

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

## 形式手法導入に係るガイダンス(このモジュールについて)



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

## 形式手法導入に係るガイダンス(このモジュールについて)



### ■ 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

SEC-FM1-02-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center

3

(見出しに※がついているスライドは管理者コースや初回学習時にはスキップ可。)

講師の方へ:

このモジュールでは、形式手法導入の次のような側面を強調してください。

前提知識も含め、見ためほど難しいものではなく、目新しいものでもない。  
証明だけでなく、完全性には欠けるものの、よりコストパフォーマンスに優れた簡便な手法もある。  
また、マネージャーの精神的サポートも含め、導入に対する周囲の理解が重要であること。  
受講者には、自分のプロジェクトのどの部分を改善するために形式手法をどのように組み込むか、  
自分のプロジェクトがどのような開発プロセスを使っている、どのように改善すべきか、  
仕様実行と証明のコストを実感する、  
といったところに注意させるようにしてください。

具体的には導入へのガイダンスを以下の三段階に分けて説明しています。

1. 導入の準備
2. 導入手法の選択
3. 導入決定後の適用時

各段階ごとに以下の点を意識し、斜体部分に注意してお話ください。

まず、導入の準備段階では

- 導入の効果について得られそうなものと期待しているものとの対応
  - 直接的な効果だけでなく間接的な効果について強調
- 態度や環境を含め、導入する際のポイントは何か

- 知見として論文化もされるほどの都市伝説と実際の差、よくある失敗事例を強調

次に、手法の選択にあたっては

- 形式手法にも様々な分類があり具体的な特徴は異なること
  - 多くの場合、モデル規範型のVDMが有望であること
- 導入コストにも特徴があること
  - VDMは導入コストも高くなく、特に書かれた仕様記述は読むのが容易であること。  
その一方で仕様記述を書くには、VDM固有ではない一般的な意味での難しさがああり、立ち上がりも遅いこと。

次に、実際の適用にあたっては

- 導入/適用の目的
  - 目的の明確化は重要なので例なども使いながら具体的なイメージを持ってもらう
- どの開発工程で用いるか
  - 具体的には一律な適用方法はなく、個別の目的などに応じて各自のプロセスと関連付ける必要がある
- 適用時に参考とするもの
  - 既存の知見や先行の成功事例を紹介する
- 上流での仕様の明確化の重要性
  - 様々な開発プロセスがありえるが、それぞれの開発単位での上流工程の仕様の重要性を強調

このコースは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

(形式手法適用のレベル0)を想定していますが、さらに進んだ証明などに興味がある場合は付録などを利用し紹介するのも良いでしょう。

適用レベルの定義は、”J. P. Bowen, et. al.: Ten commandments of formal method. IEEE Computer 28, No.4, pp.55-63(1995)”より。

# 1. 導入準備にあたって

## 形式手法適用の効果



### ■ 直接的効果

- 成果物
- 記述物
- 分析／証明結果

### ■ 間接的効果

- 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

SEC-FM1-02-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center

5

形式手法導入の効果には、

#### ・[直接的効果](#)

成果物としての仕様の厳密な記述、記述物としてのユーザマニュアルの充実、開発工程を通して利用可能な検証結果

#### ・[間接的効果](#)

厳密な記述による開発の対象の問題の認識、正確なコミュニケーションによるチーム内外との理解の共有

がある。

明示的に書かれたものによって知識や知見、ノウハウを共有できる。

また、開発者にとっては、どのようなシステムをどのようにして開発しているのか認識できるので、自信を持って開発を行うことができる。

## 実践のポイント



- 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）

SEC-FM1-02-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center

6

・形式手法に限ったことではないが、管理職(上司)の意識が重要である。

担当者は上司の意向を気にする。

・形式手法に基づいた開発における厳密性に関しては、最初は戸惑いがある。

しかし、時間が経つと慣れてくる。だから努力と支援が必要である。

形式手法の意義がわかると新鮮な感動を受ける。

## 形式手法の特質



### ■ 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

J. A. Hall: Seven Myths of Formal Methods

読みやすく形式手法の特性・特質を非常によく表している。

古い論文ではあるが、現在でも通用する。

J. P. Bowen and M. G. Hinchey: Seven More Myths of Formal Methods

上の論文の補足をしている。

数学は普遍的な体系なので、言語や文化に依存せずひろく使うことができる。



## 形式手法に関する七つの誤解 [Hall:90]



- 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

SEC-FM1-02-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center

8

形式手法の本場の欧州においても、形式手法が正しく認識されていない と

Anthony Hall が論文を書いた。

この論文は、とても素晴らしい論文で現在でも通じる。

時々読めば、形式手法に関する新たな発見もあるし、理解も深まる。

[Hall:90]

Anthony Hall: Seven Myths of Formal Methods, IEEE Software, Vol.7, No.5, pp.11-19, 1990.

