

実務家のための形式手法

厳密な仕様記述を志すための形式手法入門 第二版

# 形式手法導入に関わるガイダンス

形式手法を円滑に導入するために考慮すべき事項について、管理者および技術者の立場から理解し、形式手法導入の計画立案を開始できるようにするためのモジュール。

## ■ 事前知識・経験

- 形式手法の有用性を理解した上で導入に前向きに検討している
- 形式手法を具体的に選択・適用する知識・スキルは前提にしていない

## ■ 学習目標

- 形式手法の導入に必要な知識とスキルを得る
- 円滑な導入を支援するために必要な姿勢を獲得

## ■ 主な学習項目

1. 導入準備にあたって ..... p4
  - 直接的効果に加え間接的効果の重要性
  - 導入実践時のポイントと典型的な誤認識
2. 選択にあたって ..... p13
  - 選択方針、分類と代表的な手法
  - コスト(教育時、形式モデル構築時)
3. 適用にあたって ..... p24
  - 目的の明確化
  - 開発工程と形式手法
  - 代表的な「十戒」と事例
  - 開発プロセスと上流での仕様明確化の重要性

# 1. 導入準備にあたって

## ■ 直接的効果

- 成果物
- 記述物
- 分析／証明結果

## ■ 間接的效果

- 開発対象の理解とその共有
- 開発プロセス改善
- コミュニケーション改善
- 開発者の自信

- 管理職の意識
  - 形式手法への関心、期待、推進力
  - 担当ドメインエキスパートの意欲への影響
- 厳密な記述と議論
  - 最初の戸惑い
  - 新鮮な感動
  - 慣れ
  - 維持継続の努力と支援
- コミュニティ
  - 普段の相談相手
  - 高度なことも相談できる専門家（師匠）

## ■ 形式手法に対する認識

- J. A. Hall:  
Seven Myths of Formal Methods,  
IEEE Software, Vol.7, No.5, pp.11–19, 1990.
- J. P. Bowen and M. G. Hinchey  
Seven More Myths of Formal Methods,  
IEEE Software, Vol.12, No.4, pp.34–41, 1995.

## ■ 欧米では基本的素養

- 相互理解、議論/分析の道具建て
- 実用の段階
- 記述と分析の多面性
- 種々の理論・ツールの併用、統合

- 形式手法はソフトウェアが完全であることを保証できる
- 形式手法とは須らくプログラムの証明である
- 形式手法はセーフティクリティカルシステムにのみ有効である
- 形式手法は高度に訓練された数学者を必要とする
- 形式手法は開発コストを増加させる
- 形式手法はユーザには受け入れられない
- 形式手法は現実の大規模ソフトウェアには使われない

- 開発の初期の段階での誤りの発見に有効
- 開発対象のシステム自体を深く考えさせることに寄与
- いかなる応用分野にも有効
- 数学を基礎としてはいるものの、プログラムよりは理解し易い
- 開発コストを減少させる
- 顧客が購入しようとしているものの理解を助ける
- 産業界における実用プロジェクトに用いられて成功

- 実際の適用事例は興味深く魅力的
- 実世界から孤立した完全な世界
- 高度に数学的
- 実際に自分たちで適用するのは困難
- 自分の問題にぴったり合致する事例が必要
- 経営者を説得するのが難しい
- コストおよび効果が不明

# 一般的な企業の例：形式手法導入の観点から

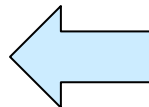
- 現場の技術者の高い能力
  - 形式仕様記述の修得は容易
- それぞれ自社の開発プロセスを所有
  - 典型的な(教科書的な)ソフトウェア開発プロセス
  - ソフトウェア工学の基本知識が不十分
- 形式手法と自分自身の開発業務との関わりに対する意識がない/  
誤認識
  - 高度に理論的
  - 完全、完璧、理想の世界

- 社内で形式手法担当者を1名決める
- その担当者に丸投げ
  - 調査、勉強
  - 試行
  - 評価
- 問題点=孤立
  - 開発プロセスとの関連なし
  - 開発現場との連携なし
  - 試行対象のドメインに(必ずしも)精通していない
  - 現場への導入の道筋も提示されていない
- ありがちな結論
  - 形式手法は(うちでは、まだ)使えない!

## 2. 選択にあたって

よく訊かれること:

どれを選んだら良いのか?



形式手法/ツール=100以上

## ■ すぐには回答できかねます

- 十分な情報提供が必要

- 開発対象、開発プロセス、目的、要員、期間、予算、etc.

## ■ 自らの決断と責任

## ■ 本来は適切な検討を行った上で、具体的な手法を選択すべきであるが、まずは、使ってみてどういうものを理解するという観点でVDM、SPIN を推奨

- ある程度のことができる
- 日本語の書籍もある
- 形式手法に対する感触を得られる

(以降途中から現場寄りの手法として特にVDMを念頭においた説明となっています)

## Formalism-in-the-Large

(cf. Programming-in-the-Large)

- モデル化 (Modeling)
- 抽象化 (Abstraction)
- 機能 (Function)
- 構成 (Construction)
- アーキテクチャ (Architecture)

## Formalism-in-the-Details

(cf. Programming-in-the-Small)

- モデル化 (Modeling)
- 分析 (Analysis)
- 性質 (Property)
- 検証 (Verification)
- 検査 (Testing)

# 基礎理論

- Z Notation: 抽象化、表記法、実行不能、ツール群
  - VDM-SL :親しみやすい表記法、ツール群、利用者のコミュニティ
  - VDM++ : オブジェクト指向、アーキテクチャ/部品
  - B Method: 抽象状態機械モデル
  - Event-B : B による状態機械をイベント駆動ネットワークにより結合したモデル
  - RAISE : EU の ESPRIT プロジェクトとして産業界での形式手法導入を意図して VDM 等の考え方を参考にして開発
  - OBJ : 代数的仕様記述
  - CSP, CCS : プロセス代数
  - モデル検査 : 有限状態機械モデル、振舞い分析/検証
- その他いろいろ

