組み込み分野へのソフトウェアエンジニアリングの活用法

2008-8-27

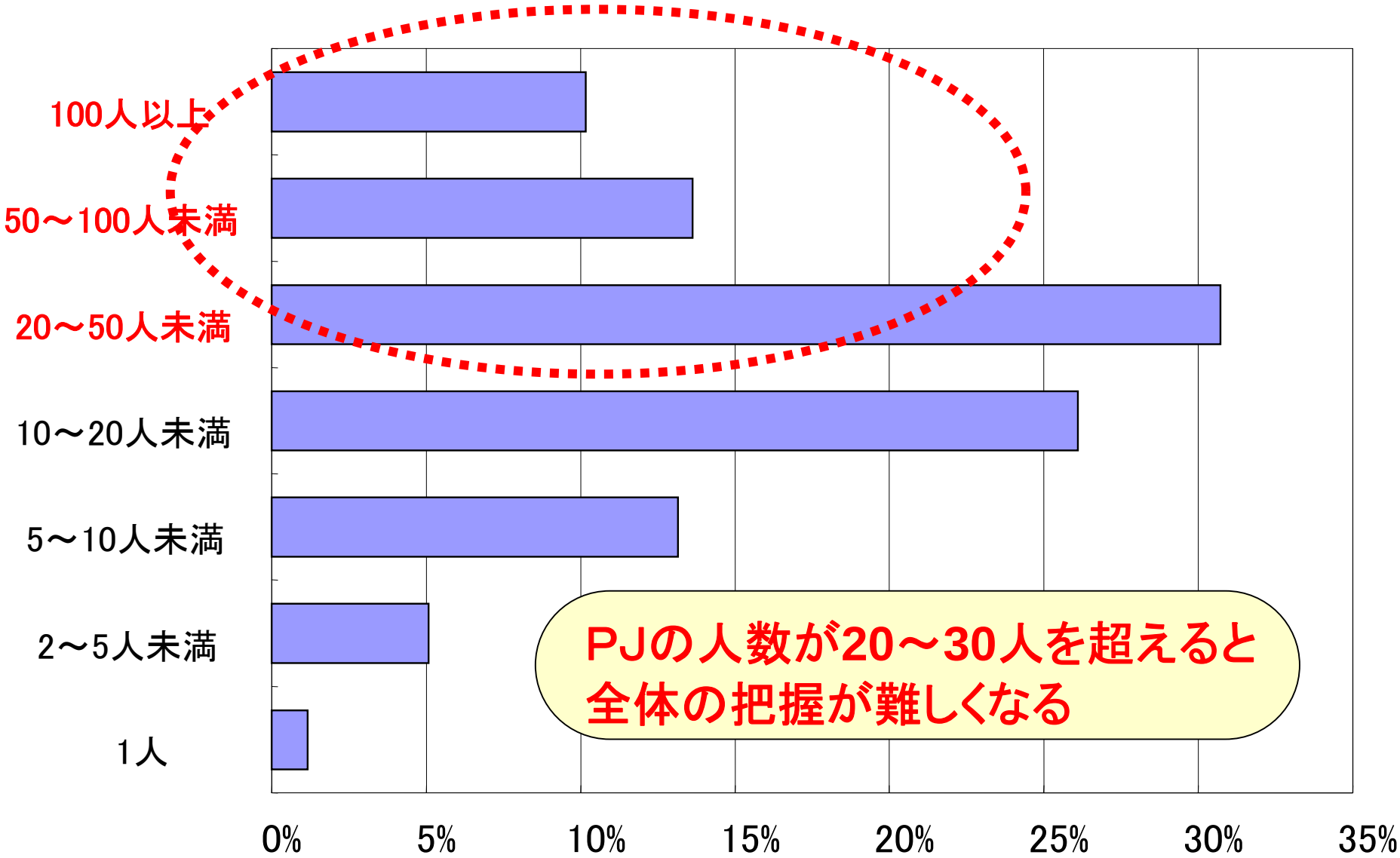
IPA/SEC　平山雅之

Today's Topics

1. 組込みシステム開発の現状
2. ソフトウェアエンジニアリングの活用戦略
3. 「要求仕様の変更」への対応
