高レジリエンスマイコンFUJIMIへの
モデル検査適用事例

株式会社フォーマルテック　早水 公二

【自己紹介】

早水　公二（はやみず　こうじ）　１９７０年大阪生まれ

１９９４年	京都工芸繊維大学　電子情報工学科　卒業
同年	メルコ・パワー・システムズ（MPS）入社
２００２年～	産業技術総合研究所でモデル検査技術を習得
２００４年～	関西電力株式会社様とモデル検査の普及活動を開始
２０１１年～	■株式会社フォーマルテックを設立 形式手法(特にモデル検査)の本格的な普及活動を開始
	■産業技術総合研究所 組込みシステム技術連携研究体　契約職員 （２０１４年退職 → 自社業務に専念）

【本日の発表内容】

1. FUJIMI(適用対象)の概要
2. モデル検査とは？
3. 詳細設計書で不具合を発見した事例
4. 仕様書間での不整合を発見した事例
5. モデル検査の適用プロセス
6. モデル検査報告書

1. FUJIMI(適用対象)の概要

機能喪失からの高確率での回復(高レジリエンス性)を目指したマイコンシステム　㈱エルイーテック様で開発中

□暴走したマイコンはリセットするしかない
　しかし、マイコン全体をリセットする必要はない

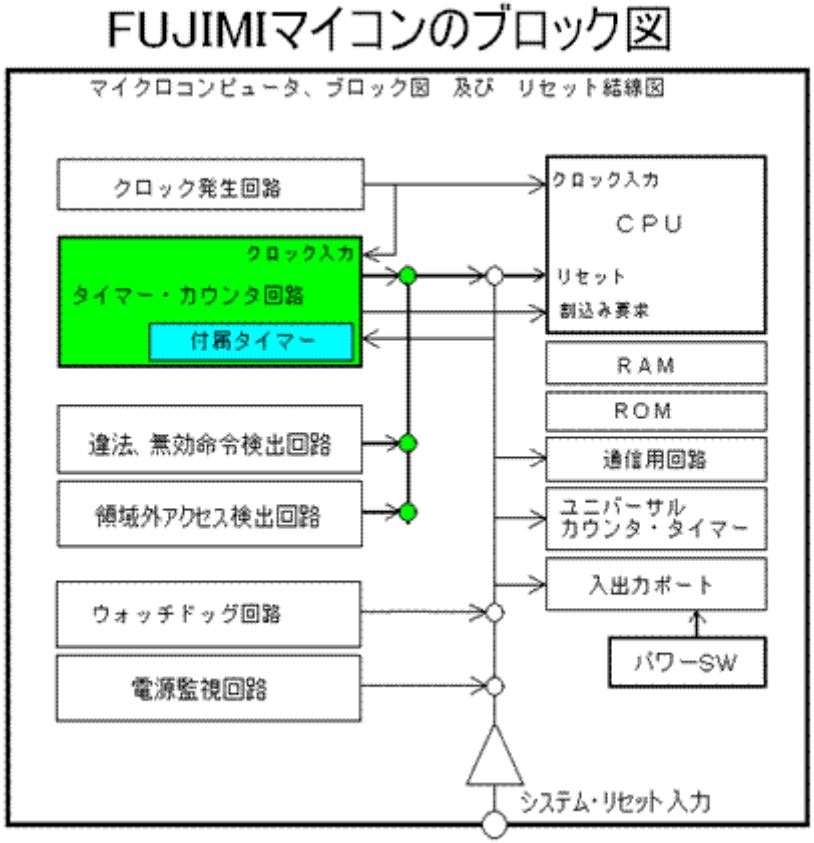
□機能不全を起こしているのは主にCPUである
　では、CPUだけリセットすればよい！
　　・RAM上の情報のほとんどは正しいはず(経験則)
　　・I/Oはソフトウェアで再設定できる