## モデル規範型:

- 集合論や命題論理、述語論理が基礎となる
- 不変条件、事前条件、事後条件を記述する
- 情報の構造や、ある状態から他の状態への変化をモデル化する
- 専用言語の利用によるコンパクトな仕様記述、曖昧さの除去、仕様の「実行」、「テスト」、「回帰テスト」、定理証明等の可能性が広がる

## 性質規範型:

- システムを外部から見てその性質を公理や抽象データ型を用いて表現、「代数仕様型」とも呼ばれる

## モデル検査:

- 振舞い仕様を、状態遷移モデルと、モデルが満たす条件として記述することで、全状態空間を生成し、自動検査する
- 従来の「テスト」では見つからない潜在的な障害を発見できる

## ■ 形式手法の種類

### ● モデル規範型

- VDM、RAISE (VDM+OBJ+CSP的仕様)、B Method、Z Notation、...
- 現場の技術者が理解しやすい

### ● 性質規範型

- CafeOBJ (OBJ3)、Maude等とツールを利用
- まだ現場に適用するには時期尚早

### ● モデル検査

- PROMELA、LTS 等の言語と SPIN (PROMELA)、LTSA (LTS) 等のツールを利用
- 採用する意味はあるが、現場の技術者には難しい

## ■ 形式手法の検証方法とツール

- 証明
  - RAISETool、Rodin、Coq、.....
  - 採用する意味はあるが、現場の技術者には難しい
- モデル検査
  - SPIN、LTSA、NuSMV、...
- 仕様実行
  - VDMTools、...
  - 現場の技術者に理解しやすく、形式検証の第一歩として採用

- 理論的にも経験的にも形式手法が有効
- VDMToolsで使用するVDMは、**現場寄りの形式手法**
  - 証明やモデル検査は、長期的にはやるべきだが、すぐにはできない
- VDM導入は、さほど難しくない
  - **3ヶ月程度の教育とコンサルティング**
  - **既存の**役立つソフトウェア工学**ツールと協調**して、より効果が出る
- モデル化は、以下が重要
  - **モデル化の範囲**を決め、仕様を分割統治
  - 名詞から型、述語から関数または操作
  - 陰仕様を作成してから、静的検証
  - 陽仕様を作成してから、動的検証
- VDMは構造化日本語仕様として使うことができる

## ■ 読むのは、難しい

- 簡単な解説
- 形式仕様のレビュー(OJT)

## ■ 書くには、経験が必要

- ドメイン知識、抽象化能力
- 言語仕様 (syntax)
- 対象のモデル化 (semantics)
- イディオム (pragmatics)