4. エンジニアリング手法導入の考え方

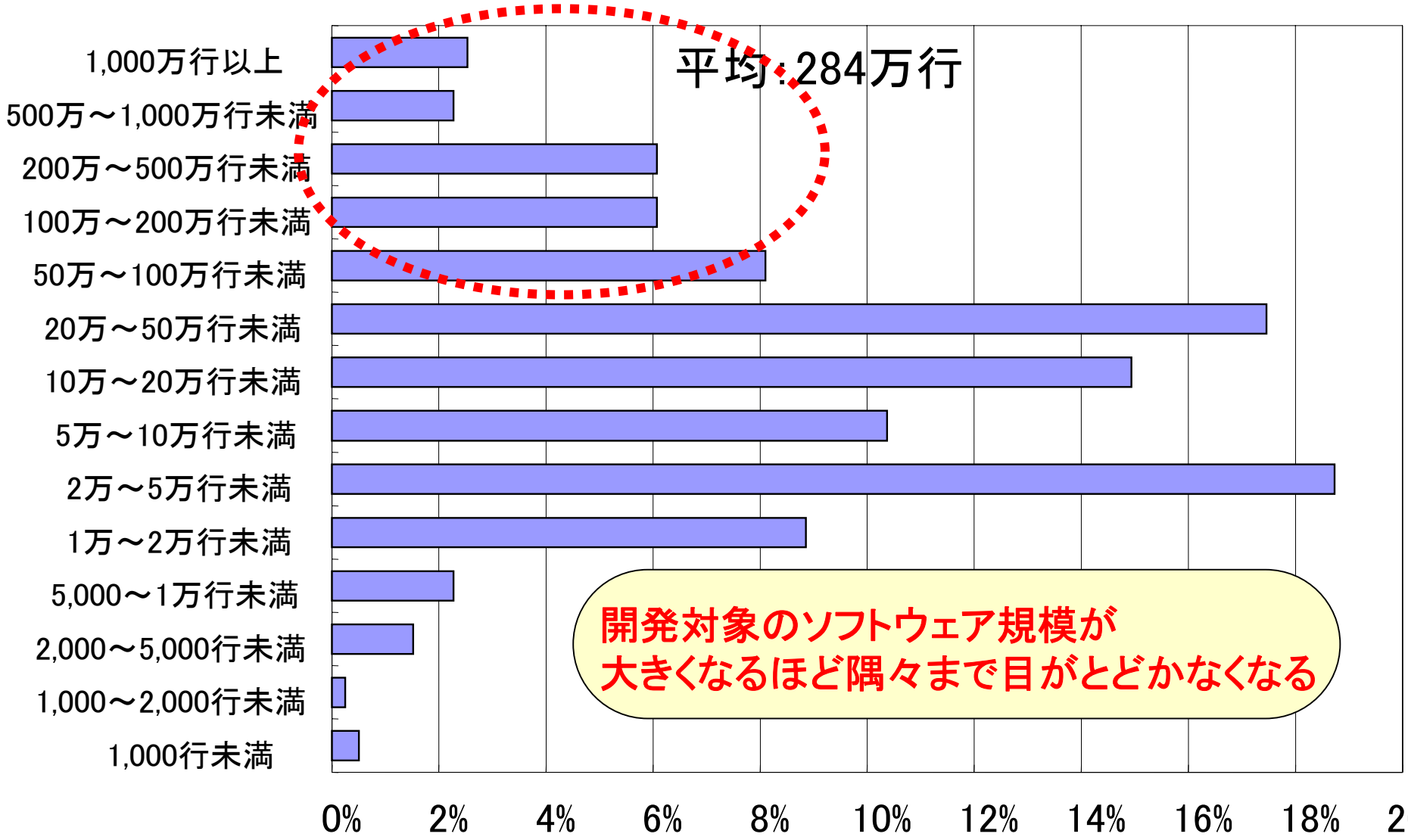
プロジェクトの人数規模：全体（社内+社外）

プロジェクト責任者Q4-2fa



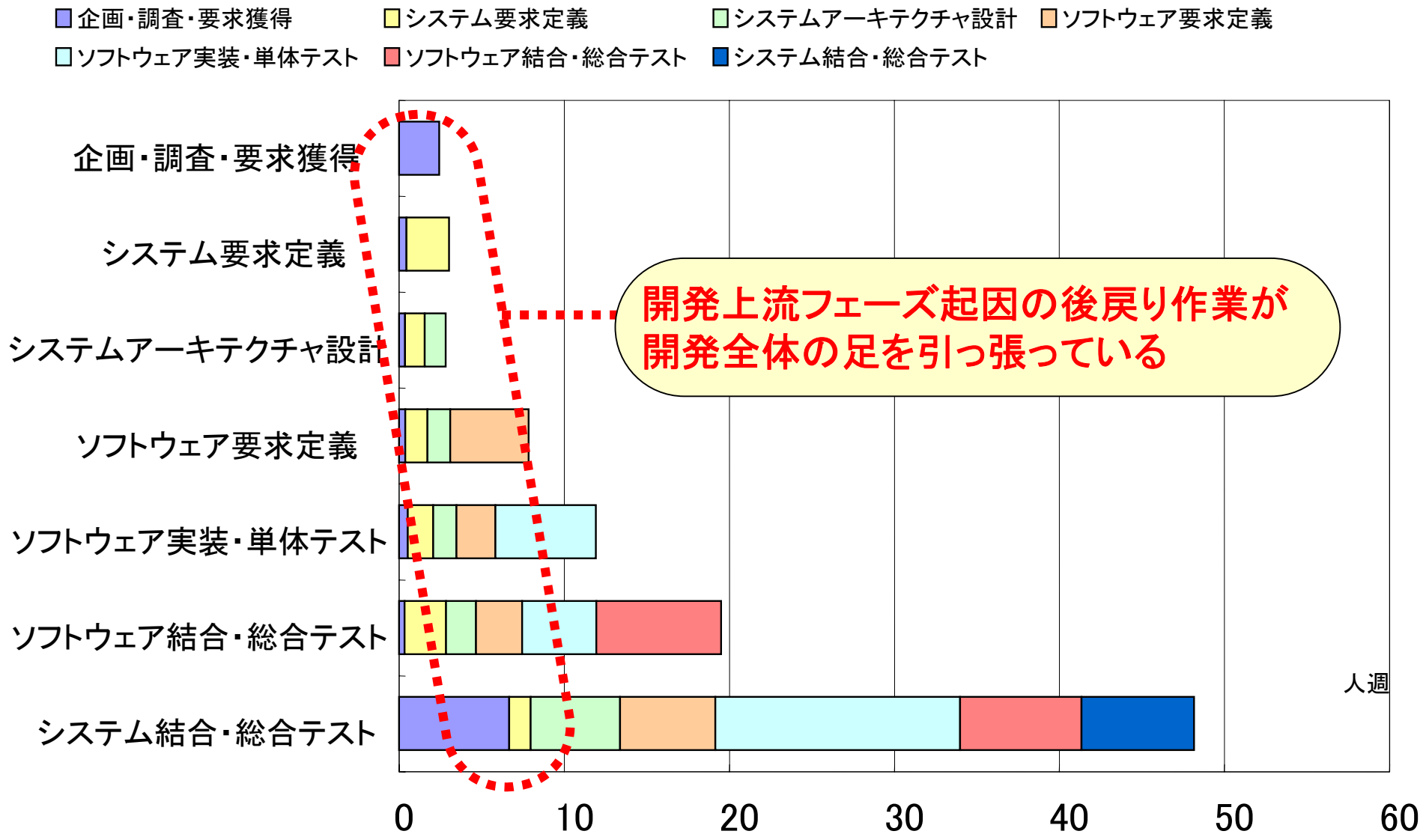
全行数(新規開発と既存の合計)