□周期的にCPUをリセットすれば暴走は見えなくなる

□しかしリセットするとソフトウェアの実行に問題

結論！　リセットを割込処理の一部にすれば問題なし

1. FUJIMI(適用対象)の概要



リセットが２系統ある

① 通常のシステム・リセット

- □電源監視
- □WatchDog Timer

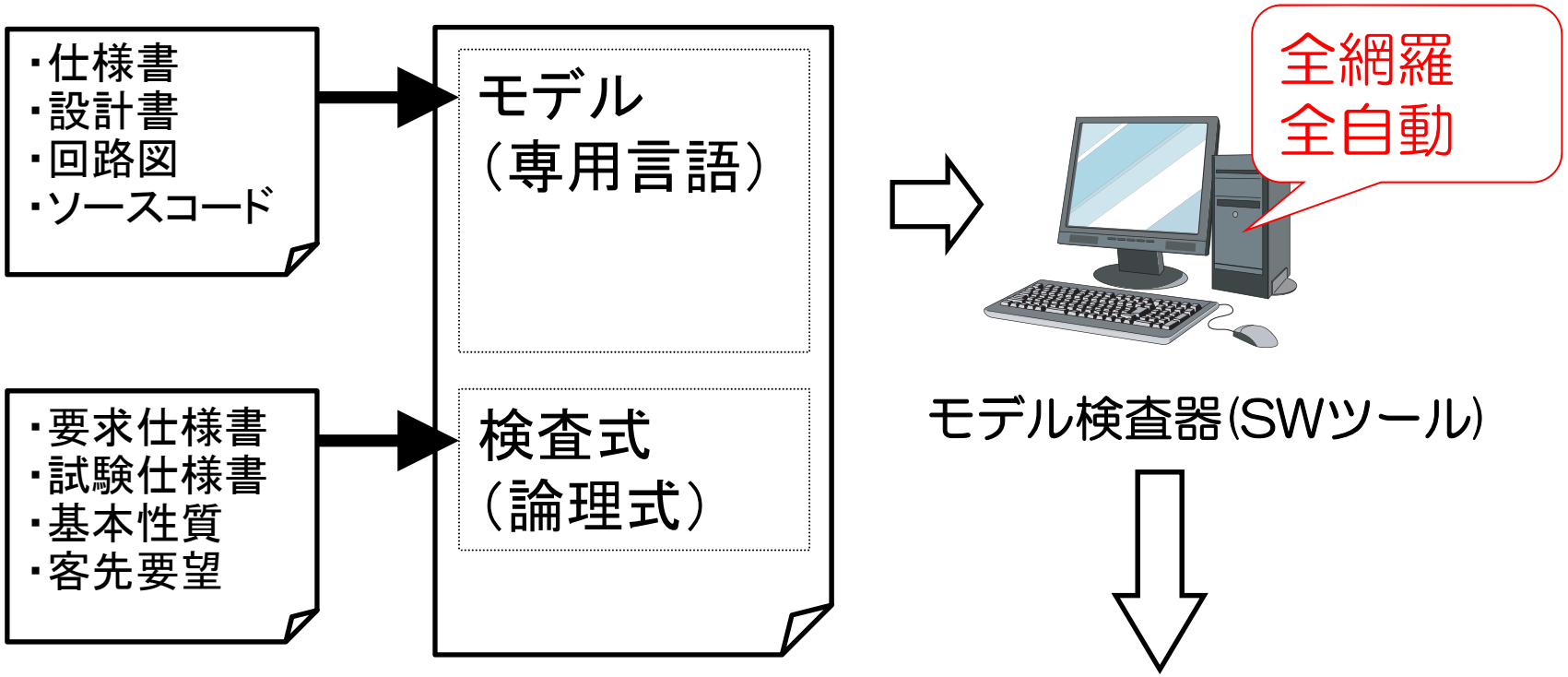
② CPUコア専用のリセット

- □命令実行の異常を検出
- □違法命令の検出
- □無効な空間のアクセス
- □定時タイマ

デモ

2. モデル検査とは？

「専用言語」で記述されたシステムを全自動で網羅的に検査する



【検査結果】

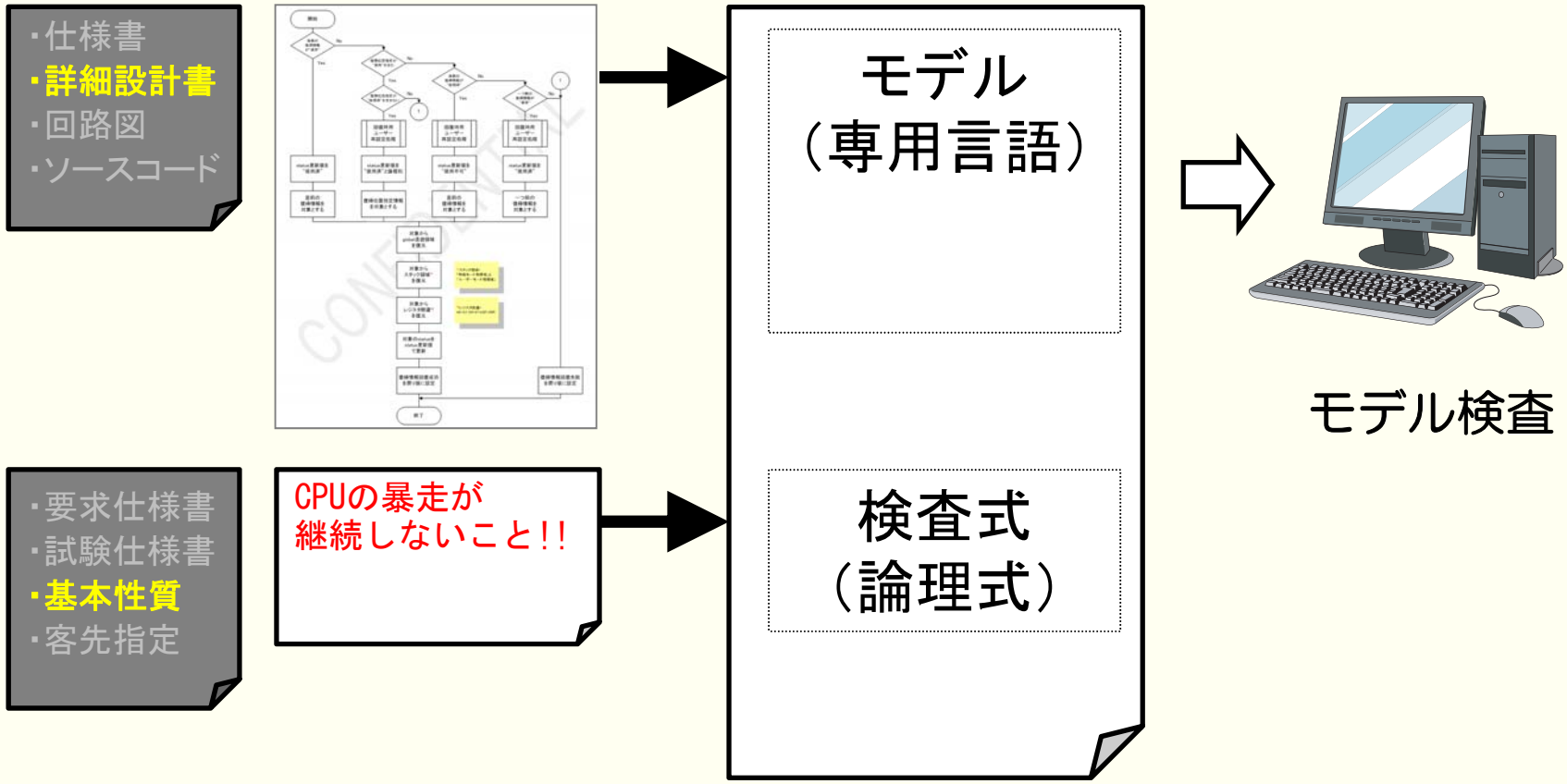
モデルが検査項目を満たす場合「True」

満たさない場合「False」＋「反例」

3. 詳細設計書で不具合を発見した事例

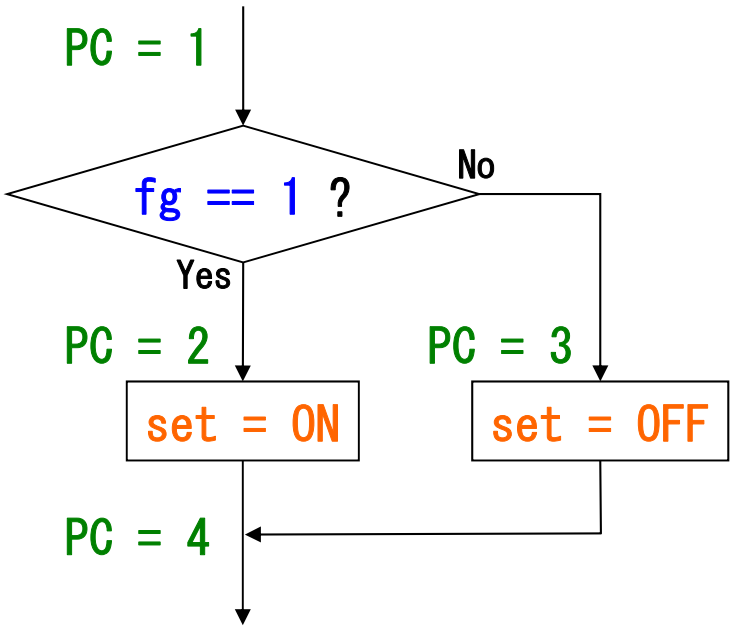
3.1 モデルと検査式

詳細設計書の内容が基本性質を満たしているか否かの検査



3. 詳細設計書で不具合を発見した事例

3.2 詳細設計書のモデル化



フローチャート

```
next(PC) := case
  PC = 1 &   fg = 1   : 2;
  PC = 1 & !(fg = 1)  : 3;
  PC = 2 | PC = 3     : 4;
           ⋮
  TRUE : PC;
esac;

next(set) := case
  PC = 2 : ON;
  PC = 3 : OFF;
  TRUE : set;
esac;
```

モデルのコード

3. 詳細設計書で不具合を発見した事例

3.3 検査式の作成

検査項目 ： CPUの暴走は継続しない（発生するが続かない）

検査式に変換

事前変換 ： 「CPU = 暴走状態」である状態が継続しない

検査式 ： !EF (EG (CPU = 999))

EF(P) ： Exist Future(P)