## 形式手法に関する七つの真実 [Hall,1990]



- 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

先ほどの誤解を全て裏返すと、形式手法の真実の姿が浮かび上がってくる。

形式手法の基礎となっている数学は、集合論、論理学、オートマトンなどせいぜい大学1年、2年くらいで学ぶものである。

問題対象によっては、問題対象特有の理論があるが、それ以外の知識は基本的なもので十分である。

## 日本企業における七つの反応



- 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

- ・フェリカネットワークスの K さんは、適応事例の説明にひっぱりだこ!
- ・形式手法だけで、ソフトウェア開発が完結するわけではない。
- ・「高度に数学的」という認識は Anthony Hall のときから変わっていない。
- ・後半の 4 つは、新しい技術を導入しない言い訳である。

## 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

日本の技術力は高く、それぞれ自社の開発プロセスを所有しているが、従来のソフトウェア工学の成果を活用できていないところも多い。

特に、中間成果物が曖昧で、説明できない部分が残っていたりする。

形式手法を導入することで、自分たちのやり方をもう一回見直す契機にもなる。

## ありがちな例：形式手法導入の調査・検討時



- 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!

企業で形式手法の導入を検討した例には、共通する点がいくつか見られる。

担当者を一人決める。しかも、若い人が多い。システム開発経験も少ない。現場への影響もない。

## 2. 選択にあたって

## 手法の選択方針



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るべきであるが、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)

SEC-FM1-02-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center 14

J. P. Bowen のウェブに 100 種類以上ある。

どれがよいかは、何をどういう風にやりたいかによる。

ここでは、手始めとして VDM とモデル検査の SPIN を薦めたい。


薦める理由

VDM : システムの(抽象的な)機能と構成という静的側面

SPIN : システムの振舞いという動的側面

二つの異なる観点の代表的方法であり、夫々日本語の教科書や書籍が充実している。

## マクロな観点とミクロな観点※



Formalism-in-the-Large (cf. Programming-in-the-Large)	Formalism-in-the-Details (cf. Programming-in-the-Small)
<ul style="list-style-type: none"><li>● モデル化 (Modeling)</li><li>● 抽象化 (Abstraction)</li><li>● 機能 (Function)</li><li>● 構成 (Construction)</li><li>● アーキテクチャ (Architecture)</li></ul>	<ul style="list-style-type: none"><li>● モデル化 (Modeling)</li><li>● 分析 (Analysis)</li><li>● 性質 (Property)</li><li>● 検証 (Verification)</li><li>● 検査 (Testing)</li></ul>

基礎理論

SEC-FM1-02-2Copyright © 2012,2013 IPAIPA Software Engineering Center15

(※このスライドは管理者コースや初回学習時ではスキップも可能)

形式手法によるモデル化には、抽象化と精密化の2つの方向があり、どちらも基礎となる数理論理に基づいている。



## 多様性と多面性※



- 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

SEC-FM1-02-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center 16

(※このスライドは管理者コースや初回学習時ではスキップも可能)

ソフトウェア自体に多様性と多面性があるので、形式手法もそれに応じて多種多様。  
一つですべてをカバーする万能薬は存在しない。

## 形式手法の一部の分類※



### 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

SEC-FM1-02-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center 17

(※このスライドは管理者コースや初回学習時ではスキップも可能)

形式手法の一部を分類する方法としては、「モデル規範型」、「性質規範型」と「モデル検査」がある。

## 現状、どのような形式手法の適用が多いか？



### ■ 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**

SEC-FM1-02-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center 18

「モデル規範型」は現場の技術者が理解しやすい。

通常のプログラムと同じように、状態があって、状態マシンとして状態がどう変化するかを仕様として書く。

「性質規範型」(「代数仕様型」)は、少し抽象度が高いのと記法になじみがないので、現場の技術者には難しい。

「モデル検査」は、採用する意味はあるが、検証すべき性質の抽出が現場の技術者には難しい。

形式手法の検証方法として、

- ・段階的詳細化などの証明
- ・時相論理に基づくモデル検査
- ・仕様実行

がある。

仕様実行はテストと同様、ある入力値について簡単に検証できるが、完全な証明はできない。

## 現場向きの軽量な形式手法:VDM



### ■ 形式手法の検証方法とツール

- 証明
- 段階的洗練
  - 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

SEC-FM1-02-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center 19

完全な自動証明はできないので、人間が数式を変形する対話的な証明が必須である。

仕様と設計の関係といった詳細化のステップが小さいと人による検証が容易である。

ある種の型付き言語でプログラムを作ることが、数学的な関数記述と同じであることが分かっており、それらを応用したツールは比較的使いやすいが、まだ研究段階である。

モデル検査は、便利なツールは揃っているが、抽象的にモデル記述するためのサポートがなく、システム全体をそのまま検証をするには、現在の計算機的能力では不足であり、問題の分割が必要という課題もある。