プロジェクト責任者 Q2-1a



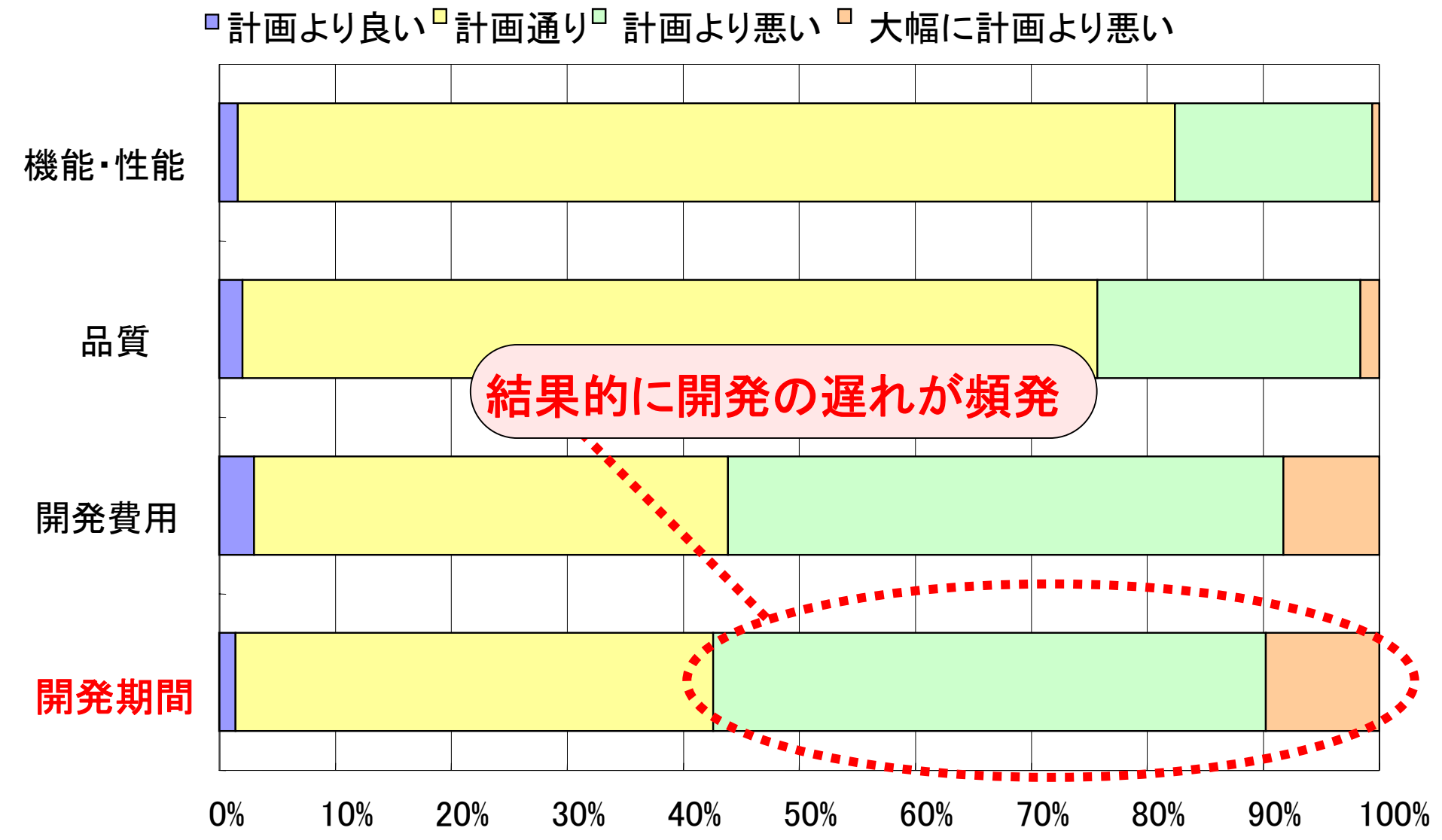
ソフトウェア不具合発見工程別修正工数

プロジェクト責任者　Q6-1b



開発計画に対するプロジェクトの結果

プロジェクト責任者Q8-1



ソフトウェア開発の遅延の仕組み

組込みシステム
　　開発規模の増大
　　プロジェクトの肥大化

急激な増大・肥大化に
開発手法が追いついていない

開発途中での不具合の増加

後戻り作業の増加

開発の遅れ

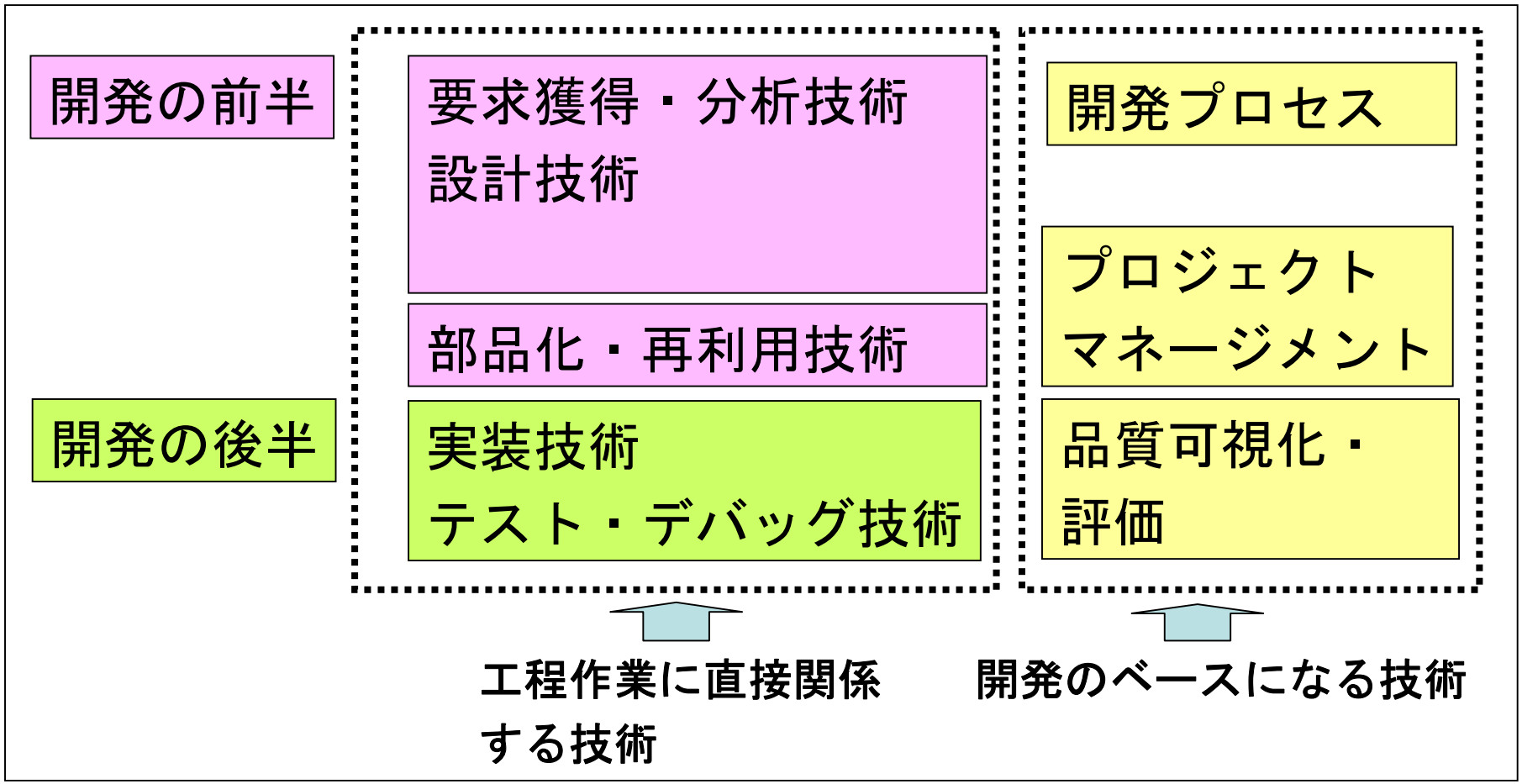
ソフトウェアエンジニアリング
を活用した
システマティックな開発へ

Today's Topics

1. 組込みシステム開発の現状
2. ソフトウェアエンジニアリングの活用戦略