将来Pとなることが在る

→ 否定(!)があるので「無い」

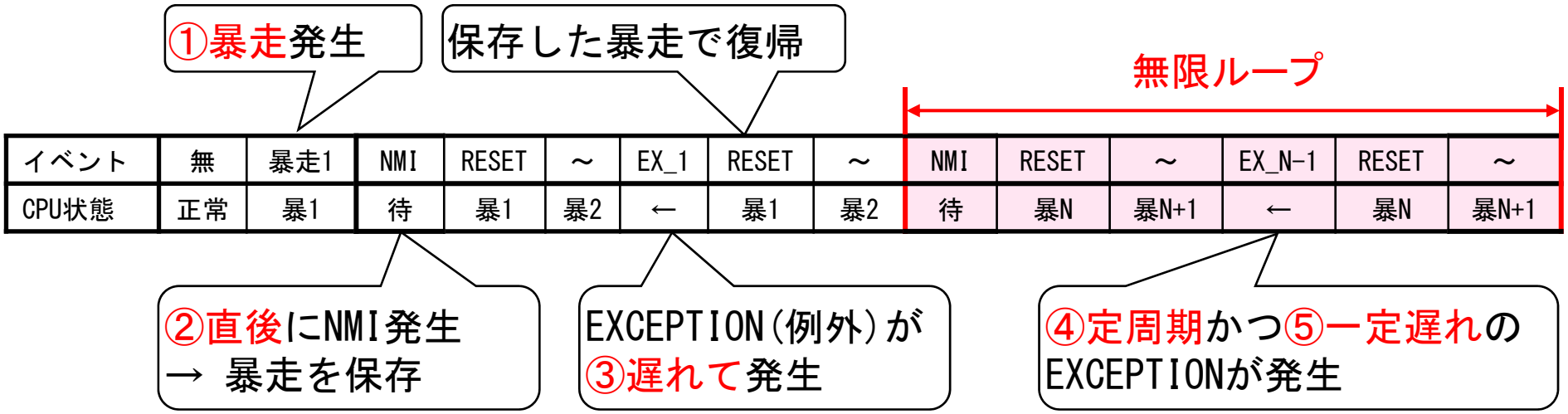
EG(Q) ： Exist Globally(Q)