VDM のツールは、複数の検証機能が用意されており、プログラム風の記述であることから始めやすい。

## 形式手法とVDMの有用性



- 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

SEC-FM1-02-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center 20

VDM は、オブジェクト指向設計など、ソフトウェア工学の他の手法を導入する基盤としても有用である。その場合、どこまでが形式手法の、どこまでがオブジェクト指向の効果かという分析は困難であるが、それはあまり問題にならないはず。

証明は、すべてのモデルのすべての部分ではなく、特に信頼性を求められる部分に用いるとよい。具体的には、VDMで全体をモデル化して仕様実行をおこない、仕様実行などで信頼性を保ちつつ、ここぞという部分はモデル検査や証明を用いて開発を行う、といった方法である。

VDM の導入はさほど難しくないが、モジュール化や階層などコンサルティングが有用な部分もある。

## VDMによる形式仕様の特徴



### ■ 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

SEC-FM1-02-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center 21

形式手法による仕様は、読むのは難しいが、書くのは容易ではない。

言語仕様を知ってるだけでなく、書く対象をきちんと理解していないと記述できないからである。

また、抽象化に関しては、数学的な知識やセンスが求められる。

数学的なセンスを使わなくても、仕様を動かしながら記述することで、妥当性を確認しながら決定することもできる。

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

SEC-FM1-02-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center 22

この表は VDM に関する教育を実施した事例である。

[Fitzgerald:10]

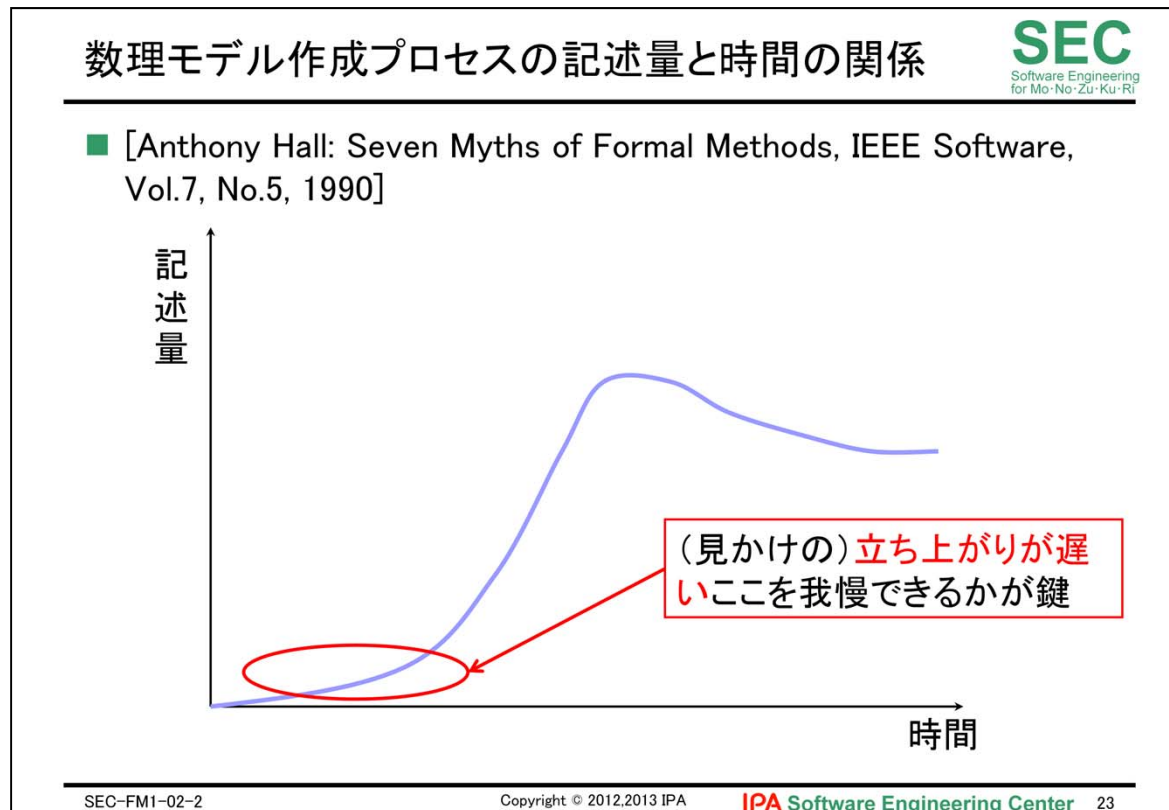
John Fitzgerald, Peter Larsen, Peter Gorm Larsen, Paul Mukherjee, and Nico Plat, 酒匂 寛(訳):

