から実装のプログラムが自動生成できる。

- コンピュータによる、仕様からのプログラムの自動生成は、開発の世界を大きく変える。


開発
実装
現在の開発のスタイル
仕様
仕様策定
コードレビュー
デバッグ
テスト
コードレビュー
デバック
テスト
コードレビュー
デバッグ
テスト
コードレビュー
デバッグ
テスト

開発

仕様

実装

将来可能な開発のスタイル

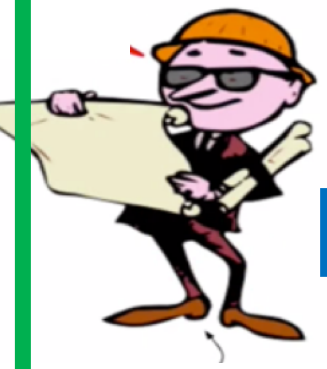
仕様策定

コンピュータによる
仕様からのプログラムの
自動生成

人間の仕事

コンピュータの仕事の拡大

仕様



仕様策定

実装

コンピュータによる
仕様からの
プログラムの
自動生成

プログラムの実行

コンピュータ

コンピュータによる仕様開発環境

開発での人間の仕事はなくなるのか？

- なくならない。

- プログラムを作成するのに、人間の「創造性」が必要だったように、形式的な仕様を作成するのには、人間の「創造的」な力がいる。
- ただ、人間による誤りの混入を防ぐために、仕様作成の各ステップでの「論理性チェック」のサポートが必要になる。それは、現在のGUIを備えた統合開発環境と同じようなものだ。コンピュータが、仕様作成を支援する。

こうした見通しは実現可能か？

- 現在では、まだ、こうした環境は部分的にしか実現されていない。こうした見通しが、開発全体に広がる可能性はあるのであろうか？
- 十分にある。

1. セキュアで堅牢なソフトウェアを提供する上で、人間から構成された開発体制の構造的複雑さは、バグの温床になる。抜本的対応がいずれ求められることになる。
2. 開発体制の複雑さは、開発期間と開発コストの増大に結びついている。開発の多くの部分の自動化には、経済合理性がある。
3. こうした開発を可能にする「証明支援システム」が登場し、その利用も大きく拡大している。

機械は論理的＝数学的推論能力を持つ

理論的背景

「証明」＝「プログラム」＝「計算」

機械の論理的＝数学的推論能力

ここでは、機械が論理的＝数学的推論能力を持つという認識がどのように形成されてきたのかを、歴史的・理論的に振り返りたいと思う。

こうした振り返りは、なぜこうした認識が今日まで十分に確立されなかったのか、その理由も明らかにする。

「証明」＝「プログラム」＝「計算」

「計算」「証明」「プログラム」という三つの言葉は、もともと別の意味を持っている。それは今でも同じだ。そして長い歴史を持っている。

この三つの言葉の中で、「計算」の起源が一番古い。4000年以上前の古代バビロニアでは、2の平方根の値も知っていたし、ピタゴラスの定理で直角三角形の斜辺の長さを計算する「計算表」も持っていた。