ずっとQで在り続ける

3. 詳細設計書で不具合を発見した事例

3.4 発見した不具合

検査項目 ： CPUが暴走し続けないこと
反例　　 ： 5つの稀なイベントが連続した場合に暴走し続ける



網羅的検査だからこそ見つかった反例

億～兆の状態の検査は人手では不可能

EXCEPTIONの遅延は無いことを前提としてWatchDog Timerで対応
→ 次の検査へ

3. 詳細設計書で不具合を発見した事例

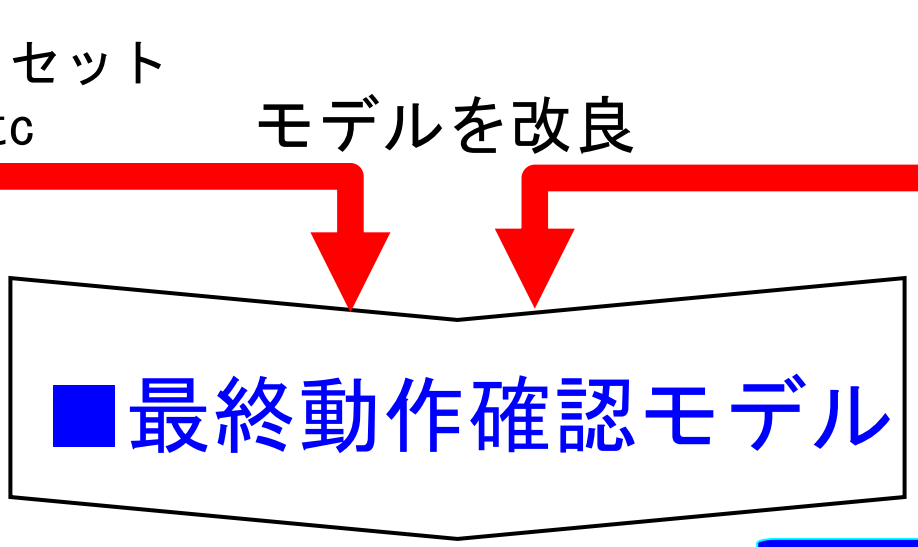
3.5 最終動作確認

前ページで・・・

→ WatchDog Timerでの対応に決定（EXCEPTIONは遅延しない）

WatchDog等によるリセット
予期せぬリセットetc

モデルを改良



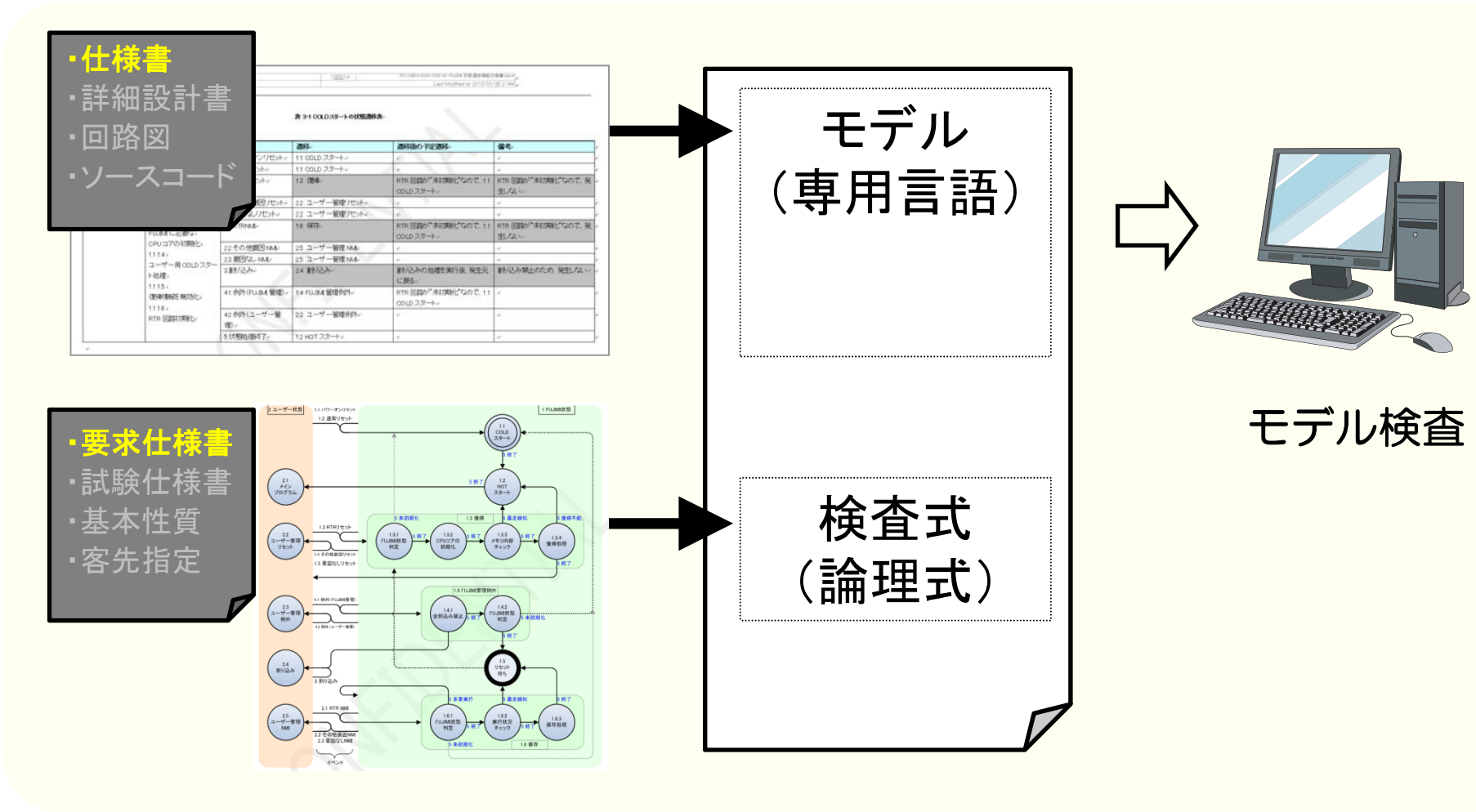
再検査 ： CPUの暴走が継続しない ⇨ TRUE!!

所要時間　　：150[分]
使用メモリ　　：約15.8[GByte] ）
到達可能状態数 ：数百億[状態](BDDノード数より推定)

4. 仕様書間での不整合を発見した事例

3.1 モデルと検査式

状態遷移表(下流)と状態遷移図(上流)での整合性の検査



4. 仕様書間での不整合を発見した事例

4.2 状態遷移表(下流側)のモデル化

EVENT \ STATE	1	2	3	4
A	→3			→1 fg = 0
B		→4 fg = 1	→2	→1 fg = 0

状態遷移表

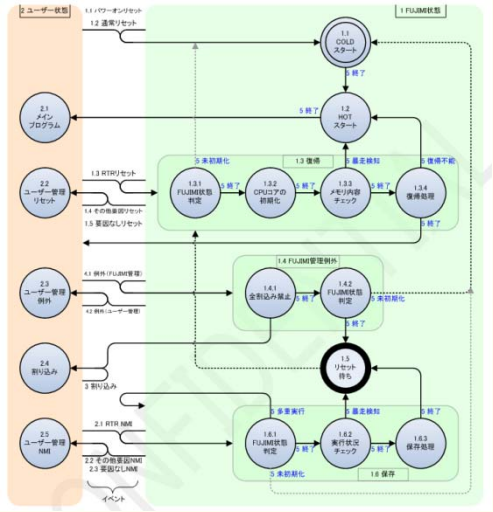
状態1でイベントAが起これば
状態3に遷移する

```
next(STATE) := case
    STATE = 1 & EVENT = A : 3;
    STATE = 2 & EVENT = B : 4;
    STATE = 3 & EVENT = B : 2;
    STATE = 4 : 1;
    TRUE : STATE;
esac;

next(fg) := case
    STATE = 2 & EVENT = B : 1;
    STATE = 4 : 0;
    TRUE : fg;
esac;

next(EVENT) := {A, B};
```

モデルのコード

4. 仕様書間での不整合を発見した事例

4.3 検査式の作成(上流側の状態遷移図から)