3. 「要求仕様の変更」への対応
4. エンジニアリング手法導入の考え方

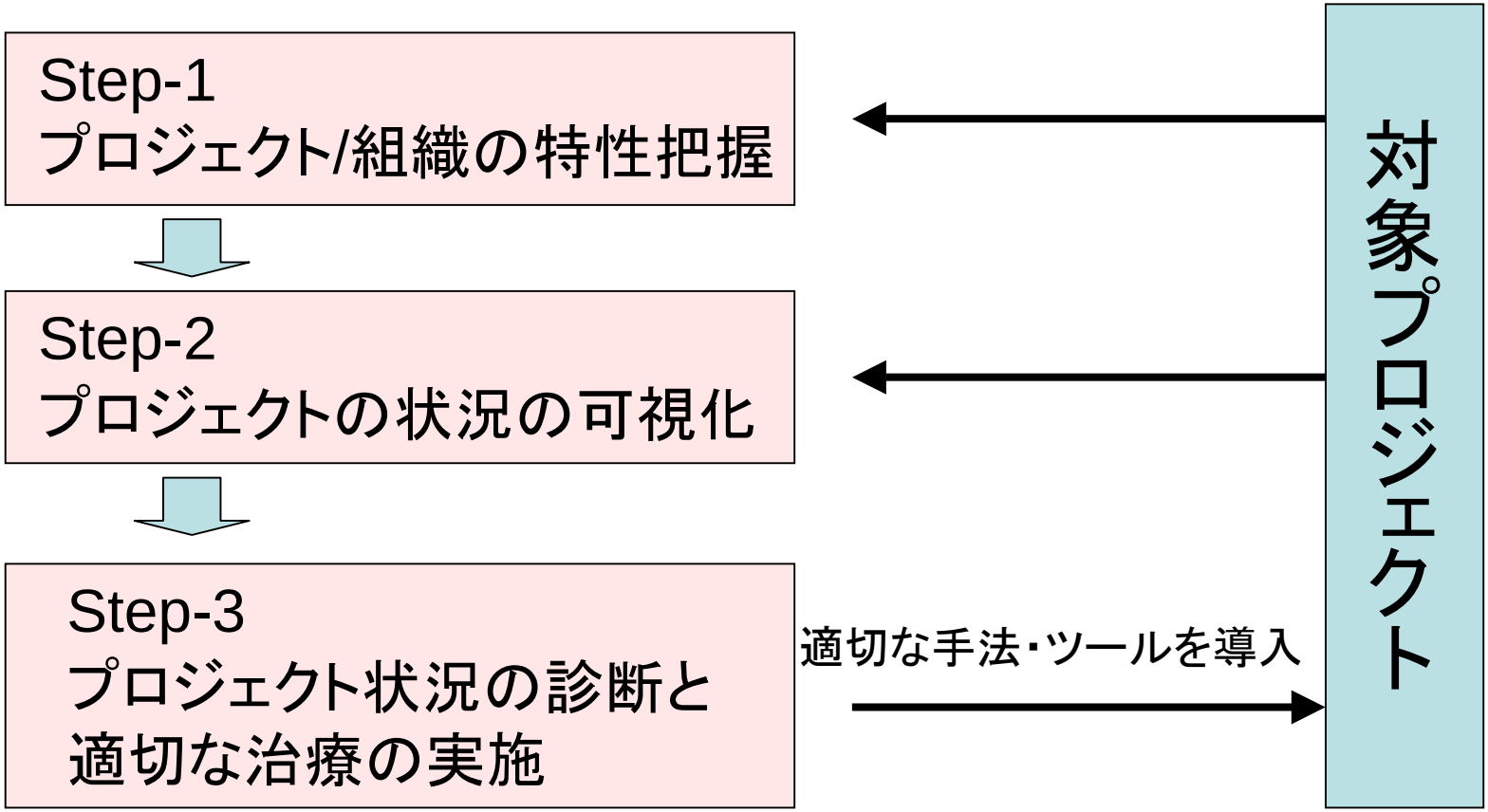
2. ソフトウェアエンジニアリングの活用戦略

ソフトウェアエンジニアリング＝高品質なソフトウェアを効率的に開発するための技術



どの技術をどのように活用するかが大きなポイント！

エンジニアリング手法導入のステップ



Step-1: プロジェクト/組織の特性の把握

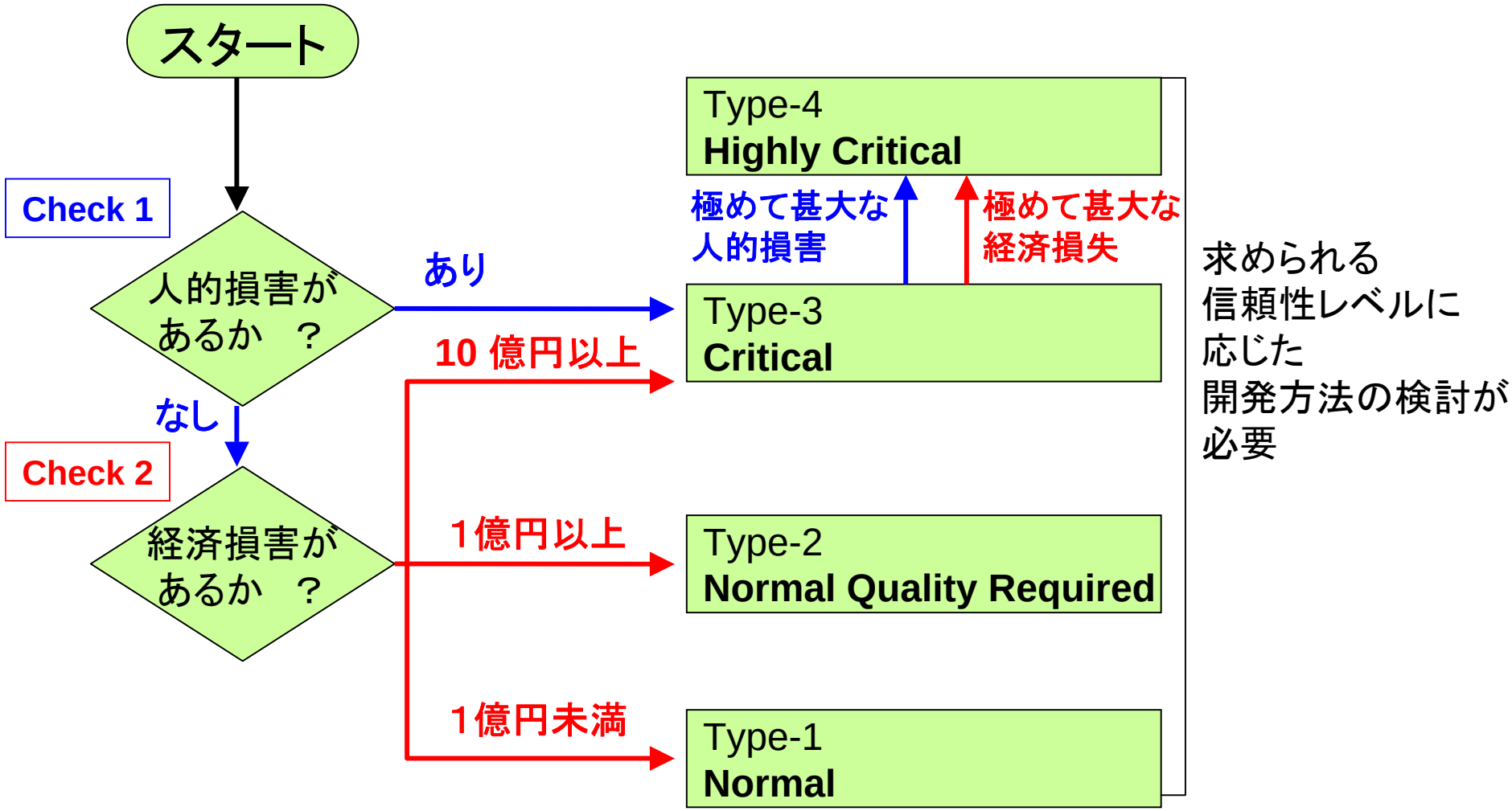
個々のプロジェクト,組織はそれぞれ固有の特性を持つ

- 特性の異なる組織、プロジェクトでの
  ベストプラクティスは役に立つとは限らない
- 組織、プロジェクトで発生する問題の多くは
  個々の特性を背景にする場合が少なくない