「証明」の概念の起源は、2300年前のユークリッドに帰してもいいと思う。

この中で、「プログラム」という項が一番新しい。その起源は、コンピュータが登場した1950年代を遡ることはないのは明らかだ。

バビロニアの数学　粘土板 YBC 7289

バビロニアでは、2の平方根の値を知っていた。

その近似値は60進法で4桁、
10進法では約6桁に相当する。
対角線に刻まれた数字は、
1, 24, 51, 10。60進では、

$$1 + 24/60 + 51/60^2 + 10/60^3 = 1.41421296...$$

となる。

$$\sqrt{2} = 1.41421296...$$

バビロニアの数学　Plimpton 322

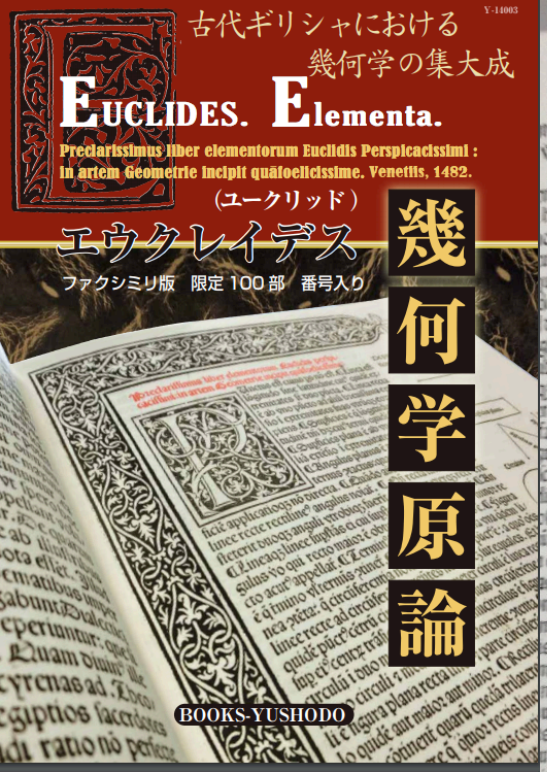
バビロニアでは、ピタゴラスの定理を知っていた。

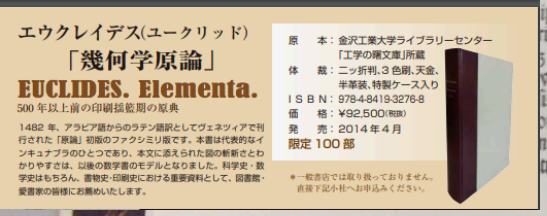