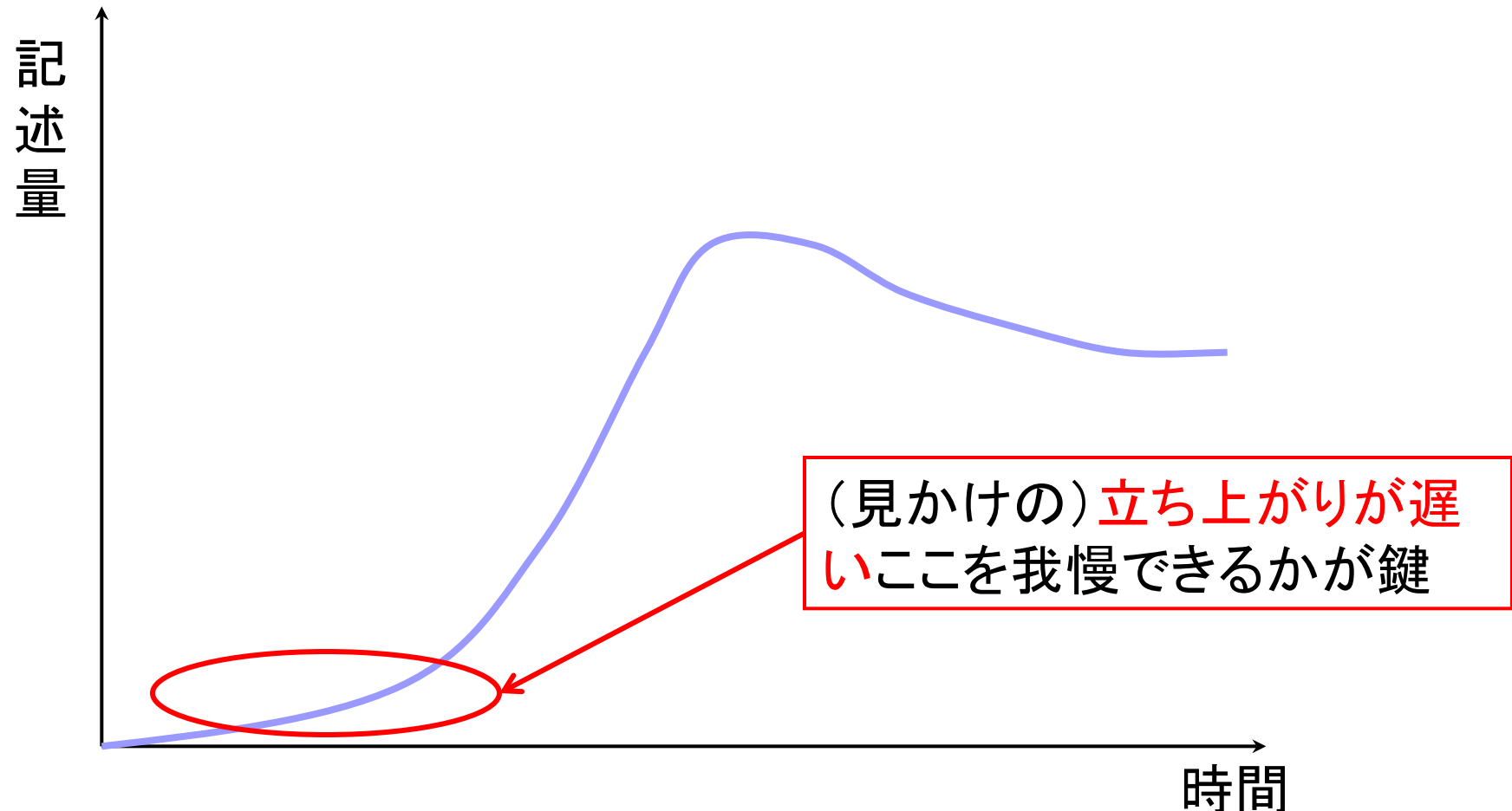
## ■ アニメーション/視覚化

- 仕様実行のこと
- multi-lingual
- multi-aspect

# 形式手法(VDM)の教育コストの例

	セミナー	教材	コンサルティング	受講者	備考
(1) SCSK	4日間	英語	無し	平均年齢50歳弱 形式手法知識 4人有 ソフト工学知識有	VDN記述予定者半数 (2人)が英語で脱落 c++Java,形式手法に ついて、実践経験無し
(2) フェリカ ネットワークス	4日間	日本語	3ヶ月/ 週1回	平均年齢20歳代 形式手法知識無 ソフト工学知識無	
(3)産業技術 総合研究所, オムロン	無し 自習	日本語 [Fitzgerald:10] [佐原:08]	無し	平均年齢30歳代 形式手法知識 1人だけ有 ソフト工学知識無	周囲に形式手法経験 者多数

- [Anthony Hall: Seven Myths of Formal Methods, IEEE Software, Vol.7, No.5, 1990]



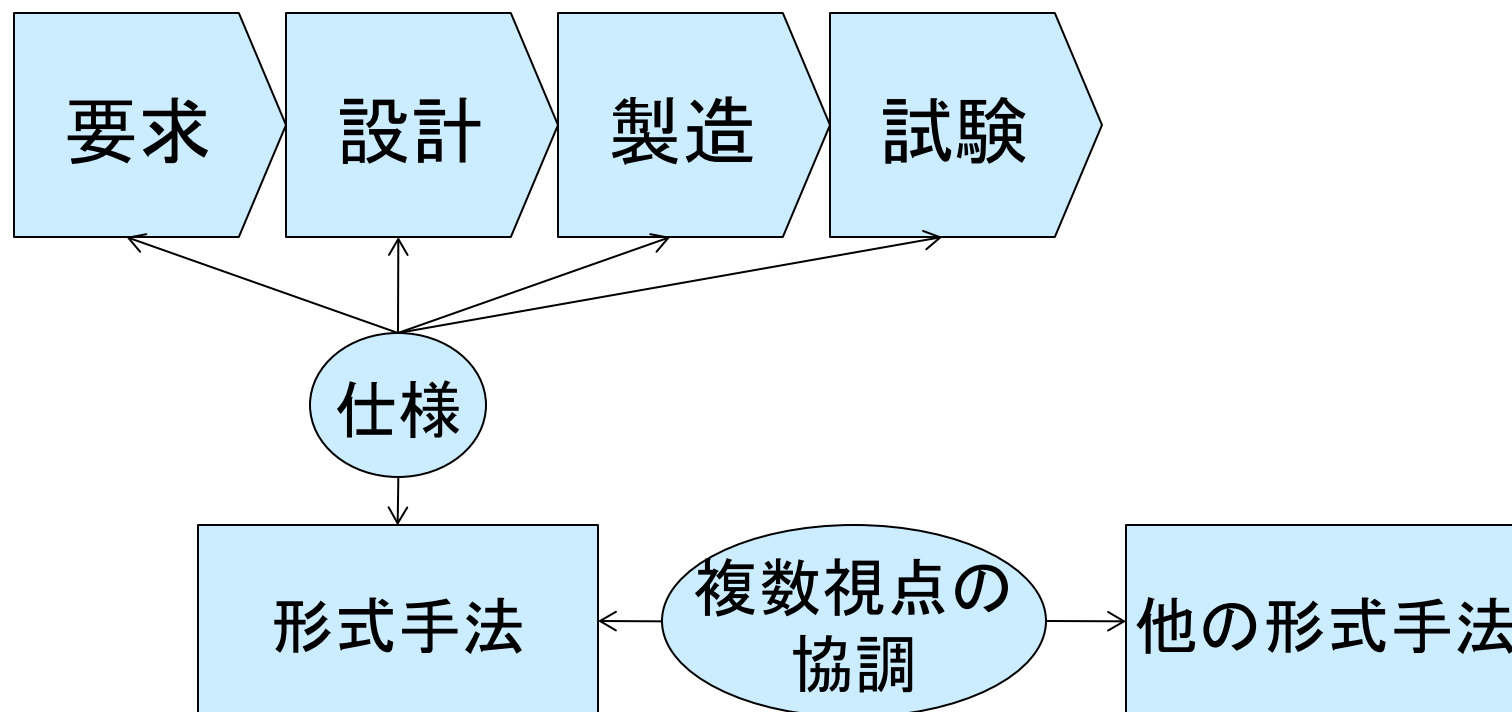
### 3. 適用にあたって

Formal Methods Europe (欧州のフォーマルメソッド推進組織)

[FME=http://www.fmeurope.org/?page-id=264](http://www.fmeurope.org/?page-id=264)