プロジェクト/組織の特性を明らかにする必要性

システム特性の把握

システムに求められる信頼性の度合い



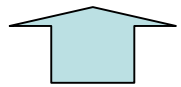
開発ビューから見たシステム特性

①	ソフトウェア規模
②	ソフトウェアの複雑さ
③	システム制約条件の厳しさ
④	仕様の明確化度合い
⑤	再利用するソフトウェアの品質レベル
⑥	開発プロセスの整備度合い
⑦	開発組織の分業化・階層化の度合い
⑧	開発メンバーのスキル
⑨	プロジェクトマネージャの経験とスキル
⑩	システム障害時のメーカ側損失額

対象システムの実装上の特性

組織・プロジェクトの特性

ビジネス面の特性



様々なビューから開発の特性を把握する

Step-2: プロジェクトの状況の可視化

プロジェクトの状況を可視化し
開発のどの部分に問題があるかを把握する

現象の可視化

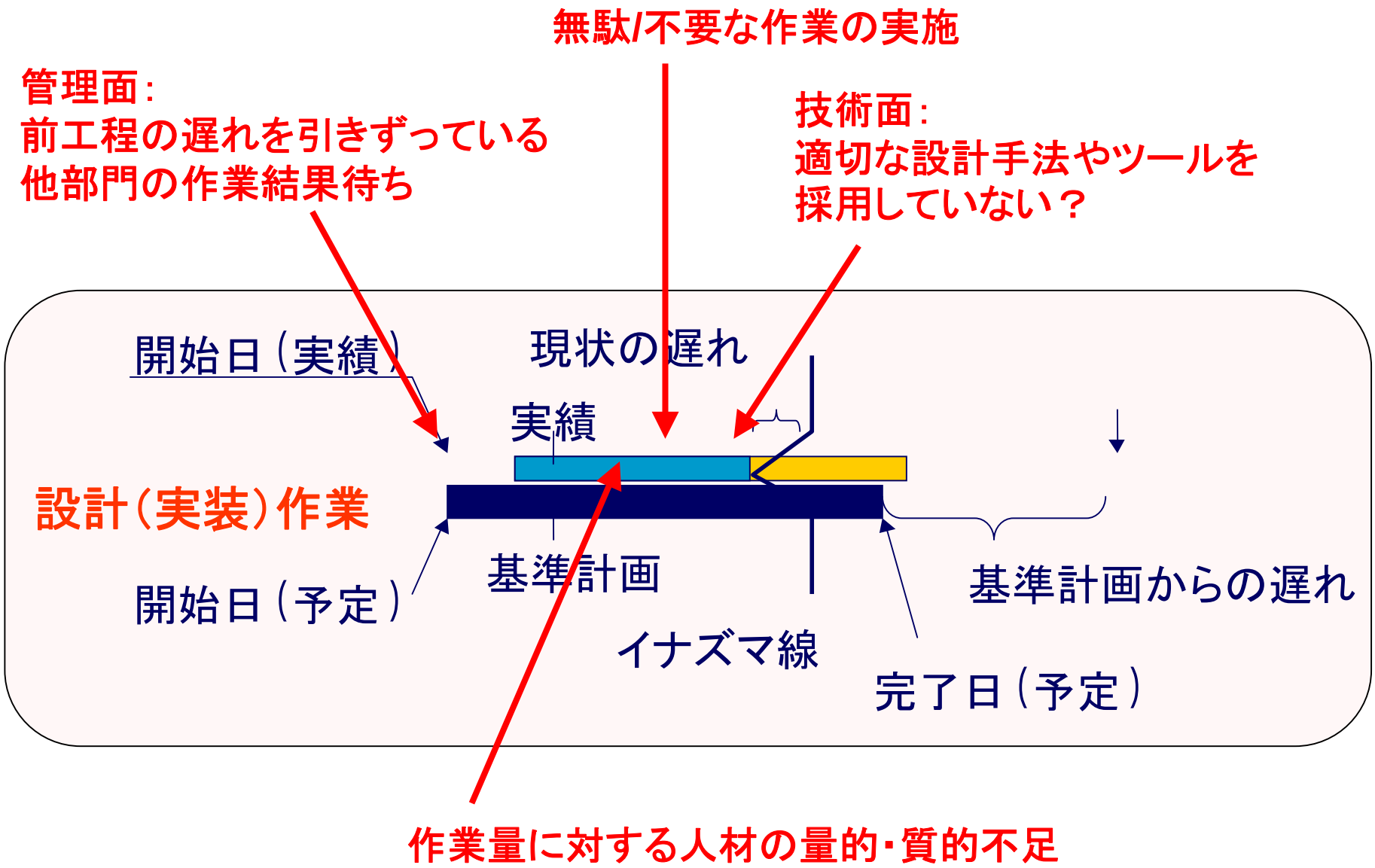
- 実際の開発の状況を可視化

原因の可視化

- 開発技術面の問題
- 開発管理/プロジェクト管理面の問題
- 人材・スキル面の問題

・問題の原因は組織・プロジェクト特性に起因する場合が少なくない
・表面的な問題ではなく、問題の真因を突き止める

進捗遅れから読み取れる遅延要因



Step-3:
プロジェクト状況の診断と適切な治療の実施

プロジェクトの不調は複合的な原因による結果

- 主たる要因の把握
- 解決可能な原因と解決不可能な原因の区別
- 解決すべき要因の優先度付け

最も効果的な手法・ツールを選択

Today's Topics

1. 組込みシステム開発の現状
2. ソフトウェアエンジニアリングの活用戦略
3. 「要求仕様の変更」への対応
4. エンジニアリング手法導入の考え方

3.「要求仕様の変更」への対応

現象

要求仕様の変更が多い ……▶ 開発の後戻りにより スケジュール遅延が発生

要求定義の段階で全ての要求を決めているか？

原因の分析

- 決めない　：　本来決めれるところがあるのに決めていない　✕ ⇨ 意思決定の問題
- 決めたつもりが　：　決めれたつもりが、実は曖昧なところが多く残っていた　✕
- 決めれない：　特別な事情により決めることができない部分がある
- 決めたが変わった　：　いったん決めたものの事情により変更が生じた