VDM++ によるオブジェクト指向システムの高品質設計と検証, 翔泳社, 2010年8月

[佐原:08]

佐原 伸: ～ソフトウェアトラブルを予防する～形式手法の技術講座, ソフト・リサーチ・センター, 2008年





形式的な仕様を構築する際の、時間経過と記述量の関係はグラフのようになる。

横軸は時間経過、縦軸は出来上がった記述量である。

特徴的なのは、立ち上がりが極めて悪いということであり、ドメインの理解や抽象的なモデル化を行うために時間がかかる。

その時間を我慢して乗り切ると、ある時理解が進み、一気に書けるようになる。

その後、記述を整理・改善するために記述量が落ち着いていく。

立ち上がりは、プロジェクトリーダーやプロジェクトマネージャーも我慢しなければならないところである。

この段階では、上流のところでシステムの本質を捉えたモデルを記述し、抽象度の高い仕様を書くことも重要であるが、これも時間のかかる要因のひとつである。



### 3. 適用にあたって

## 形式手法を適用する目的



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>

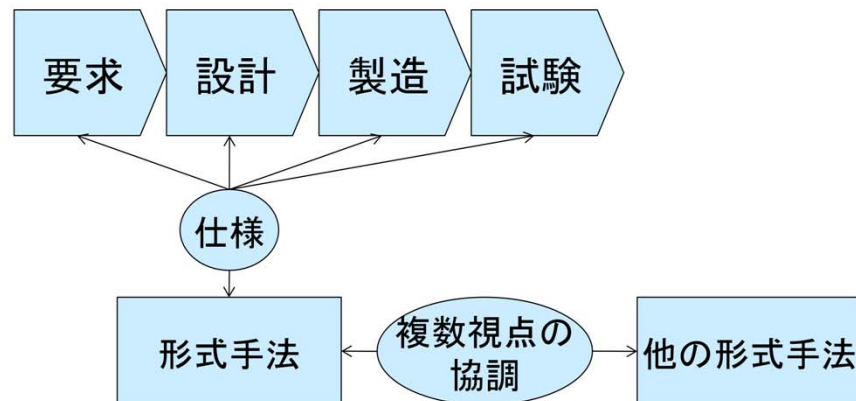
- 開発プロセス全体の厳密性を高め品質を向上させる
- システムの完全性や信頼性などの性質を向上させる
- 仕様の誤りを減少させる
- 要求定義の信頼性を改善させる
- 設計文書を改善し、設計の理解度を向上させる
- 保守や拡張のためのより堅固な土台を提供する
- 設計アーキテクチャの性質を精査する
- テストデータ選択のより合理的な基礎を提供する
- 設計と実現に誤りがないことを可能な限り保証する
- 特定の顧客や標準からの要求に適合させる

何のために形式手法を用いるのかを明確にすることが、まず重要である。

## 開発プロセスと形式手法



- 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？




SEC-FM1-02-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center 26

開発プロジェクトに形式手法を導入する場合の留意点 4点。

## 形式手法とアーキテクチャ

  
Software Engineering  
for Mo·No·Zu·Ku·Ri

■ 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

利用  
コンポーネント

← 前提条件

→ 保証条件

提供  
コンポーネント

SEC-FM1-02-2Copyright © 2012,2013 IPAIPA Software Engineering Center 27

アーキテクチャの特性と相互作用について、次のようにして形式手法を活用できる。

・特性

アーキテクチャの構造モデルによって構成要素を定義し、構成要素の操作と状態に基づいて、振舞いをモデル検査する

・相互作用

アーキテクチャの構成要素としてのコンポーネントについて、コンポーネントの利用側における前提条件に対して、提供側の保証条件による充足性を検証する。

アーキテクチャを考慮することで、形式手法をどのようにと入れるかを考える手がかりとなる。

ソフトウェアの静的あるいは動的な関連を見極め、検証する対象とする特性に対して、適切な形式手法を導入すべき。

## 導入に向けて



### ■ 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

SEC-FM1-02-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center 28

我が国でも、形式手法適用の実践と事例の蓄積が必要で、

また、それらに基づいた日本流のガイドラインも必要とされている（本資料もそのひとつ）。

先進的な「研究」は論文を書くことが目的であることが多く、実際の問題とはギャップがある可能性がある

。

むしろ、日常的な基礎を固めることが重要である。

日常的な基礎とは、ソフトウェア工学の成果、形式手法の基礎、さらには文章、コミュニケーション等である。

## Ten Commandments of Formal Methods※



[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.