計算

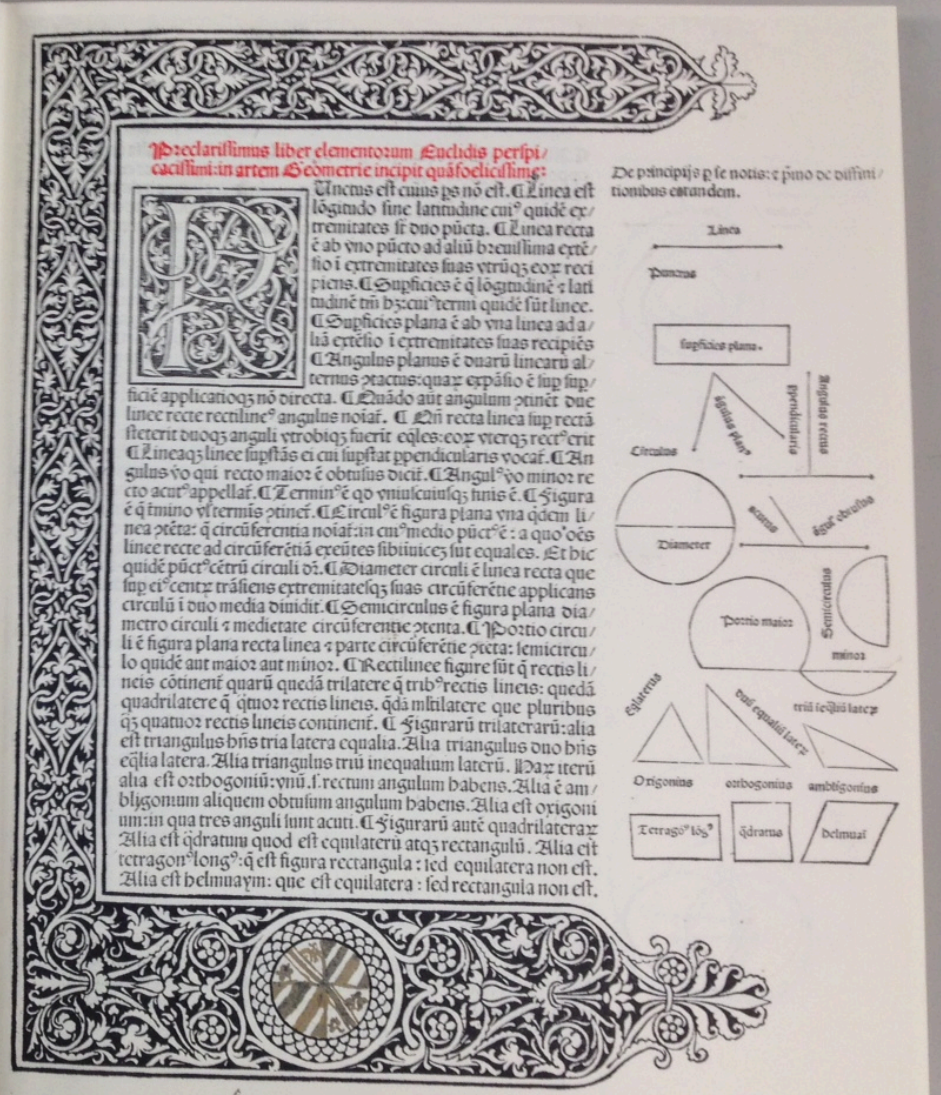
1822 BC ~ 1762 BC

証明

ラファエロの壁画「アテナイの学堂」から



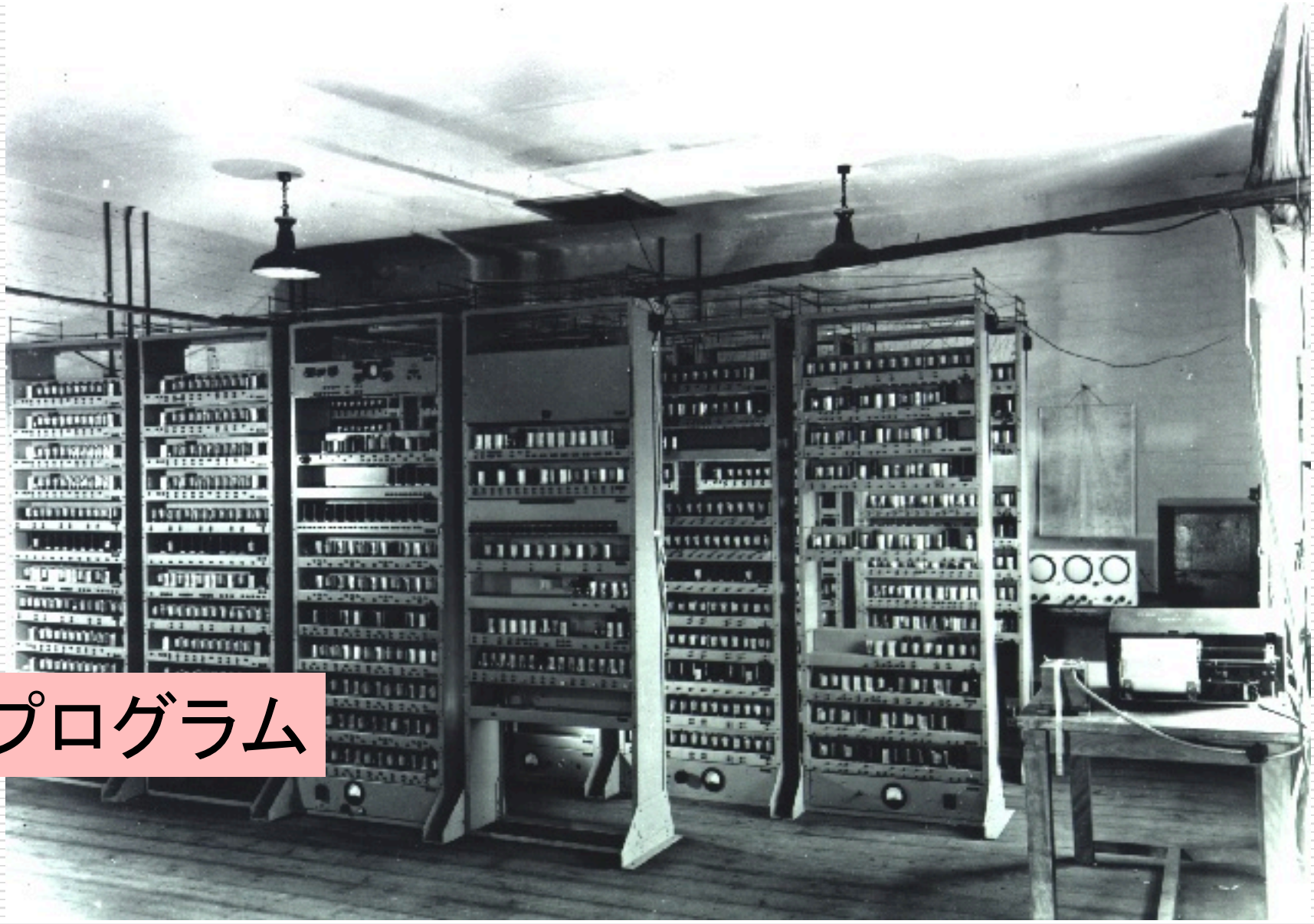




証明

1482 年、アラビア語からのラテン語訳としてヴェネツィアで刊行された「原論」初版のファクシミリ版

EDSAC (Electronic delay storage automatic calculator) 1949



プログラム

amazon web services™

Microsoft Azure

Kubernetes

κυβερνήτης: *Greek for "pilot" or "helmsman of a ship"*
the open source cluster manager from Google