（決めれない・決めたが変わった）→ 予め仕様変更を想定した開発プロセスで対応

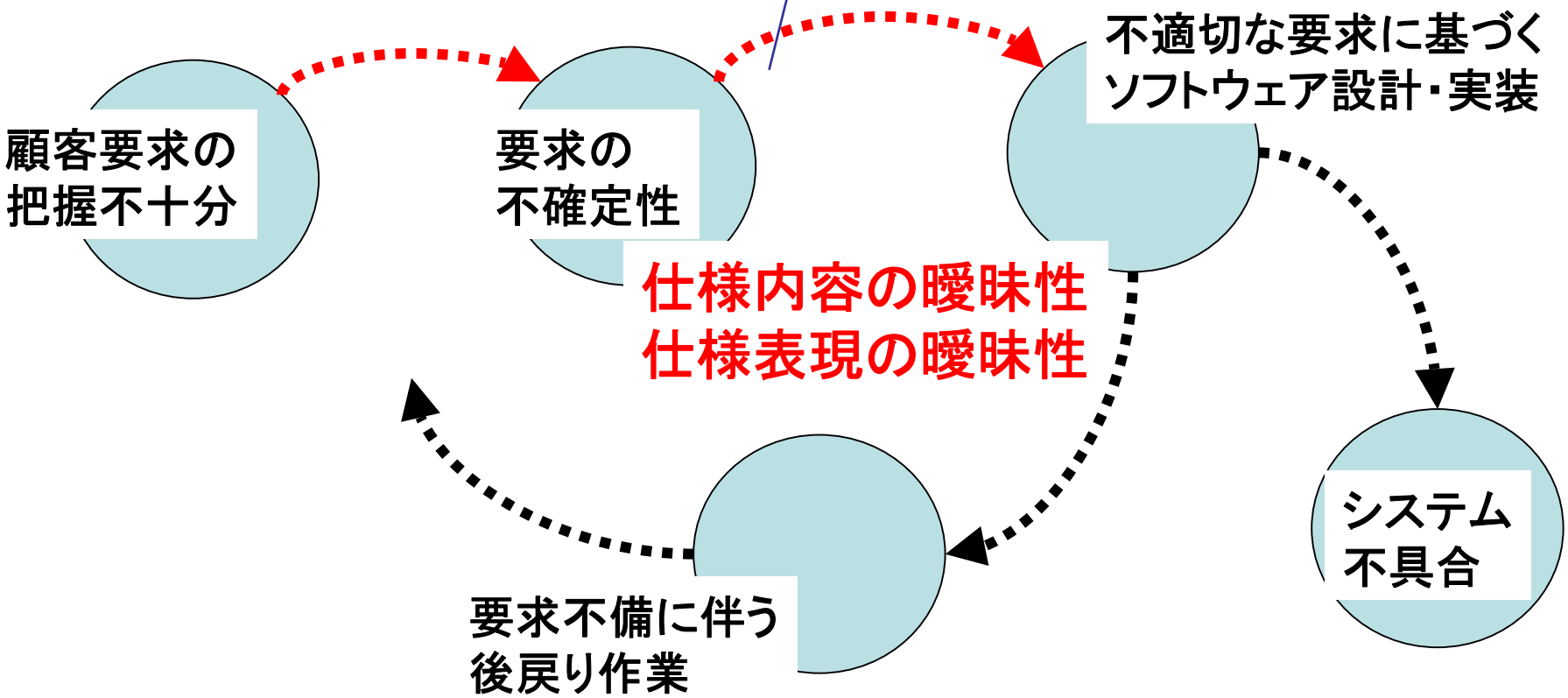
曖昧な要求仕様による開発への影響

要求分析＆定義

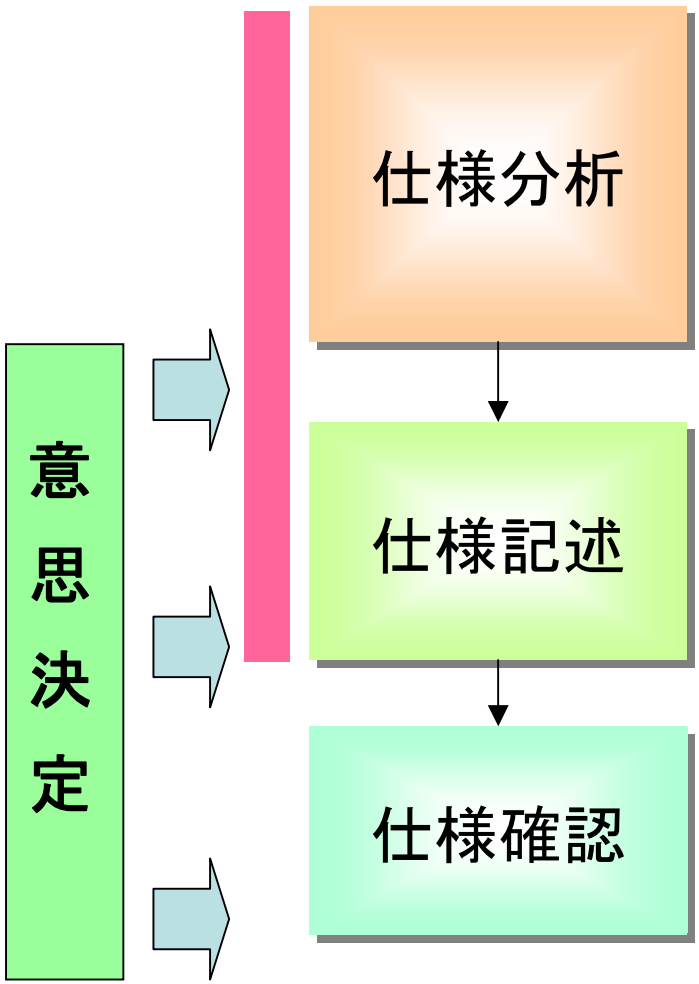
要求仕様（書）の確認＆レビュー

- 仕様決定事項/未決定事項の可視化
- 可変要因の洗い出し
- ハードウェアなど制約事項の洗い出し

- システム動作環境（コンテキスト）の網羅
- 機能間のコンフリクトの回避
- 非機能要求面の確認



要求仕様定義フェーズの問題



仕様分析の不完全性

① システムの想定利用者像特定
② システムの利用コンテキスト特定
③システムの機能/非機能要求の特定

仕様記述の不完全性

④ システム全体像の把握
⑤ 仕様の曖昧性排除

仕様分析の不完全性

仕様の曖昧性　⇒　決めたつもりが実は決まっていない
仕様の行間が埋まっていない

仕様の暗黙の了解事項
⇒「自明なこと」は技術者により異なる

- システム利用者像と利用シナリオ
- システムの動作コンテキスト
- 機能要求と非機能要求

① システム利用者像の特定

システムの典型的な利用者像を検討

=

ペルソナ

ペルソナによって
必要な機能などに差が生じる

(例) DVD録画再生機

ペルソナ1:
20代　男性
独身　　会社員