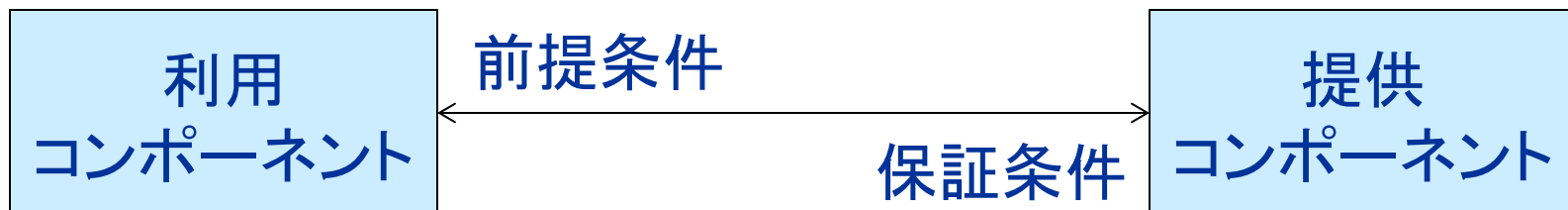
- 開発プロセス全体の厳密性を高め品質を向上させる
- システムの完全性や信頼性などの性質を向上させる
- 仕様の誤りを減少させる
- 要求定義の信頼性を改善させる
- 設計文書を改善し、設計の理解度を向上させる
- 保守や拡張のためのより堅固な土台を提供する
- 設計アーキテクチャの性質を精査する
- テストデータ選択のより合理的な基礎を提供する
- 設計と実現に誤りがないことを可能な限り保証する
- 特定の顧客や標準からの要求に適合させる

- どの開発工程に形式手法を適用するか？
- 適用工程と他の開発工程との関係をどうするか？
- 適用工程内で、形式手法と他の手法の関係をどうするか？
- 複数の形式手法を用いる場合、どのように組み合わせるか？



## ■ アーキテクチャの特性と相互作用に対して形式手法を活用できる

検証対象	形式手法の活用方法の例
特性	構造モデルで構成要素を定義 構成要素の操作と状態に基づいて、振舞いをモデル検査
相互作用	利用コンポーネントの前提条件に対して、提供コンポーネントの保証条件による充足性を検証



- Practice & Experience
  - 共有 / 再利用
  - ガイドライン / Cook Book
- 先進的な研究よりは、むしろ**基本的な知識と技術**
  - ソフトウェア工学
  - 形式手法
  - Technical Writing
  - 日常的なコミュニケーション
- 例えば、Ten Commandments of Formal Methods（形式手法の十戒）の実現

[J. P. Bowen & M. G. Hinchey: Ten Commandments of Formal Methods, IEEE Computer, Vol.28, No.4, pp.56–63, 1995]

- #1: Thou shalt choose an appropriate notation.
- #2: Thou shalt formalize but not over Formalize.
- #3: Thou shalt estimate costs.
- #4: Thou shalt have a formal methods guru on call.
- #5: Thou shalt not abandon thy traditional development methods.
- #6: Thou shalt document sufficiently.
- #7: Thou shalt not compromise thy quality standards.
- #8: Thou shalt not be dogmatic.
- #9: Thou shalt test, test, and test again.
- #10: Thou shalt reuse.

[J. P. Bowen & M. G. Hinchey: Ten Commandments of Formal Methods, IEEE Computer, Vol.28, No.4, pp.56–63, 1995]

- #1: 汝、適切な表記法を選ぶべし
- #2: 汝、形式化を行うべし、されど過ぐること勿れ
- #3: 汝、コスト予測をすべし
- #4: 汝、形式手法の師匠を身近に持つべし
- #5: 汝、従来よりの開発法を棄つること勿れ
- #6: 汝、十分に文書化すべし
- #7: 汝、自らの品質標準を危うくすること勿れ
- #8: 汝、独善となる勿れ
- #9: 汝、テストすべし、またテストすべし、さらにテストすべし
- #10: 汝、再利用すべし