プログラム

「証明」＝「プログラム」＝「計算」

ここでは、機械の能力を人間がどう認識してきたかを、この三つの概念の関係の認識の歴史をメイン・ストーリーにして述べたものである。

概略を述べておこう。

まず、「計算」と「証明」が、本質的には同じものであるという認識が生まれる。1930年代のことだ。

それから40年ほど経って、「プログラム」は「証明」とみなせるという認識が生まれる。1970年代のことだ。

理論的には、この二つの発見で、三項の関係は明確になった。「コンピュータは、プログラムの実行という形で、証明を計算する」のである。

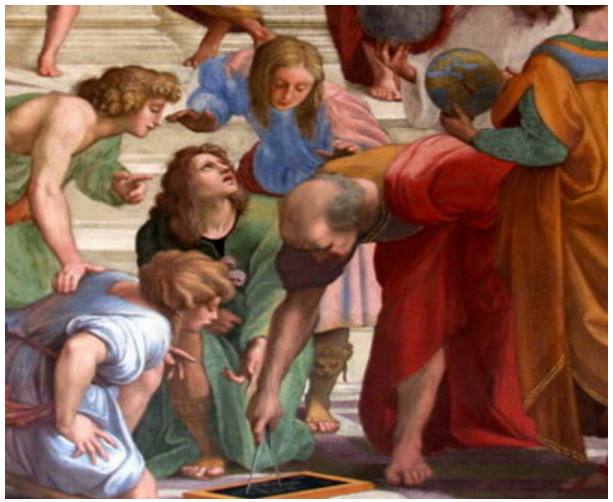
プログラム

1970年代

コンピュータは、
プログラムの
実行という形で、
証明を計算する

計算

証明



1930年代

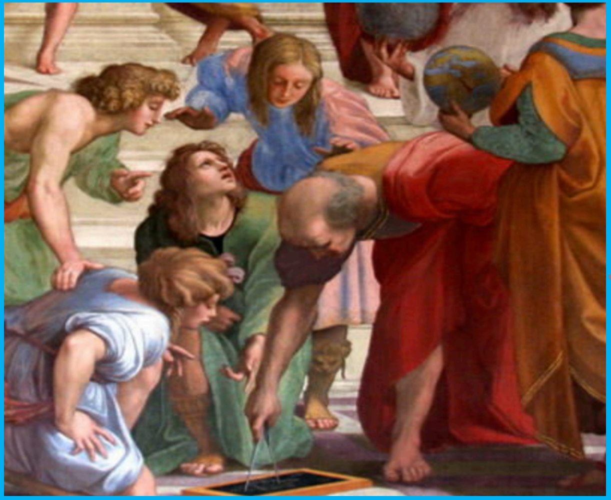
「証明」＝「プログラム」＝「計算」

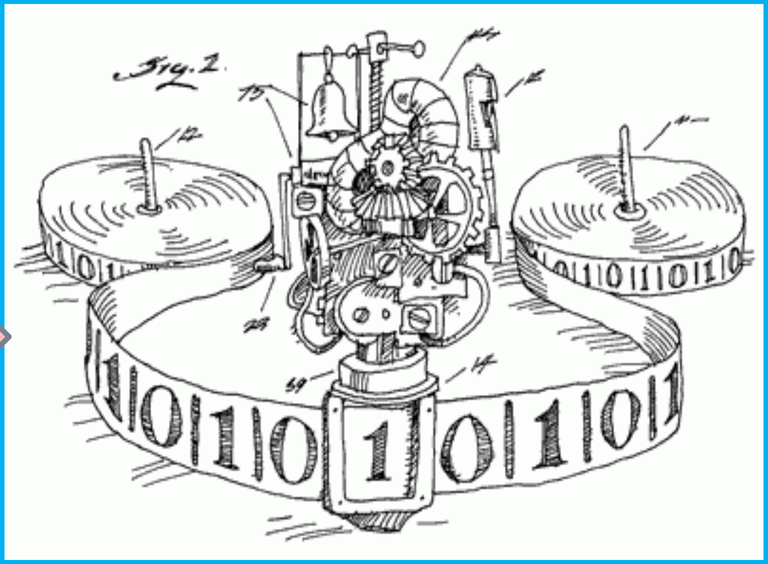
- 理論的には、現在は、三つ目の大きな転換点を迎えている。
- 数学とその証明をコンピュータ・プログラムとして記述しようというVoevodskyのUniMath, プログラムの正しさを数学的に証明しようというChlipalaらのDeep Specificationの動きが、それである。
- 機械が論理的＝数学的推論能力を持つという認識は、現実を動かす力になろうとしていると僕は考えている。

「証明」＝「計算」

計算

証明





「証明」＝「計算」

「証明」＝「計算」の認識の突破口は、今から90年前、ゲーデルの有名な「不完全性定理」によって切り開かれた。

1930年　ゲーデルの不完全性定理

ゲーデルの不完全性定理とは、次のような定理である。

ある形式的体系Sを作ったとしよう。その体系では、初等数論を定義でき、矛盾がないとする。その時、Sに属する命題Gで、Sの中では証明も反証もできないような命題Gが存在する。

Fを、初等数論を含む、無矛盾の形式的体系としよう。この時、この形式的体系Fの無矛盾性は、この体系Fの中では証明出来ない。

「計算とはなにか？」「証明とは何か？」

歴史的に見ると、不完全性定理は、証明あるいは計算の性質についての探求の中で生まれた「最後の結論」ではないのだ。事実は、その逆である。

1930年代に、この分野は、不完全性定理に強烈な刺激を受けて、「証明とは何か？」「計算とは何か？」という基本的な「問い」に突き進むことで発展する。

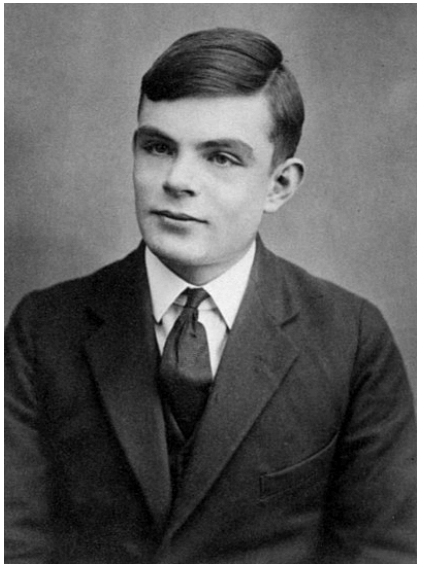
ゲーデルの結果を受けて、「計算可能性」「証明可能性」についての探求が一斉に始まる。

この時代の白眉は、「計算可能性」についての見かけは全く異なるこれらのアプローチが（もちろん、それは、ゲーデルの結果を再現できるものだ）、次々と、「同値」であることが証明されていくことだ！

様々な「計算可能性」へのアプローチと
その同値性の認識

Kurt Gödel
1906-1978

Alonzo Church
1903-1995



Alan Turing
1912-1954

帰納関数論

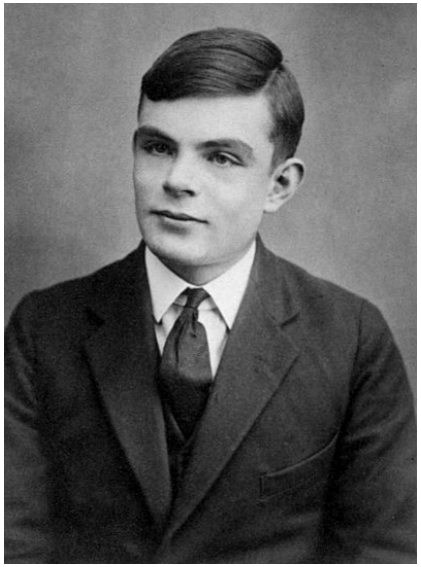
ラムダ計算

チューリング
マシン

様々な「計算可能性」へのアプローチと
その同値性の認識

Kurt Gödel
1906-1978

Alonzo Church
1903-1995



Alan Turing
1912-1954

帰納関数論 ＝ ラムダ計算 ＝ チューリングマシン

チャーチのラムダ計算

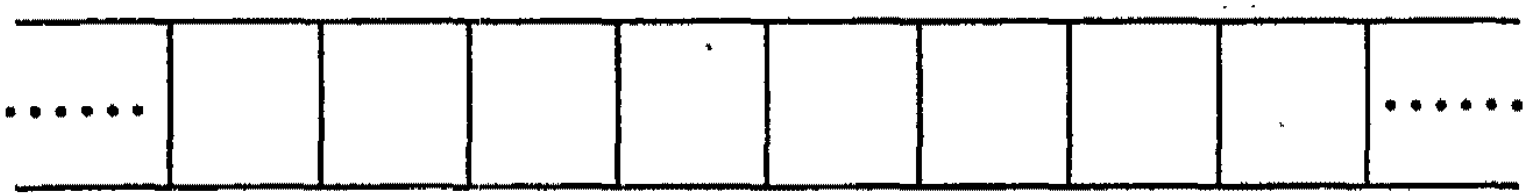
型のないラムダ計算

1. 変数x,...はλ式である。
2. tがλ式で、xが変数であるなら、λxによる抽象化λx.t はλ式である。(abstraction)
3. t, sがλ式であるなら、tへのsの適用 t s は、λ式である。(application)