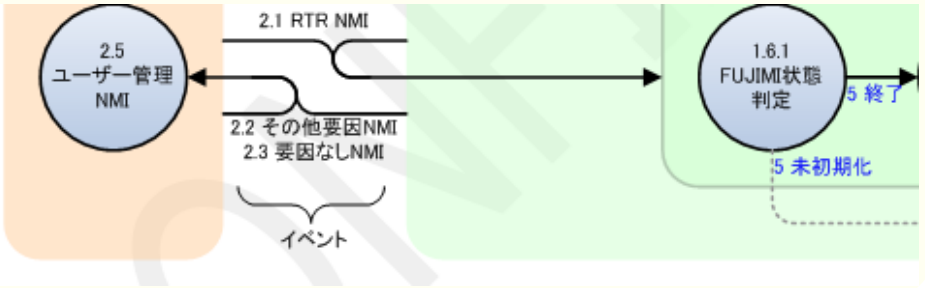


２９式の検査式

```
SPEC AG(((STATE_1     | STATE_2) & EVENT = 11) -> AX(STATE_11  ))
SPEC AG(((STATE_1     | STATE_2) & EVENT = 12) -> AX(STATE_11  ))
SPEC AG(((STATE_1     | STATE_2) & EVENT = 13) -> AX(STATE_13  ))
SPEC AG(((STATE_1     | STATE_2) & EVENT = 14) -> AX(STATE = 22))
SPEC AG(((STATE_1     | STATE_2) & EVENT = 15) -> AX(STATE = 22))
SPEC AG(((STATE_1     | STATE_2) & EVENT = 21) -> AX(STATE_16  ))
SPEC AG(((STATE_1     | STATE_2) & EVENT = 22) -> AX(STATE = 25))
SPEC AG(((STATE_1     | STATE_2) & EVENT = 23) -> AX(STATE = 25))
SPEC AG(((            | STATE_2) & EVENT = 3 ) -> AX(STATE = 24))
SPEC AG(((STATE = 141         ) & EVENT = 3 ) -> AX(STATE = 24))
SPEC AG(((STATE_1     | STATE_2) & EVENT = 41) -> AX(STATE_14  ))
SPEC AG(((STATE_1     | STATE_2) & EVENT = 42) -> AX(STATE = 23))
SPEC AG(((STATE_11            ) & EVENT = 5                  ) -> AX(STATE_12   ))
SPEC AG(((STATE_12            ) & EVENT = 5                  ) -> AX(STATE =  21))
SPEC AG(((STATE = 131         ) & EVENT = 5 & RTR_INIT = SUMI) -> AX(STATE = 132))
SPEC AG(((STATE = 131         ) & EVENT = 5 & RTR_INIT = MADA) -> AX(STATE_11  ))
SPEC AG(((STATE = 132         ) & EVENT = 5                  ) -> AX(STATE = 133))
SPEC AG(((STATE = 133         ) & EVENT = 5 & MEM_CHK = SEIJO) -> AX(STATE = 134))
SPEC AG(((STATE = 133         ) & EVENT = 5 & MEM_CHK = IJO  ) -> AX(STATE_12  ))
SPEC AG(((STATE = 134         ) & EVENT = 5 & FK_DATA = ARI  ) -> AX(STATE_2   ))
SPEC AG(((STATE = 134         ) & EVENT = 5 & FK_DATA = NASHI) -> AX(STATE_12  ))
SPEC AG(((STATE = 141         ) & EVENT = 5                  ) -> AX(STATE = 142))
SPEC AG(((STATE = 142         ) & EVENT = 5 & RTR_INIT = SUMI) -> AX(STATE_15  ))
SPEC AG(((STATE = 142         ) & EVENT = 5 & RTR_INIT = MADA) -> AX(STATE_11  ))
SPEC AG(((STATE = 161         ) & EVENT = 5 & RTR_INIT = SUMI ) -> AX(STATE = 162))
SPEC AG(((STATE = 161         ) & EVENT = 5 & RTR_INIT = MADA ) -> AX(STATE_11  ))
SPEC AG(((STATE = 162         ) & EVENT = 5 & JOKYO_FG = SEIJO) -> AX(STATE = 163))
SPEC AG(((STATE = 162         ) & EVENT = 5 & JOKYO_FG = IJO  ) -> AX(STATE_15  ))
SPEC AG(((STATE = 163         ) & EVENT = 5                  ) -> AX(STATE_15  ))
```

検査式の例　SPEC AG(((STATE_1 | STATE_2) & EVENT = 21) -> AX(STATE_16))