## ■ 実践経験

- 開発プロセス全体にうまく組入れる

## ■ 継続と発展

- 導入方法、適用方法
- 要素技術開発
- 管理方法

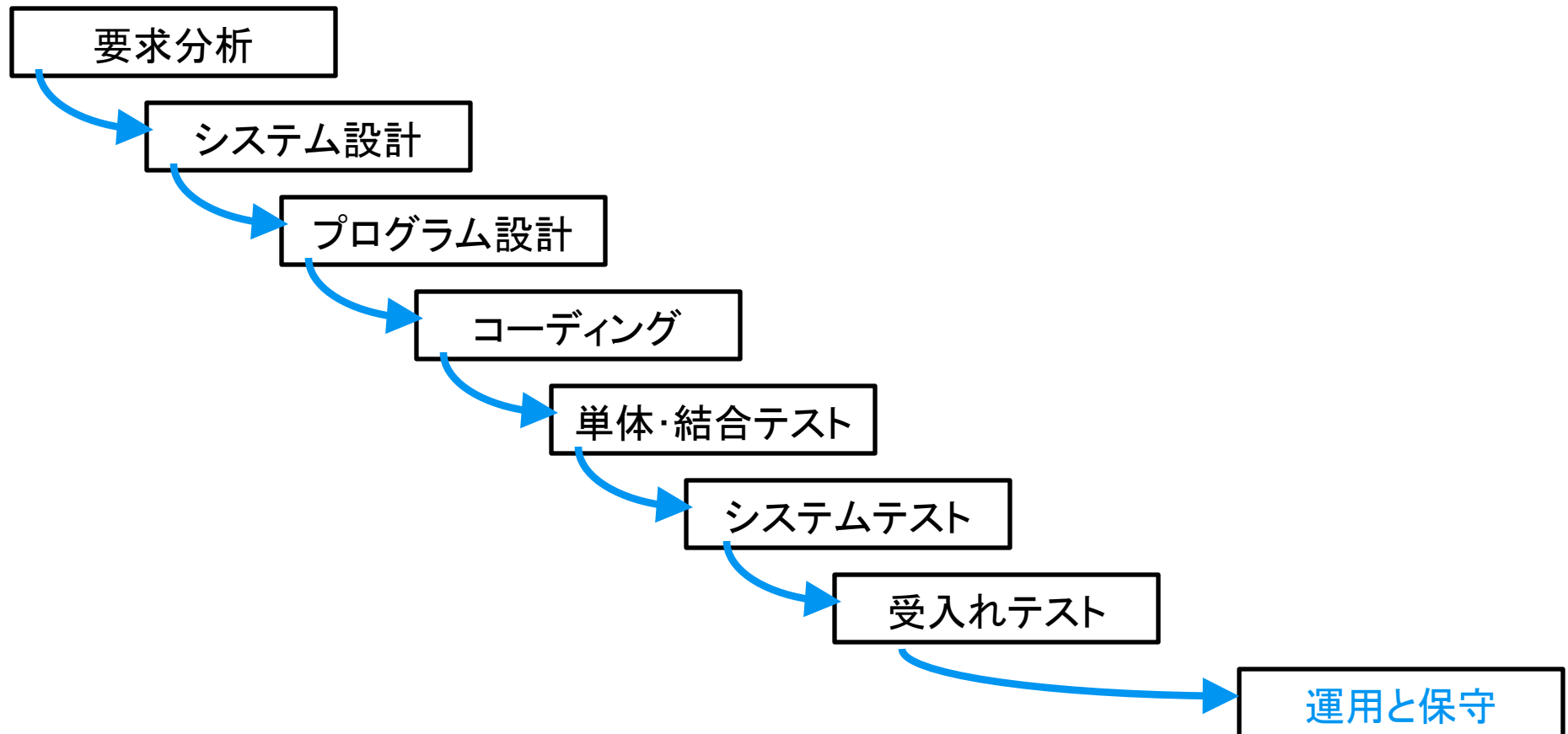
## ■ FeliCa の成功事例 vs. 十戒

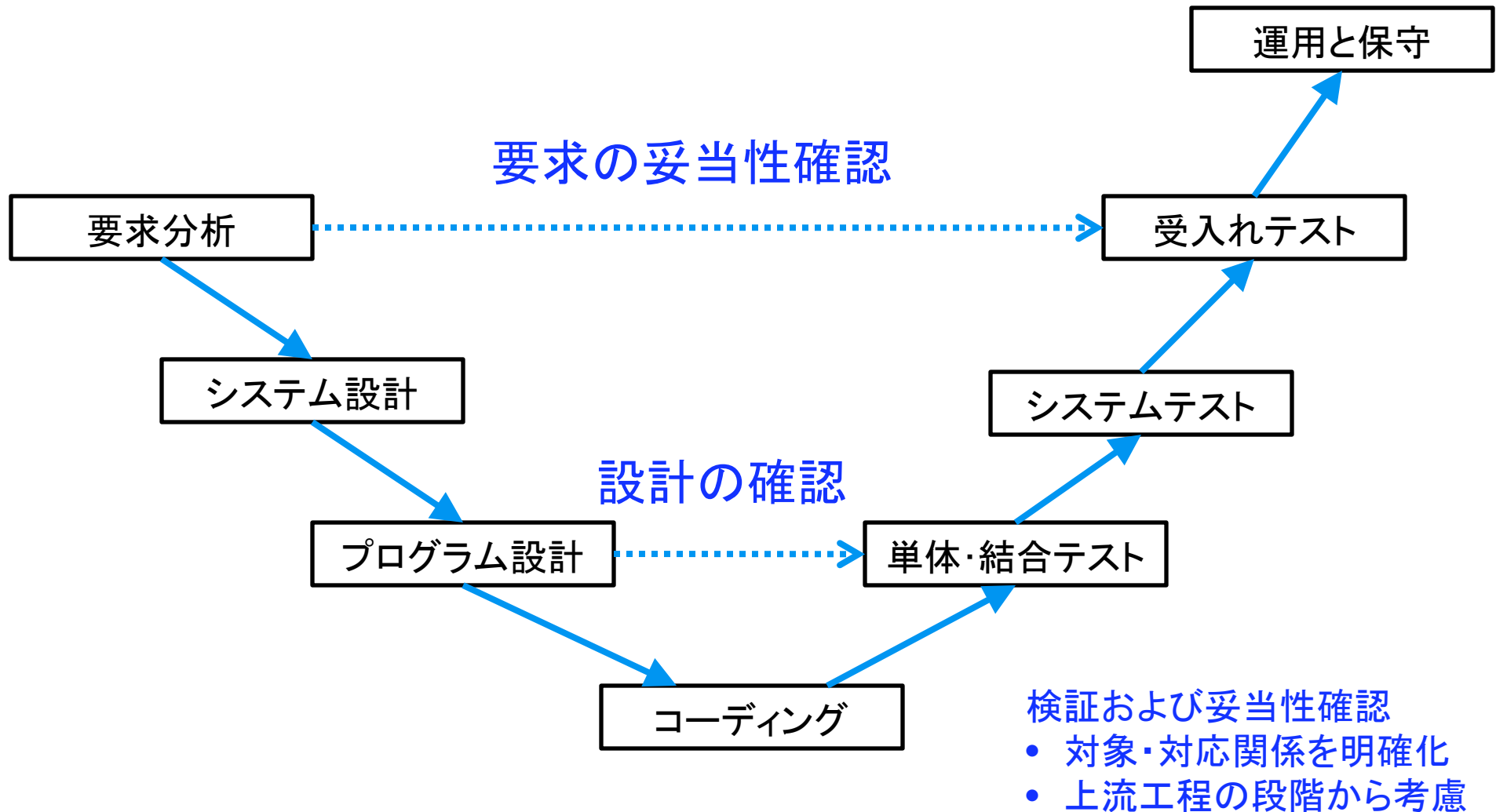
- いちいち当てはまる
- 分析/評価
- 導入ガイドラインとして取りまとめ中

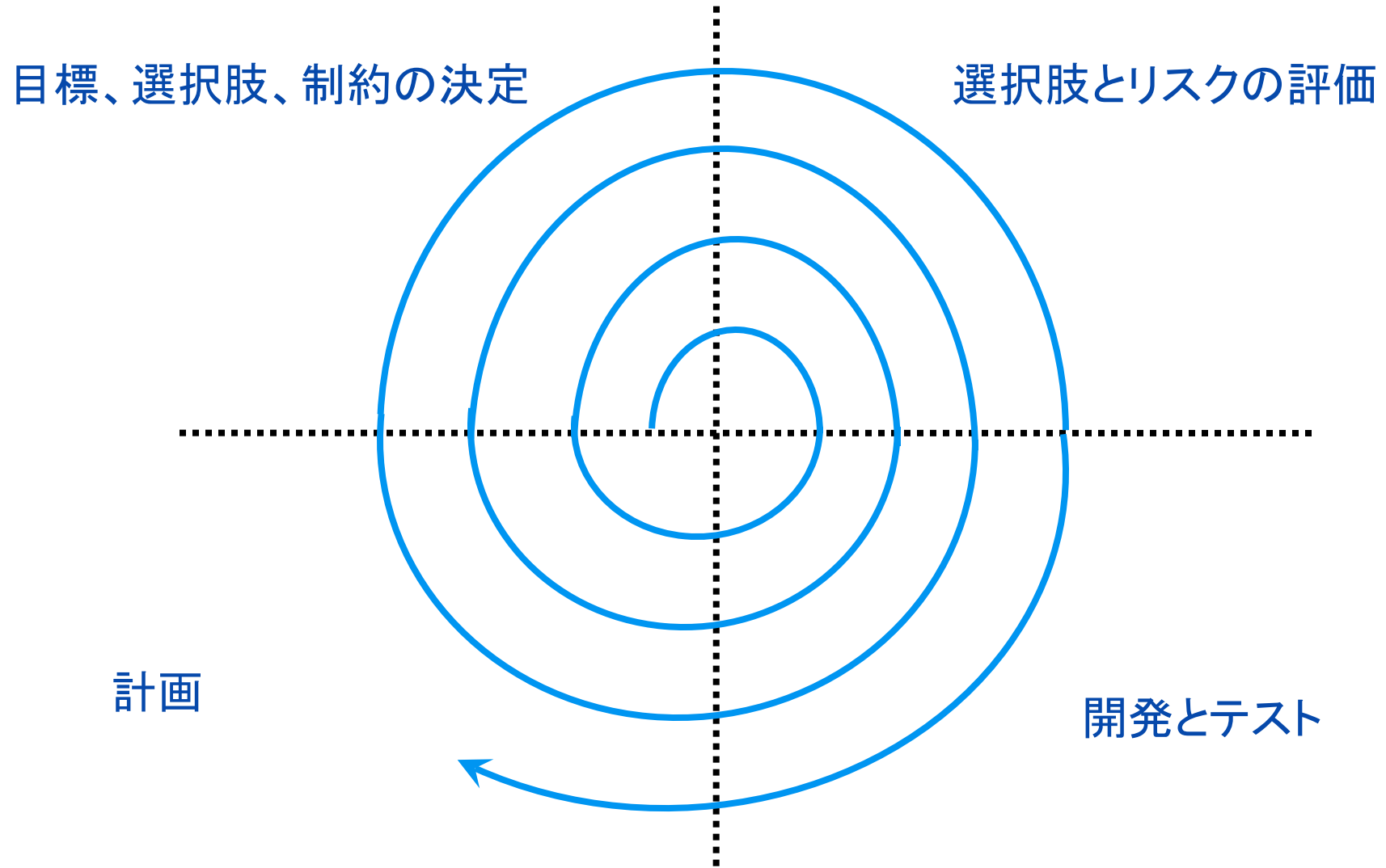
- 栗田太郎、中津川泰正、荒木啓二郎: 形式手法適用の実際と教訓—「形式手法の十戒」に照らし合わせて—, ソフトウェア工学の基礎ワークショップ(FOSE2009)論文集, 近代科学社, 2009年11月.

- 要求分析・定義: 利用者の意図や目的
- 仕様記述: システムが何をするか
- 設計: どのようにして実現するか
  - 概要設計
  - 詳細設計
- コーディング: プログラム作成
- テスト・デバッグ: 検査と修正
- 運用・保守: 利用と変更・拡張

# ウォーターフォール（落水）モデル※







## ■ 失敗プロジェクトの原因

- 不完全な要求 (13.1%)
- 利用者とのコミュニケーション不足 (12.4%)
- 資源の欠如 (10.6%)
- 非現実的な期待 (9.9%)
- 管理サポート不足 (9.3%)
- 要求と仕様の変更 (8.7%)
- 計画の欠如 (8.1%)
- もはや必要とされないシステム (7.5%)

(National Institute of Standards and Technology: Planning report 02-3, The Economic Impact of Inadequate Infrastructure for Software Testing, 2002)

## ■ 修正コストの例

- 要求 : 設計 : コーディング : 単体テスト : 出荷後  
1 : 5 : 10 : 20 : 200

## ■ 手戻り=全開発費の30～50%

- 前頁の失敗プロジェクトの原因のなかで、要求にかかわるものはどれですか？

# サマリー

形式手法を円滑に導入するために必要な知識とスキルおよび考慮すべき事項について理解し、形式手法導入の計画立案と導入の支援ができるようになるために、主に以下を学習

- 導入準備にあたって
  - 直接的効果に加え間接的効果の重要性
  - 導入実践時のポイントと典型的な誤認識
- 選択にあたって
  - 選択方針、分類と代表的な手法
  - コスト(教育時、形式モデル構築時)
- 適用にあたって
  - 目的の明確化
  - 開発工程と形式手法
  - 代表的な「十戒」と事例
  - 開発プロセスと上流での仕様明確化の重要性