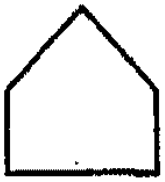
型付きラムダ計算

1. 単純な変数 v_i は型を持つ。 $v_i : \tau$
2. $x : \sigma$ で $e : \tau$ なら、$(\lambda x_\sigma.e) : (\sigma \rightarrow \tau)$
3. $e_1 : (\sigma \rightarrow \tau)$で、$e_2 : \sigma$ なら、$(e_1\ e_2) : \tau$

チューリング・マシン



テープ（マスで区切られている）



ヘッド

テープ：ひとマスずつに区切られており、左右に移動する。長い。

ヘッド：テープのひとマスの記号を読み取り、マシンの「状態」に応じて、次の動作をする。

可能な動作：

1. マス目に記号を書き込む（消す＋書く）。あるいは、そのままにして何も書き込まない。
2. マシンの状態を変える。あるいは状態を同じに保つ。
3. ヘッドを、右あるいは左に移動する。（テープが動く）

チャーチ＝チューリングのテーゼ

1940年代には、今日、「チャーチ＝チューリングのテーゼ」と呼ばれる「計算可能性」についての認識は、確立されることになる。（これが、「複雑性理論」の第一世代である。）

「チャーチ＝チューリングのテーゼ」というのは、次のような提案である。

Every effectively calculable function (effectively decidable predicate) is general recursive

すべての実効的に計算可能な関数（実効的に決定可能な述語）は、一般帰納的である。

「証明可能性」と「計算可能性」の同一性の認識

ゲーデルの不完全性定理から、チャーチ＝チューリング・テーゼへの流れは、今から 90年前のこの時期に、「証明可能性」と「計算可能性」の同一性の認識は、明確に認識されていたことを示す。

しかし、コンピュータは、まだ存在しない！

プログラム

証明

「証明」＝「プログラム」

「カリー・ハワード対応」の発見と「従属型理論」の成立

ゲーデルの発見から40年後の1970年代、論理学・数学の分野で、重要な動きがあった。ハワードの「カリー・ハワード対応」の（再）発見と、それに基づくマーティン・レフの「従属型理論」の成立である。

Curry-Howard対応

- 1940年代のChurchの仕事から、ラムダ計算に対する関心が、ふたたび活発化するのは、20年近くたった1960年代からである。
- 理由ははっきりしている。コンピュータが現実に動き出したからである。
- この時期の代表的な成果は、HowardのCurry-Howard対応の研究である。

Curryの発見
型付きラムダ計算と直観主義論理との対応

- 既に1934年に、Curryは、型付きラムダ計算と直観主義論理とのあいだに、対応関係があることを発見していた。

- 型を持つラムダ計算の関数の型を示す矢印 → と、論理式“A → B” の中に出てくる含意を意味する矢印 → との間に、対応関係があるというのだ。

Curry, Haskell (1934), "Functionality in Combinatory Logic”
http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1076489/pdf/pnas01751-0022.pdf

カリーが気づいたこと
次の二つのルールの形が似ているということ

論理式の推論ルール

$$\frac{A \to B \qquad A}{B}$$

A → B が成り立ち、
Aが成り立つなら
Bが成り立つ

関数の適用の型のルール

$$\frac{f\colon A \to B \qquad a\colon A}{f\ a\colon B}$$

f が A → B という型を持ち、
a がAという型を持つなら
f a は型 B を持つ

論理式の含意を意味する矢印 → と関数の型を示す矢印 →の間に、対応関係があるということ。

ハワードとマーティン・レフが、考えたこと

論理式 A → B が、A → B という関数と同じ型を持つと考えられるとしたら、他の論理式も、型を持つと考えることができるのでは？

例えば、論理式 A ∨ B は、型 A ∨ B を、論理式 A ∧ B は、型 A ∧ B を持つと考えることはできないか？

そして、この論理式（命題：Proposition）を型とみなすアイデアは、うまく機能するのである。こうした考えを、

Proposition as Type

と呼ぶ。

命題Pと証明p ⇨ p : P

彼らは、この時、型Pの項pにあたるものを、命題Pの証明と考えることができることに気づく。それは、命題の証明が何から構成されるかについて、次のような自然な解釈を与える。

- **p: A ∧ B**　　pは、Aの証明とBの証明の両方
- **p: A ∨ B**　　pは、Aの証明、あるいは、Bの証明
- **p: A → B**　　pは、Aの証明からBの証明を導く方法
- **p: (∀x)B(x)**　　pは、任意のaに対してB(a)の証明を与える方法
- **p: (∃x)B(x)**　　pは、あるaに対するB(a)の証明