社宅住まい

・時間がないので番組を撮り貯めて
後で必要な番組をまとめて再生視聴
・メカ操作には比較的強い
⋮

ペルソナ2:
30代　女性
既婚者　専業主婦
持ち家住まい
子供あり

・昼間の時間は比較的自由に視聴可能
・視聴する番組は出演者の顔ぶれなど

で決める
・メカ操作には不慣れ

② システムの利用コンテキスト

システムの動作環境条件 …… どこまでを想定するか

沸騰モード

内部の水の温度：T

T<100℃	---	沸騰していないと判断し ヒータで過熱
100℃≦T	---	沸騰したと判断し保温モードへ

東京	：標高0m	⇒	沸点 100℃
ボルダー	：標高1629m	⇒	沸点 94℃
富士山	：標高3776m	⇒	沸点 87℃

ボルダーでは水の温度は100℃まで上がらない

⇒ ヒータ加熱が続いて水が蒸発しきってしまう？

③ 機能要求 vs 非機能要求

要求＝「組込みシステム」の設計に必要な情報

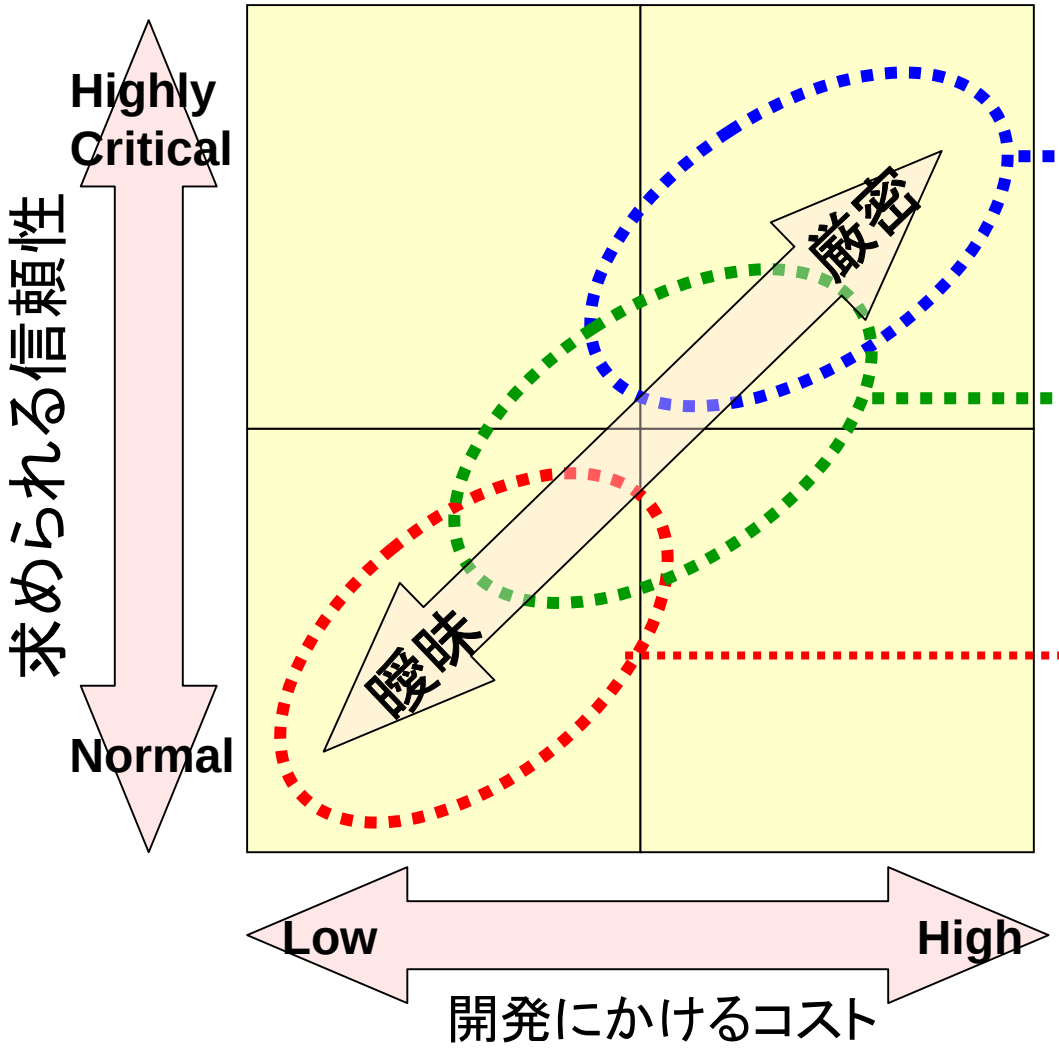
	機能要求	非機能要求
ハードウェア関連情報	どのような HWデバイス CPUを使うか	HWの性能はどの程度か どのような HW制約,精度があるか
ソフトウェア関連情報	どのような 機能を実現するか MWが必要か	SWとしてどの程度の**性能**が必要か **保守性,移植性,操作性**はどの程度を求められるか