# 付 録

J. P .Bowen, et al. : Ten commandments of formal method,  
IEEE Computer 28,No.4,pp.55–63(1995)

## ■ レベル0: 形式仕様記述

- 数学的な記法を用いて厳密な仕様を記述し、証明や分析までは行わずに、この記述を基にしてプログラムを開発する

## ■ レベル1: 形式的開発および検証

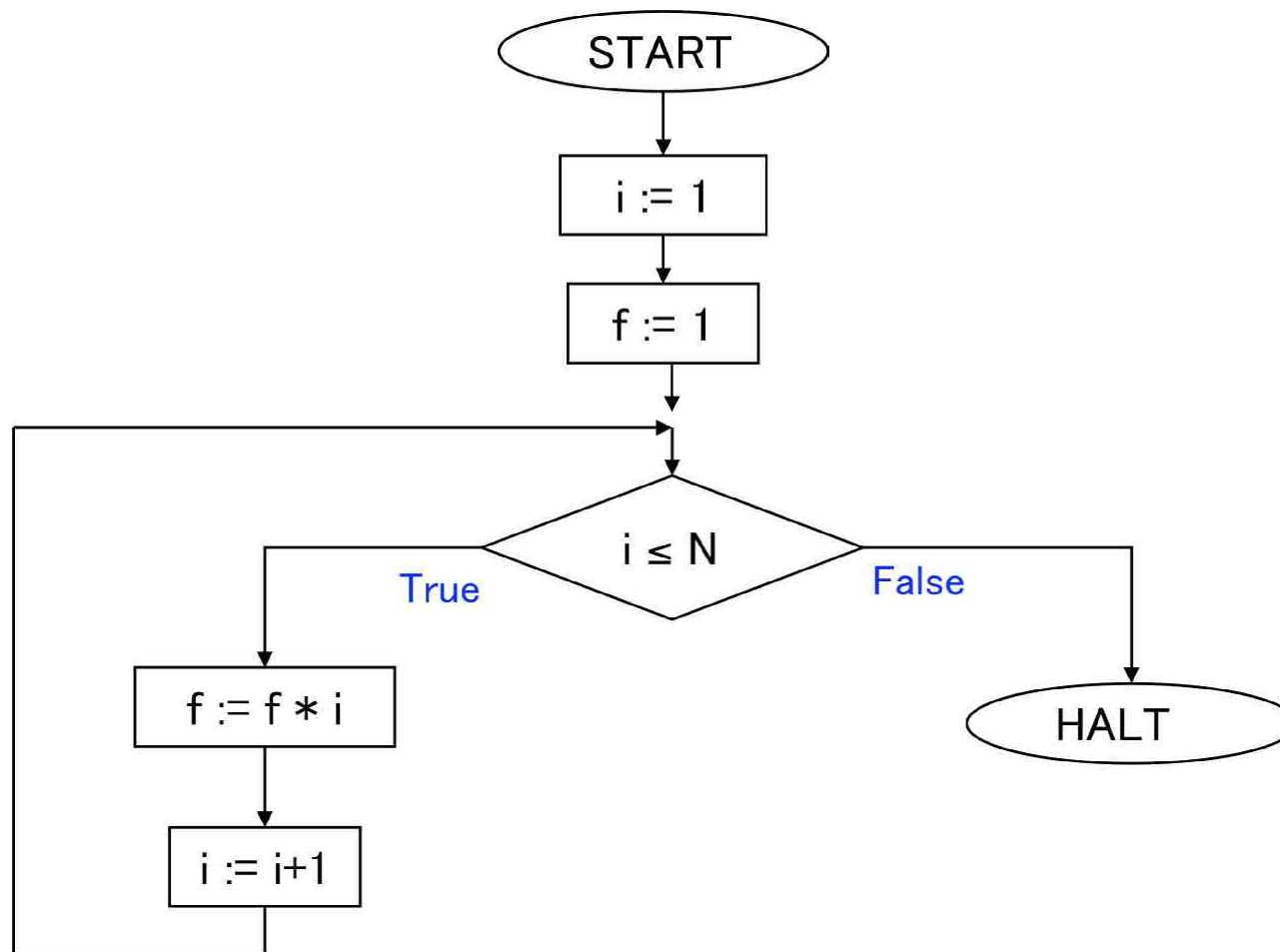
- プログラムの性質を証明したり、詳細化により仕様からプログラムを作成する

## ■ レベル2: 機械支援による証明

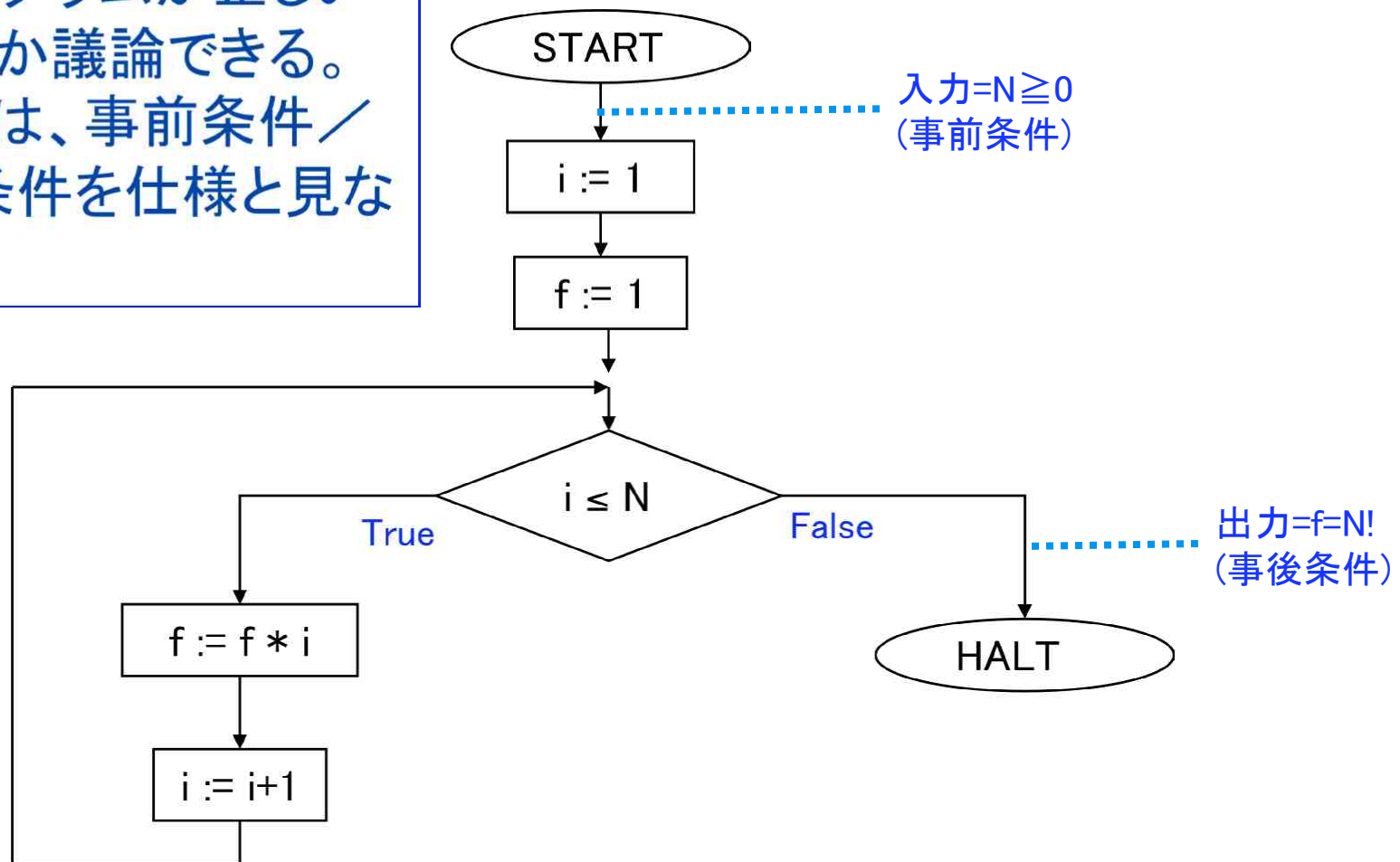
- 定理証明器や証明支援器を用いてプログラムの性質を証明する

# 正しいプログラムとは？※

以下のプログラムは、正しいプログラムでしょうか？



仕様が与えられて初めてプログラムが正しいかどうか議論できる。ここでは、事前条件／事後条件を仕様と見なす。



$\{N \geq 0\}$

$i := 1;$

$f := 1;$

While  $i \leq N$

Do

$f := f * i;$

$i := i + 1$

End

$\{f = N!\}$

# 証明図(これが証明です!) ※

$$\begin{array}{c}
 \begin{array}{c}
 N \geq 0 \\
 \Rightarrow 1 \leq 1 \leq N+1 \\
 \wedge 1 = (1-1)!
 \end{array}
 \end{array}$$


---


$$\begin{array}{c}
 \begin{array}{c}
 \{ 1 \leq i \leq N+1 \\
 \wedge 1 = (1-1)! \} \\
 i := 1 \\
 \{ 1 \leq i \leq N+1 \\
 \wedge 1 = (i-1)! \}
 \end{array}
 \end{array}$$


---


$$\begin{array}{c}
 \{ N \geq 0 \} \\
 i := 1 \\
 \{ 1 \leq i \leq N+1 \\
 \wedge 1 = (i-1)! \}
 \end{array}$$


---


$$\begin{array}{c}
 \{ 1 \leq i \leq N+1 \\
 \wedge 1 = (i-1)! \} \\
 f := 1 \\
 \{ 1 \leq i \leq N+1 \\
 \wedge f = (i-1)! \}
 \end{array}$$


---


$$\begin{array}{c}
 \{ N \geq 0 \} \\
 i := 1; f := 1 \\
 \{ 1 \leq i \leq N+1 \wedge f = (i-1)! \}
 \end{array}$$


---


$$\begin{array}{c}