こうした考えを、

Proof as Term

と呼ぶ。

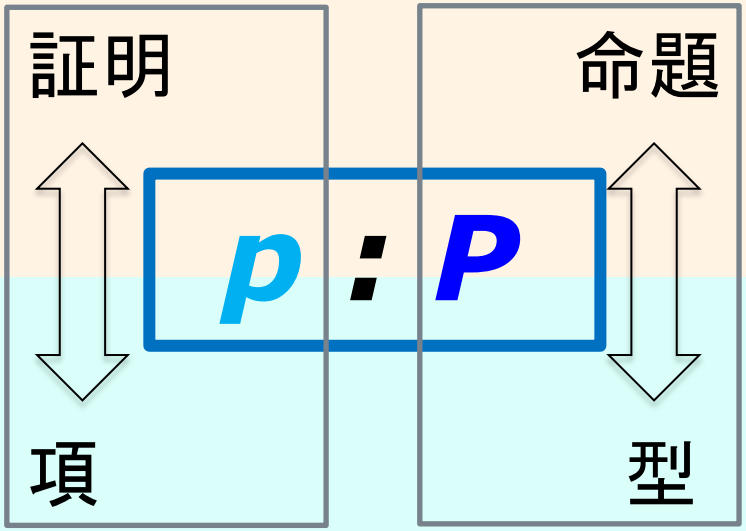
こうした、「型」と「命題」、「項」と「証明」との対応を

Curry-Howard対応 という

「*p* は、命題 ***P*** の証明である」

Proof as Term

証明は
項である



Proposition as Type

命題は
型である

「*p* は、型 ***P*** の項である」

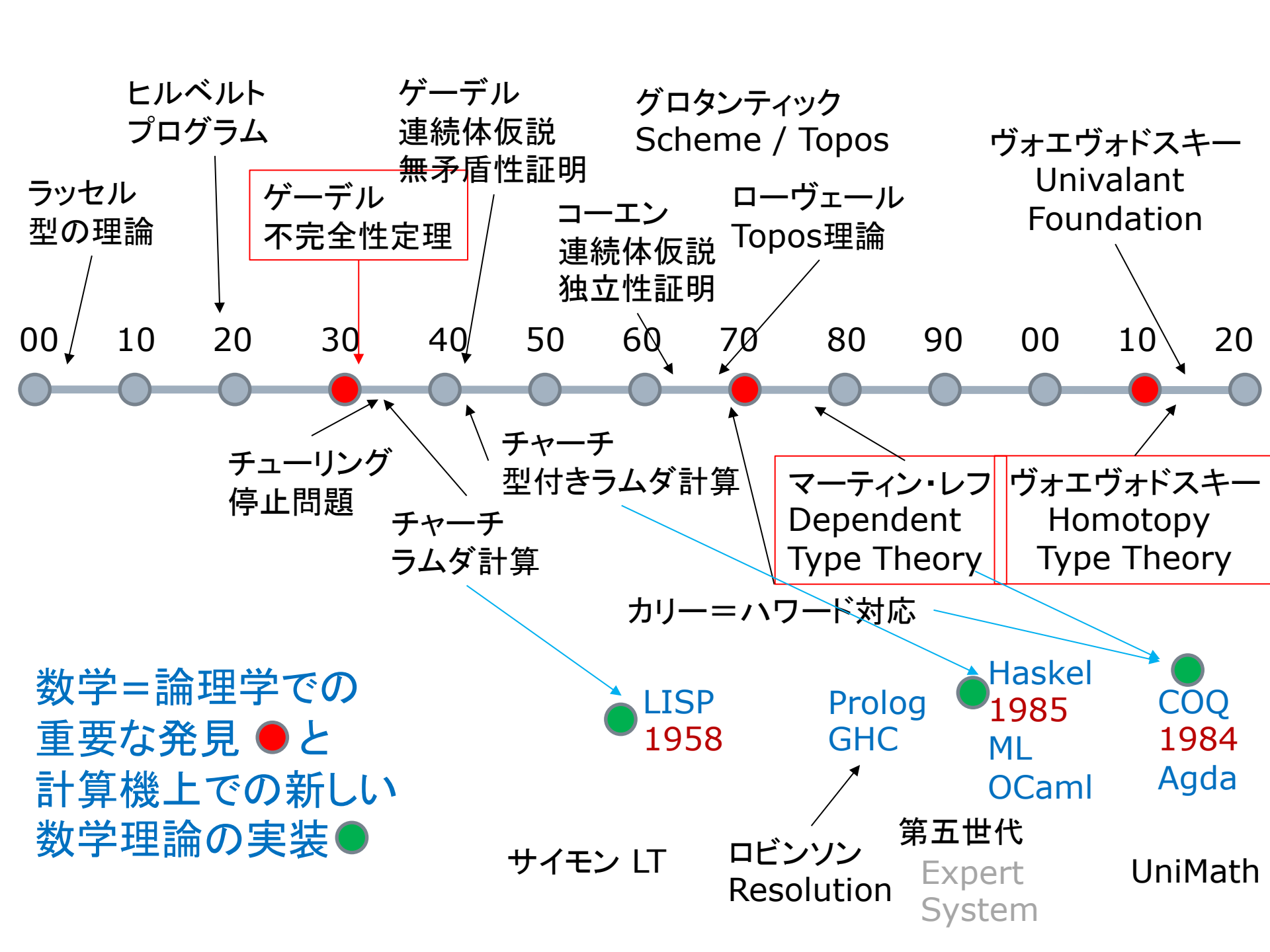
「証明」＝「プログラム」

重要なことは、このCurry-Howard対応は、項の計算をプログラムと見なせば、”Proof as Program”としても解釈出来るということである。

いまから 40年近く前に、論理学者と計算機学者の一部は、「証明」＝「プログラム」という認識にたどり着いていた。

タイムラグ

- 40年に一度の大発見でも、それが他の分野に伝わるには、10年単位の時間がかかることがある。論理学・数学の世界での新しい認識がコンピュータの世界に広い影響を与えるまでには、一定の時間がかかる。
- 30年代のチャーチの「型なしラムダ計算」は、LISP (1958)の理論的基礎。同じチャーチの「型付きラムダ計算」を関数型言語Haskell(1985)は基礎にしている。「従属型理論」に基づくCoqの開発が始まったのは、1984年である。
- ちなみに、マーティン・レフの理論から40年近く経ってからHomotopy Type Theoryで論理学の世界を一新した数学者のVoevdskyは、たまたまコンピュータ実習室で学生の演習を見ていて、「従属型理論」に出会ったという、


ヒルベルト
プログラム
ゲーデル
連続体仮説
無矛盾性証明
グロタンティック
Scheme / Topos
ヴォエヴォドスキー
Univalant
Foundation
ラッセル
型の理論
ゲーデル
不完全性定理
コーエン
連続体仮説
独立性証明
ローヴェール
Topos理論
00
10
20
30
40
50
60
70
80
90
00
10
20
チューリング
停止問題
チャーチ
型付きラムダ計算
チャーチ
ラムダ計算
マーティン・レフ
Dependent
Type Theory
ヴォエヴォドスキー
Homotopy
Type Theory