FUJIMI状態あるいはユーザー状態で、
RTR NMI(イベント21)が入力されると
必ず状態 1.6 FUJIMI状態判定に遷移する



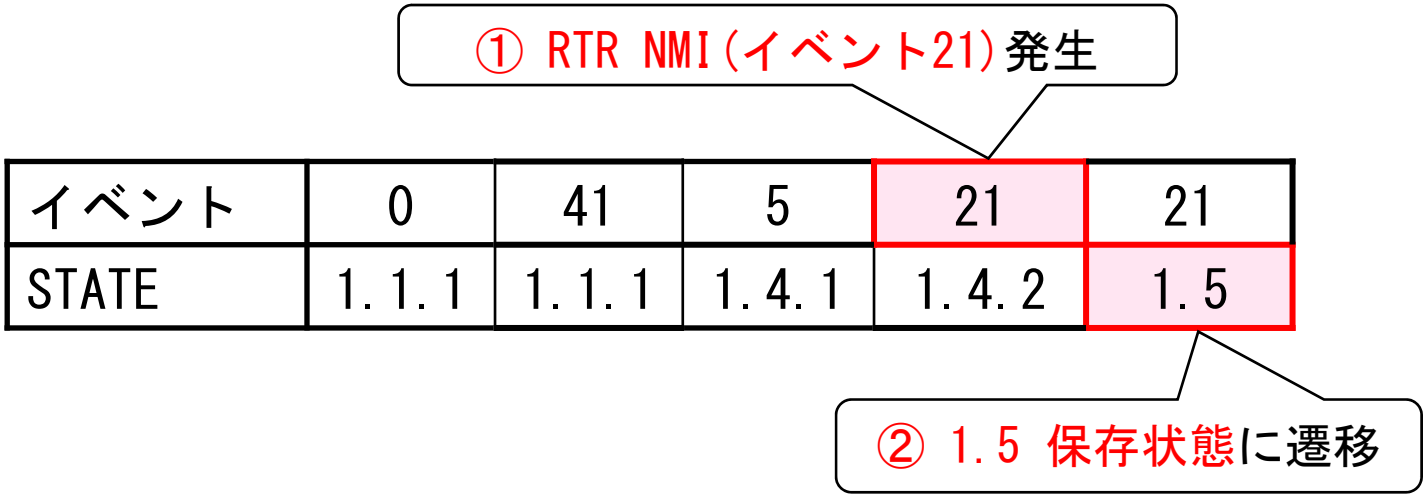
4. 仕様書間での不整合を発見した事例

4.4 検査結果

検査結果は？ → 29式の検査式のうち1式でFALSE

検査項目 ： FUJIMI状態 or ユーザ状態で割込み(イベント21)が発生すると
必ず状態 1.6 FUJIMI状態判定に遷移する

反例 ： 1.6 FUJIMI状態判定ではなく1.5 保存状態に遷移してしまう



1箇所でも記述ミスがあれば必ず発見

人間は見落とすことがある

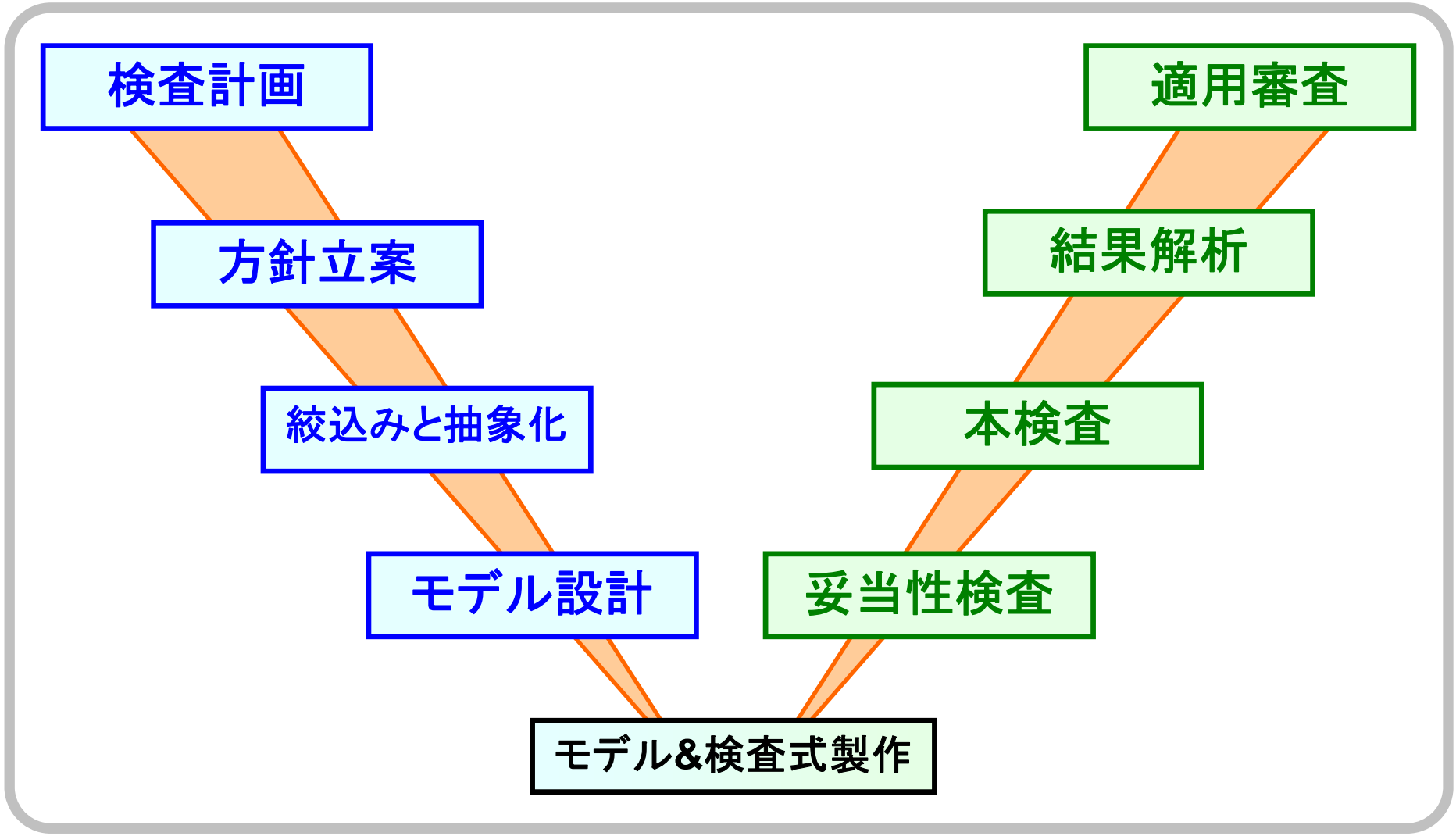
4. 仕様書間での不整合を発見した事例

4.5 発見した不整合

		3割り込み	24 割り込み	割り込みの処理を実行後、発生元に戻る	
		4.1例外(FUJIMI管理)	1.4 FUJIMI管理例外	RTR回路が"未初期化" 1.1 COLDスタート RTR回路が"初期化済" FUJIMI管理例外の処理を実行後、1.5 リセット待ち	
		4.2例外(ユーザー管理)	23 ユーザー管理例外		
		5.状態処理終了	1.4.2 FUJIMI状態判定		