SEC-FM1-02-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center 29

(※このスライドは管理者コースや初回学習時ではスキップも可能)

「形式手法の十戒」

原文では古典的な英語を意図的に使っている

## 形式手法の十戒



[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

SEC-FM1-02-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center 30

#2 形式化、抽象化を行うこと、ただしそれが目的化してはいけない。

#5 形式手法だけで独立して完結していないので、従来の開発手法の中に上手く形式手法を取り入れる。



## FeliCa の事例から学ぶ導入/適用方法



### ■ 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.

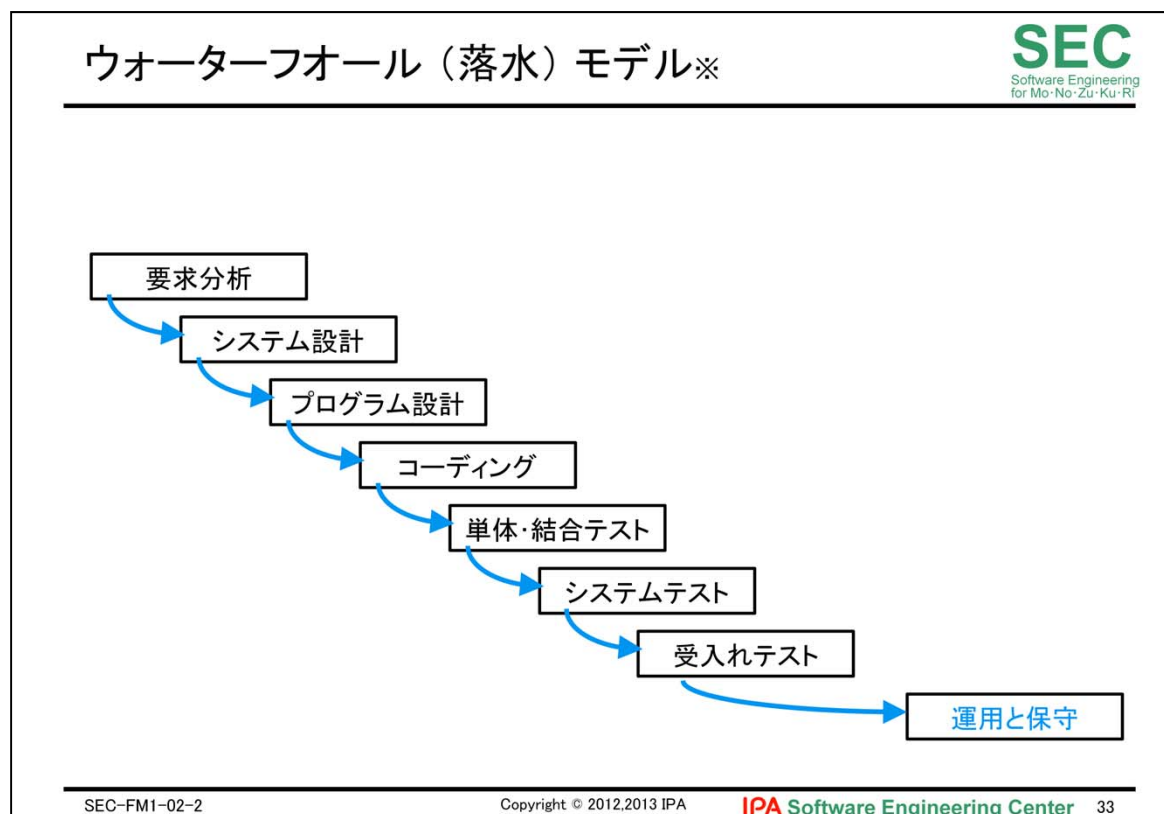


## ソフトウェア開発の過程(プロセス)再考



- 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

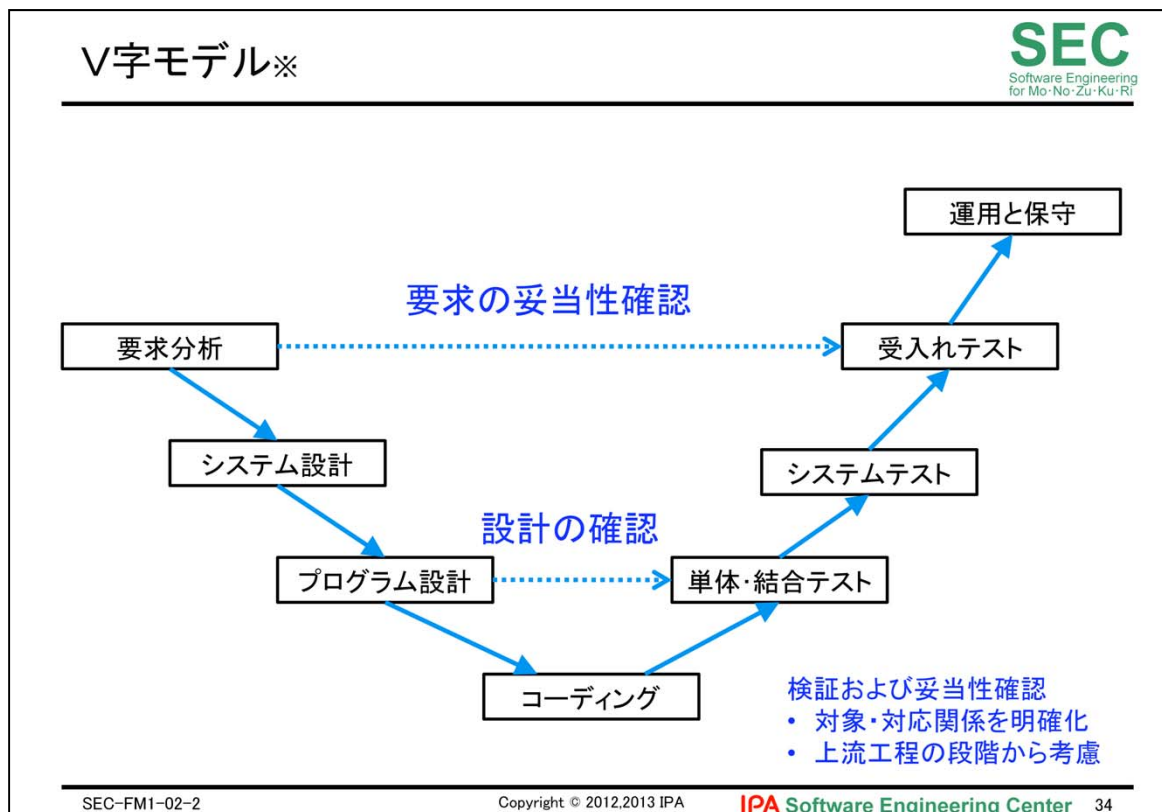
ソフトウェア開発は、いくつかの過程に分けられる。



(※このスライドは管理者コースや初回学習時にはスキップも可能)