カリー＝ハワード対応
数学＝論理学での
重要な発見 と
計算機上での新しい
数学理論の実装
LISP
1958
Prolog
GHC
Haskel
1985
ML
OCaml
COQ
1984
Agda
サイモン LT
ロビンソン
Resolution
第五世代
Expert
System
UniMath

プログラム

計算

「プログラム」＝「計算」

「プログラム」＝「計算」

- 「証明」＝「プログラム」＝「計算」というテーゼの中では、「プログラム」＝「計算」というのが、一番わかりやすいように思う。IT技術者なら、「プログラム」＝「アルゴリズム」＝「計算」と考えればいいと思う。
- 「プログラム」という概念自体、20世紀の中頃に生まれたものであるので、「プログラム」＝「計算」 というテーゼは、三つのテーゼの中では一番新しいものである。
- この認識は、コンピュータの普及と、ITビジネスがコンシューマ中心にシフトして成功する中で、多くの人の中で自然に拡大した。

「プログラム」＝「計算」

- 「証明」＝「プログラム」＝「計算」というテーゼの中では、「プログラム」＝「計算」というのが、一番わかりやすいように思う。IT技術者なら、「プログラム」＝「アルゴリズム」＝「計算」と考えればいいと思う。
- 「プログラム」という概念自体、20世紀の中頃に生まれたものであるので、「プログラム」＝「計算」 というテーゼは、三つのテーゼの中では一番新しいものである。
- この認識は、コンピュータの普及と、ITビジネスがコンシューマ中心にシフトして成功する中で、多くの人の中で自然に拡大した。

ただ、次のような問題もある。

「プログラム」＝「計算」

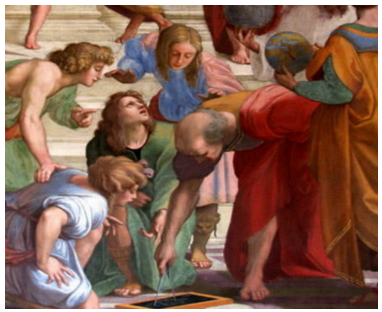
- 現代では、誰もが「コンピュータはプログラムで動く」ことを知っている。ただ、コンピュータは、万能の「情報処理機械」とみなされていて、それが「計算する機械」であることを意識する人は少ない。
- インターネットにしろYoutubeにしろFacebookにしろ、そのサービスを実現するソフトウェアが存在することを、多くの人は知っている。ただ、その基礎が、足し算や掛け算、電卓と同じ「計算」であることに、思いはいたらない。
- サービスを開発するプログラマ自身、数値計算でもしない限り、自分たちの仕事が「計算」の領域に属するという意識は薄いように思う。