 \{ 1 \leq i+1 \leq N+1 \wedge f = (i-1)! \wedge i \leq N \} \\
 f := f * i \\
 \{ 1 \leq i+1 \leq N+1 \wedge f = i! \}
 \end{array}$$


---


$$\begin{array}{c}
 \{ 1 \leq i+1 \leq N+1 \wedge f = (i-1)! \wedge i \leq N \} \\
 f := f * i \\
 \{ 1 \leq i+1 \leq N+1 \wedge f = i! \}
 \end{array}$$


---


$$\begin{array}{c}
 \{ 1 \leq i+1 \leq N+1 \wedge f = i! \} \\
 i := i + 1 \\
 \{ 1 \leq i \leq N+1 \wedge f = (i-1)! \}
 \end{array}$$


---


$$\begin{array}{c}
 \{ 1 \leq i \leq N+1 \wedge f = (i-1)! \wedge i \leq N \} \\
 f := f * i; i := i + 1 \\
 \{ 1 \leq i \leq N+1 \wedge f = (i-1)! \}
 \end{array}$$


---


$$\begin{array}{c}
 \{ 1 \leq i \leq N+1 \wedge f = (i-1)! \} \\
 \text{While } i \leq N \text{ Do } f := f * i; i := i + 1 \text{ End} \\
 \{ 1 \leq i \leq N+1 \wedge f = (i-1)! \wedge i > N \}
 \end{array}$$


---


$$\begin{array}{c}
 \{ 1 \leq i \leq N+1 \wedge f = (i-1)! \} \\
 \text{While } i \leq N \text{ Do } f := f * i; i := i + 1 \text{ End} \\
 \{ f = N! \}
 \end{array}$$


---


$$\{ N \geq 0 \} i := 1; f := 1; \text{While } i \leq N \text{ Do } f := f * i; i := i + 1 \text{ End } \{ f = N! \}$$

実務家のための形式手法

厳密な仕様記述を志すための形式手法入門

# 形式手法導入に関わるガイダンス

独立行政法人情報処理推進機構

技術本部 ソフトウェア・エンジニアリング・センター

統合系システム・ソフトウェア信頼性基盤整備推進委員会

上流品質技術部会 人材育成WG(編)

2013年3月 第二版発行

記載されている個々の情報に関しての著作権及び商標はそれぞれの権利者に帰属するものです  
なお、本書の内容は将来予告なしに変更することがあります