それぞれの過程で、どの段階でも役に立つ形式手法が見つかる。

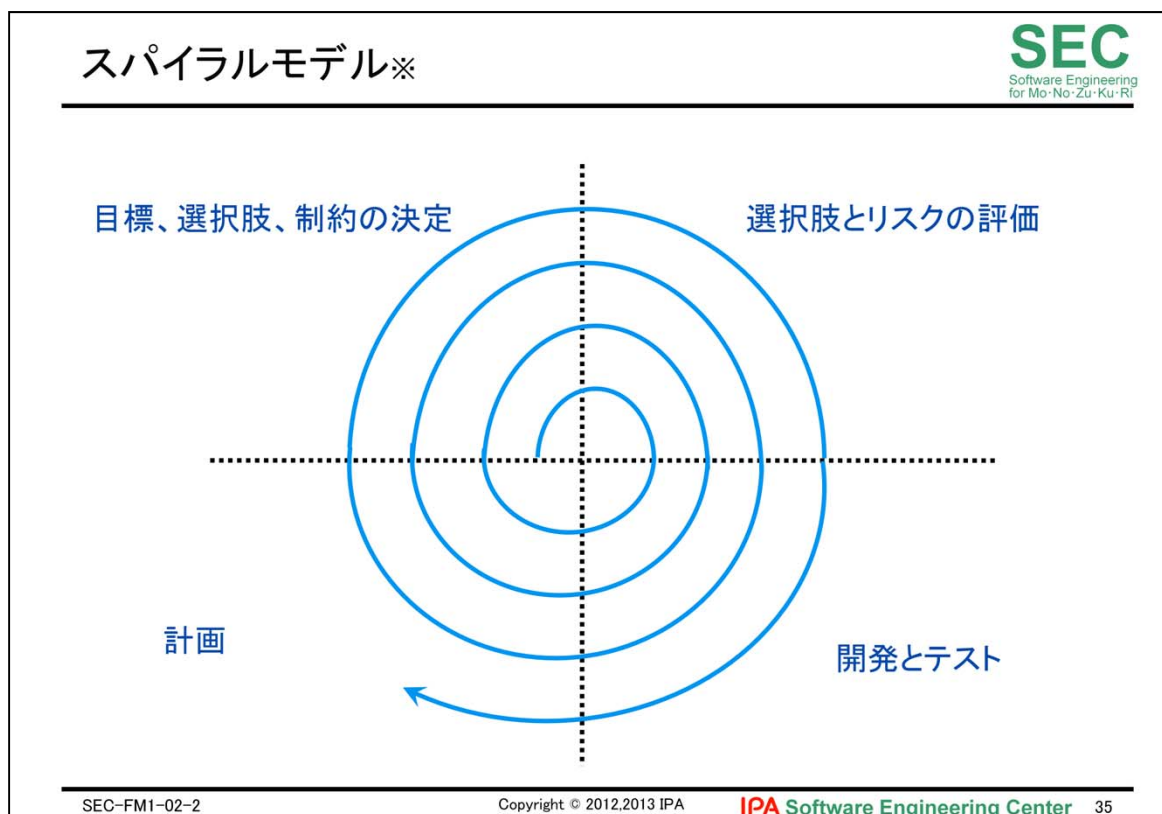
現在は、設計とテストの過程で使われることが多い。



(※このスライドは管理者コースや初回学習時ではスキップも可能)

V字モデルは、それぞれのテストで何を確認しているのかという対応がとりやすい。

上流の作業が、後々どう役立つのかということを見通して形式手法を導入するとよい。



(※このスライドは管理者コースや初回学習時ではスキップも可能)

スパイラルな開発過程でも、形式手法の導入は可能である。

この場合、形式手法の数学的モデルを利用した、モデルベース開発との相性がよい。

## 上流工程の重要性



### ■ 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%

SEC-FM1-02-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center 36

数字は昔のアメリカでのシステム開発の失敗プロジェクトで、その原因を調査したもの。

要求工程での不備を無くしておかないと、不備が後工程に引き継がれて修正コストがどんどん大きくなる。

【さらに学習したい人に】

[NIST:02]

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

## 失敗の原因※



- 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？

(※このスライドは管理者コースや初回学習時にはスキップも可能)

「要求」と書かれているものはもちろん、利用者とのコミュニケーション不足、非現実的な期待、もはや必要とされないシステムも要求に関わる話である。

# サマリー

## 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



※

SEC  
Software Engineering  
for Mo-No-Zu-Ku-Ri

付 録

SEC-FM1-02-2      Copyright © 2012,2013 IPA      IPA Software Engineering Center      40

(※このスライドは管理者コースや初回学習時にはスキップも可能)

## 形式手法適用レベル※



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

SEC-FM1-02-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center 41

(※このスライドは管理者コースや初回学習時ではスキップも可能)

「なぜ形式手法か」のスライドにもあったように形式手法の導入には、いくつかのレベルがある。

記述するだけのレベル 0 でも効果があり、むしろコストパフォーマンスは高い。

レベル 2 にはレベル 1 が、レベル 1 にはレベル 0 が前提となる。

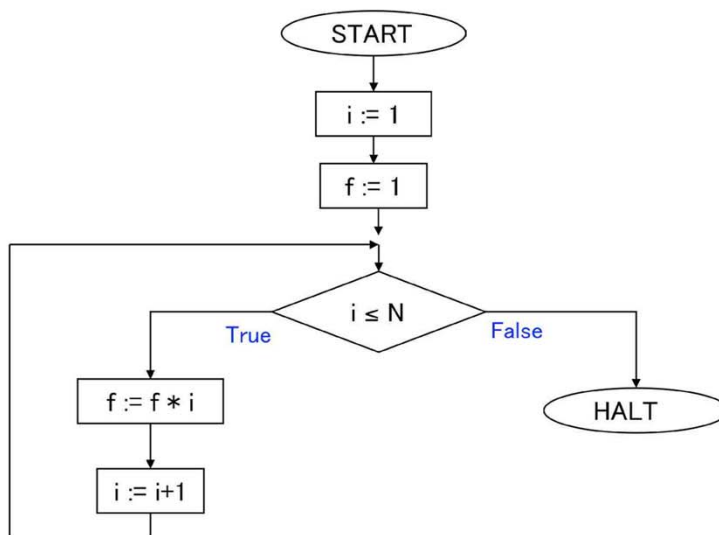
レベルが進むにつれ確信度は高くなるが、そのコストも高くなる。

(例え簡単なプログラムでも証明はそれなりにコストがかかる)