サービス

プログラム

計算

証明



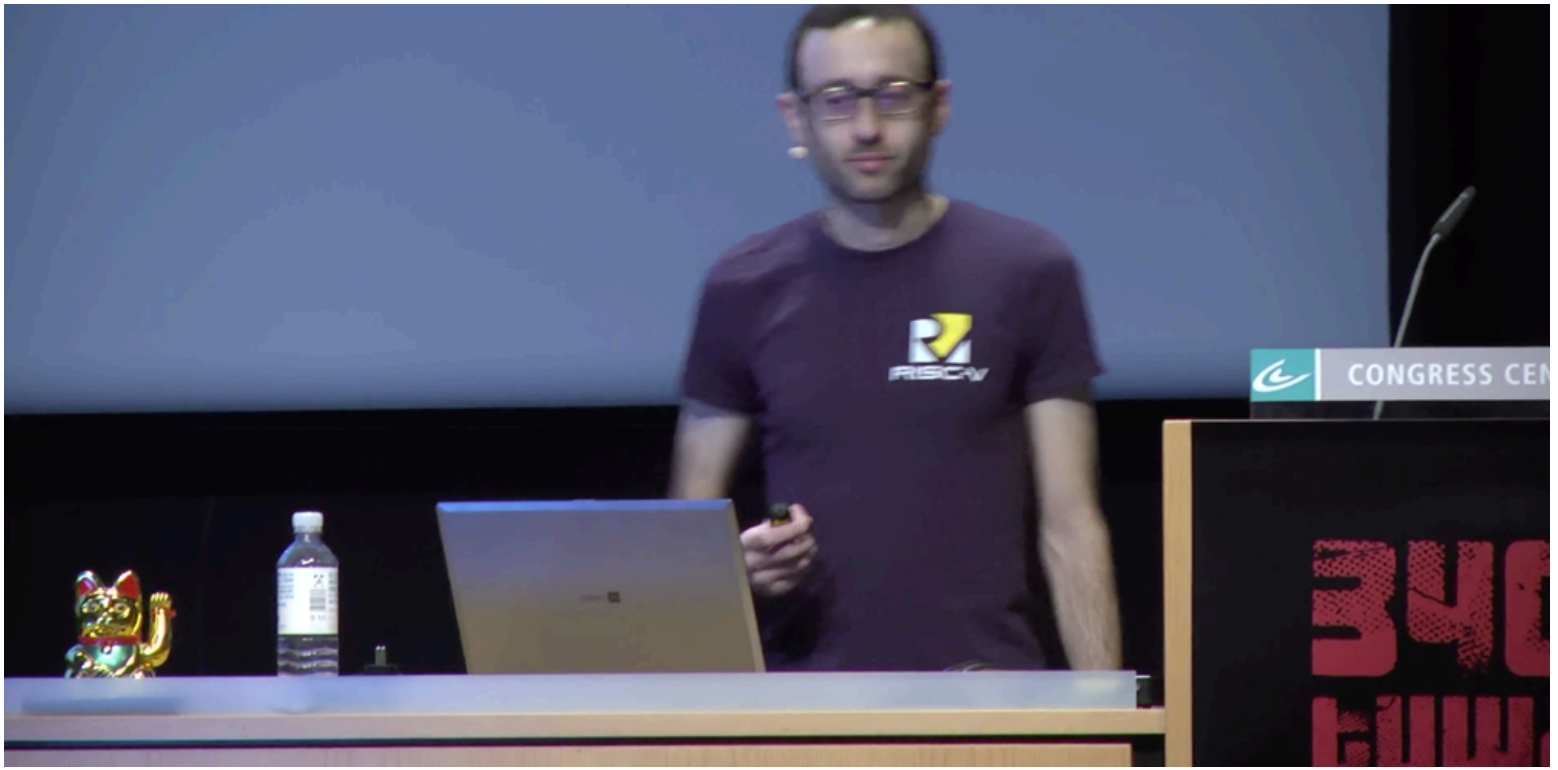
「プログラム」＝「計算」

- 「チャーチ＝チューリングのテーゼ」や「カリー＝ハワード対応」に言及することなく、コンピュータの「論理的＝数学的推論能力」を理解することは可能であろうか？
- 僕は、「プログラムの行っていることは、基本的には、計算である」ということが腑に落ちれば、十分可能だと思う。
- ラムダ計算もチューリング・マシンも、人間が行う計算の、プリミティブだが忠実なモデルに他ならない。それは、コンピュータが行なっている計算と同じものだ。計算に関して言えば、人間とコンピュータに違いはない。
- そうしたプリミティブな機械の働きが、「証明」とつながることを実感するには、いまなら、Coqの経験は、とても役に立つ。
- それは、「計算」＝「証明」、「証明」＝「プログラム」という、多くの人にとっての「ミッシング・リング」を埋めてくれるだろう。
  僕が、IT技術者にCoqの学習を勧める理由は、そこにある。

機械が論理的＝数学的推論能力を持つという認識が、やがて、現実を動かすようになる

「もうじき来るぞ。毎日のソフトとハードの開発に、機械がチェックする数学的証明が」

"Coming Soon: Machine-Checked Mathematical Proofs in Everyday Software and Hardware Developme"



Adam Chlipala http://bit.ly/2GppRxe

VoevodskyとUniMath

- 2017年に亡くなったVoevodskyは、Milner予想、Bloch-Kato予想を解くなど、代数幾何でグロタンディックが進もうとした道で、大きな業績を残した。また、Homotopy Type Theory という新しい型の理論で、数学の新しいスタイルでの基礎付けに、画期的な道を開いた。
- Voevodskyの最後の仕事は、数学の基礎とコンピュータに関係していた。彼は、数学の証明に、コンピュータを使うべきだと主張した最初の数学者の一人で、また、そのためのコンピュータによる証明支援システムのライブラリーUniMathを開発した。
  GitHub: https://github.com/UniMath/UniMath
- 2016年9月の講演 "UniMath - a library of mathematics formalized in the univalent style" https://goo.gl/3sJr1M

人間と機械の関係を考える

人工知能論の未来

人工知能論を深めるために考えるべきこと

- 人間の「創造性」と機械の力
  -- Coqでの証明は、人間と機械の協力で成り立っている
  -- 人間と機械との共生の未来？
- 機械との対話から、言語の問題を考える
  -- 開発の問題とは、機械に伝わる言語で、機械に正確に「意味」を伝えることの問題である
- 言語的認識と数学的認識
  -- なぜ、人間にとって、ことばによる説明はわかりやすく、数学的な説明は難しいのか？
- 量子機械の登場と人工知能論への影響
  -- 量子超越性と複雑性理論
- 情報の科学の未来