1.4.2 FUJIMI状態判定	1.4.2.1 RTR回路が未初期化? 1.4.2.2 FUJIMI実行中フラグを"実行中"	1.1パワーオンリセット	1.1 COLDスタート		
		1.2通常リセット	1.1 COLDスタート		
		1.4.その他要因リセット	[illegible]リセット		
		1.5.要因なしリセット	22 [illegible]ユーザー管理リセット		
		2.1 RTRNMI	1.5 保存	RTR回路が"未初期化" 1.1 COLDスタート RTR回路が"初期化済" RTR実行中フラグが"実行中"または FUJIMI実行中フラグが"実	

誤) 1.5 保存
正) 1.6 FUJIMI状態判定

5. モデル検査の適用プロセス



5. モデル検査の適用プロセス

なぜプロセスが必要か？

モデル検査の最大の**課題**（運用面で）

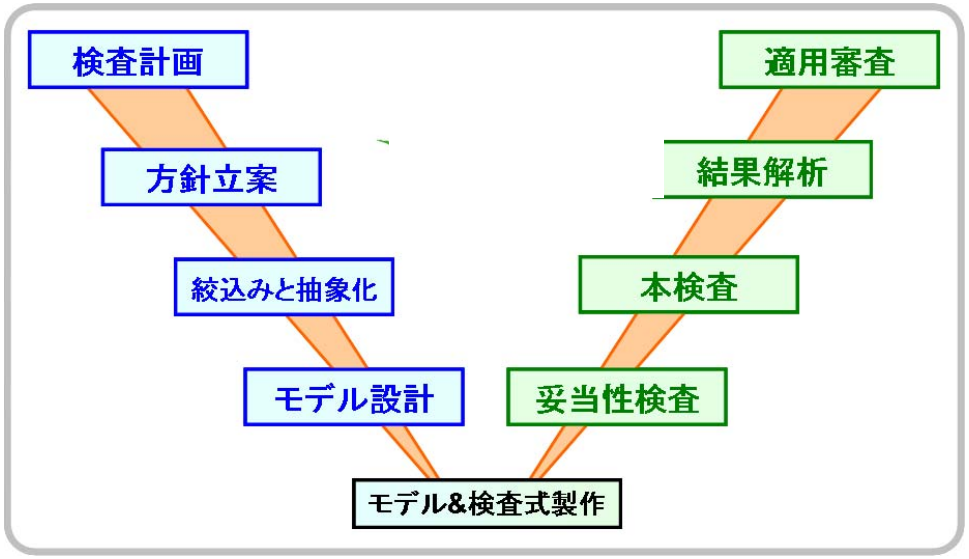
初心者「何から始めてどうしたらいいのか？」
依頼者「何をしてくれるのか？結果は？」

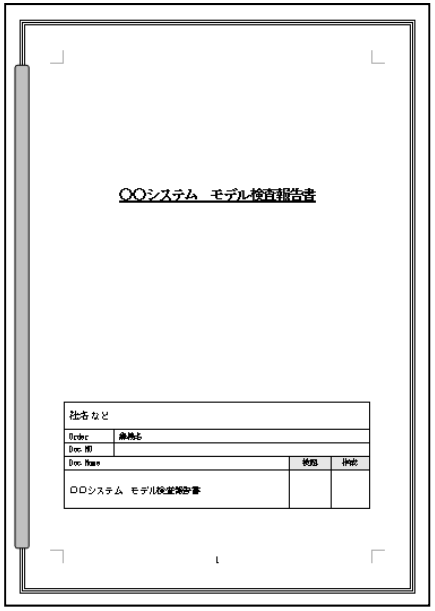
■適用プロセスの確立
■アウトプットの定義 → モデル検査報告書

6. モデル検査報告書

本事例では140頁以上の報告書を作成

■モデル検査適用プロセスに沿って作業
■アウトプットを明確化 → モデル検査報告書





モデル検査報告書

ご清聴ありがとうございました。