## 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？



SEC-FM1-02-2

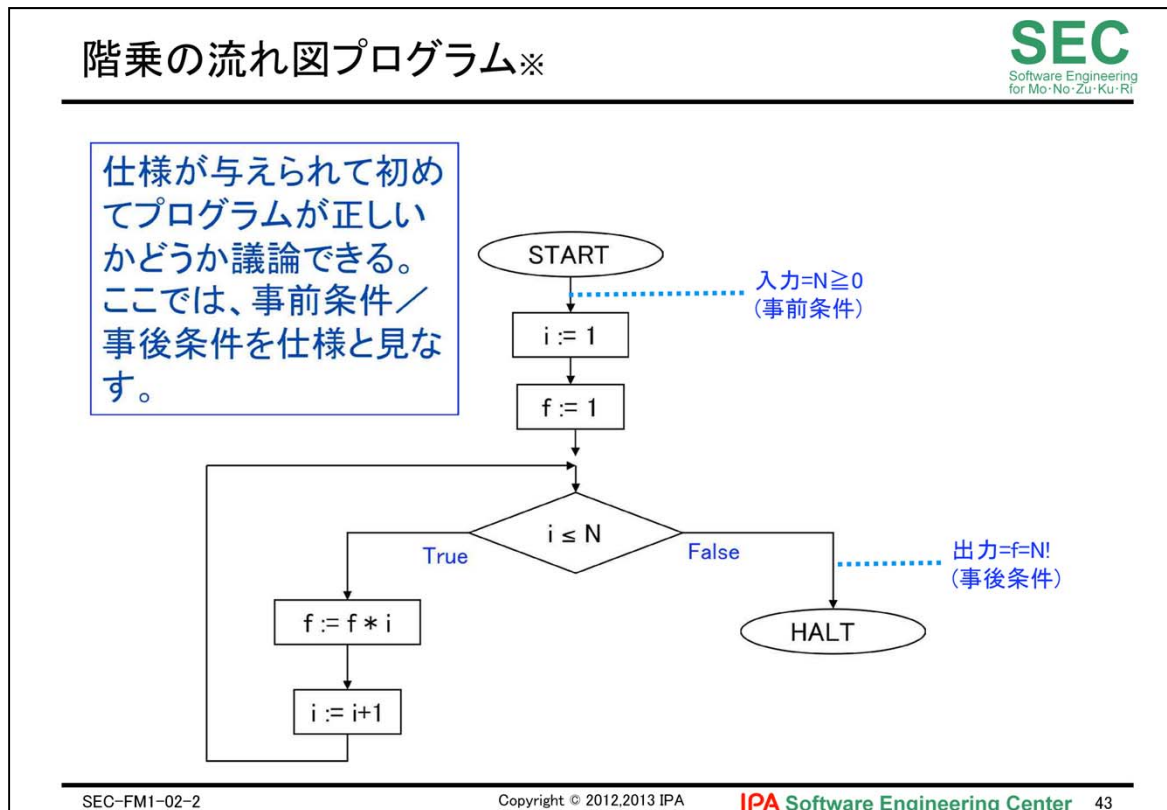
Copyright © 2012,2013 IPA

IPA Software Engineering Center

42

(※このスライドは管理者コースや初回学習時にはスキップも可能)

このプログラムが正しいかどうかはこれだけでは分からない。



(※このスライドは管理者コースや初回学習時ではスキップも可能)

プログラムが正しいかどうかは、その正しさの基準があって、プログラムがその基準を満足しているかどうかで決まる。

プログラムが正しいかどうかの基準は、仕様が基準となる。

例題は 1967年にフロイドが提示した方法によるプログラムの証明である。

事前条件を満足する入力データあるいは初期状態に対してプログラムを実行すると、必ず事後条件を満足する出力あるいは最終状態が得られる。

これが、プログラムが正しい ということである。

この例では、0 以上の整数  $N$  が入力として与えられると、

$N$  の階乗  $N!$  が得られて、答  $f$  となる ということが、

このプログラムが正しいということである。

このように、プログラムの正しさを証明するために形式手法は発展してきた。

## 階乗に関するホーアの論理※



```
{N ≥ 0}

i := 1;
f := 1;
While i ≤ N
Do
  f := f * i;
  i := i + 1
End

{f = N!}
```

SEC-FM1-02-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center 44

(※このスライドは管理者コースや初回学習時ではスキップも可能)

1969 年ホーアは、テキスト形式のプログラムに対する、プログラムの正しさの証明法を数学の論理学の枠組みの中で提示した。

見かけは、入力データに対する制約がコメントとして書かれており、結果に対して、最後の記述を満足する結果が返るといったコメントの形で書いてある。

これが、実は論理式である。


部分的正当性の表明と呼ばれる拡張された論理式である。

入力が 0,1,2,3,...と試して、結果と照らし合わせプログラムを確認できる。

これは仕様に対するテストであるが、確認すべきデータは無限にある。

数学的な証明をすると、有限の手順で無限のデータに対する正しさを保証できる。

これがテストと証明の違いである。


  
 Software Engineering  
 for Mo-No-Zu-Ku-Ri

## 証明図(これが証明です!) ※

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

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

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

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

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

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

公理から出発して、推論規則に基づいて、最下段の結論を導出する。

これらは、真であることは容易に示すことができる。

もともと真である公理と、これら真である論理式を前提として。

推論規則に従って真である結論を導き出す。



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