「機能的要求」は当たり前

どこまで「非機能的要求」を事前に抽出できるかがポイント

仕様記述の不完全性

システム/PJ特性と仕様表現＆確認手法の選択



形式的仕様検証による
曖昧性の除去

形式的仕様記述法の導入
要求モデリングの導入

仕様レビューの実施
仕様書の記載内容の
標準化

仕様の表現形

要求として
　何を決めておけばよいか
　何をドキュメントとして残すべきか

Semi Formal

要求仕様書テンプレート
- 構成の枠組み
- 表による記述内容の提示
- サンプル図

要求として
・検討すべき/記述すべき
　標準的な項目を規定
・ 記述のフォームを規定

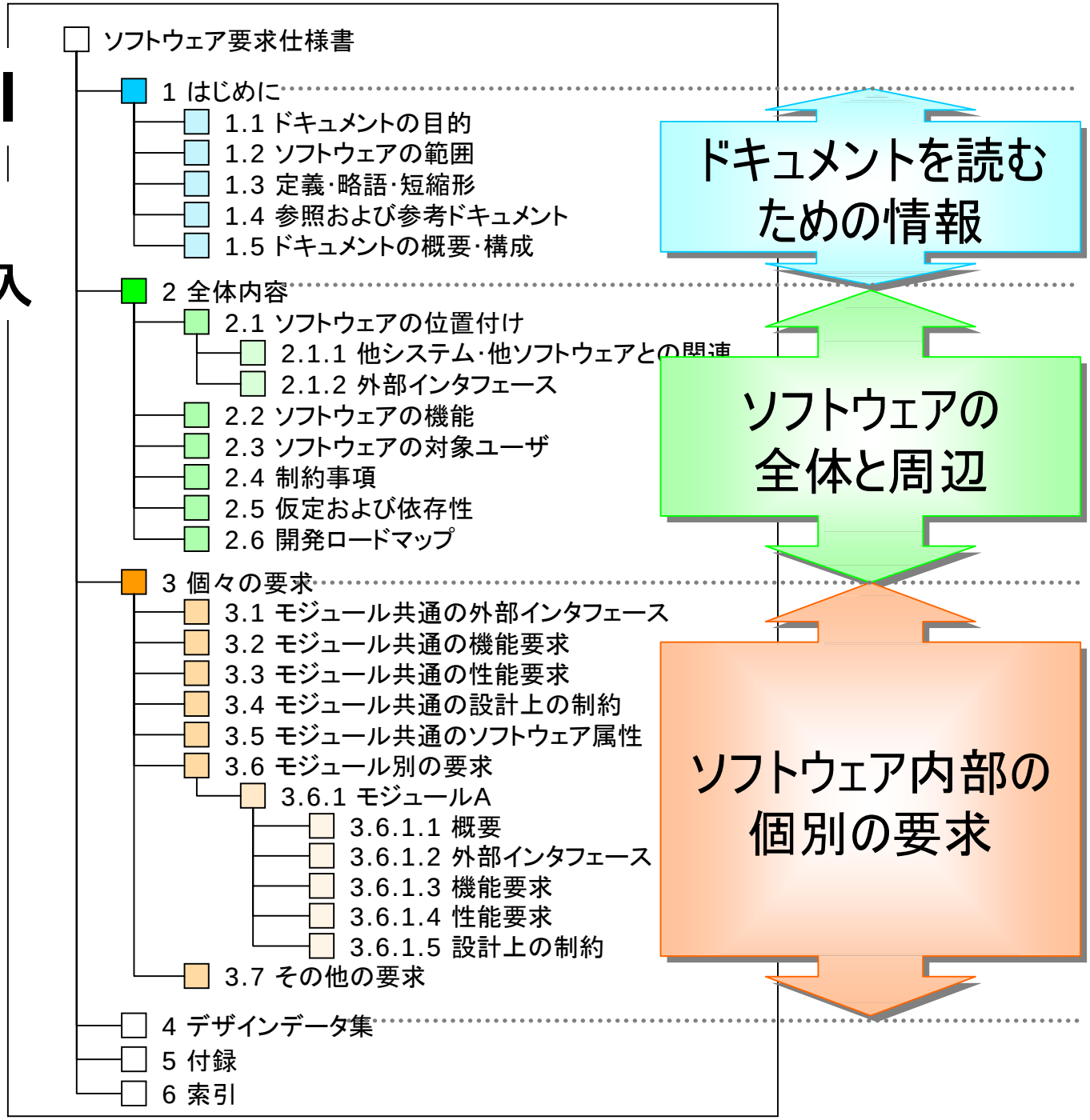
Formal Specification

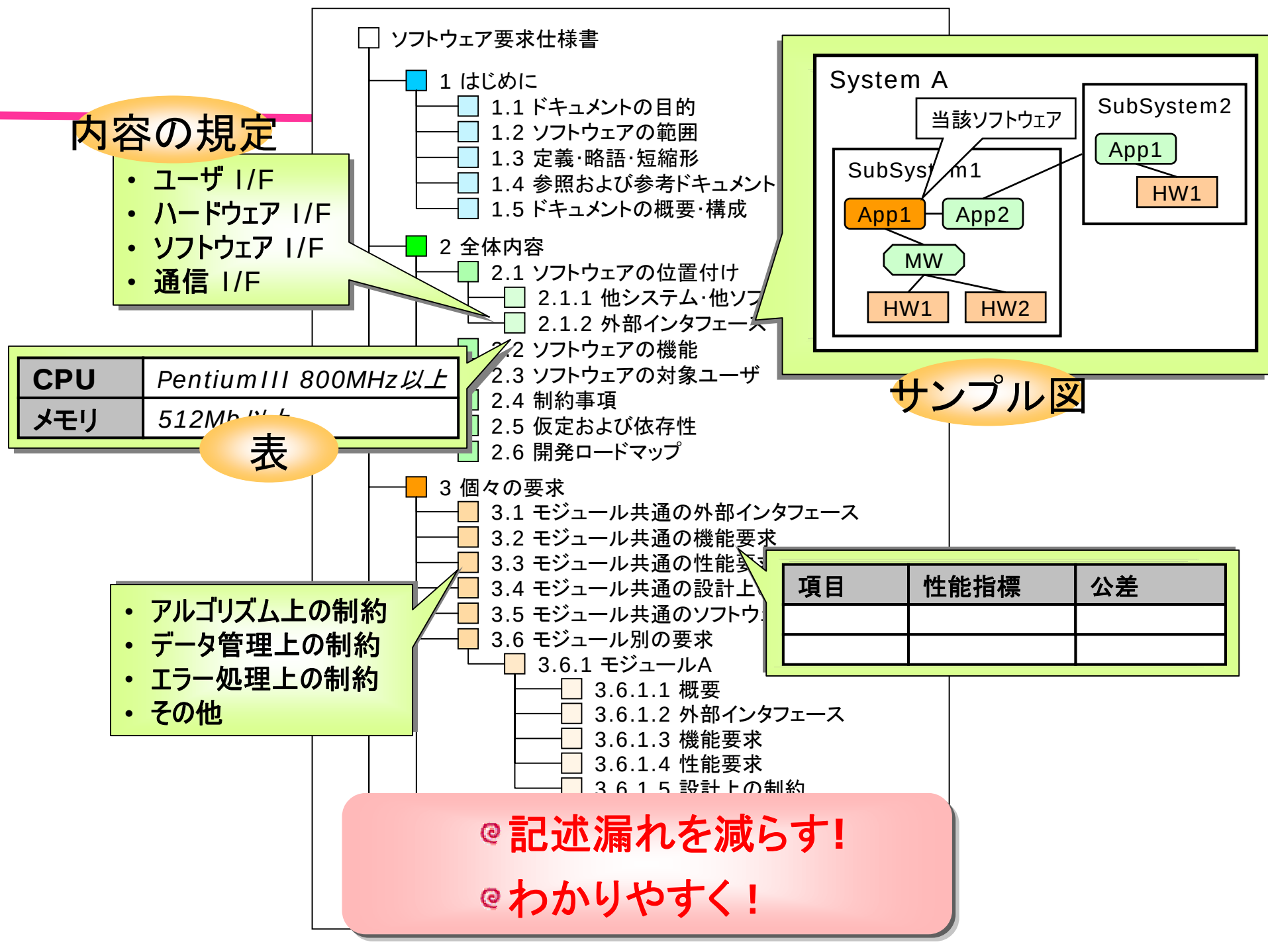
UML他のモデリング手法
・厳密な書式による仕様の
（論理的）表現

形式検証ツールなどによる
論理的検証が可能

Semi Formal

要求仕様書
テンプレートの導入




内容の規定
• ユーザ I/F
• ハードウェア I/F
• ソフトウェア I/F
• 通信 I/F
CPU
Pentium III 800MHz以上
メモリ
512Mb以上
表
• アルゴリズム上の制約
• データ管理上の制約
• エラー処理上の制約
• その他
ソフトウェア要求仕様書
1 はじめに
1.1 ドキュメントの目的
1.2 ソフトウェアの範囲
1.3 定義･略語･短縮形
1.4 参照および参考ドキュメント
1.5 ドキュメントの概要･構成
2 全体内容
2.1 ソフトウェアの位置付け
2.1.1 他システム･他ソフ
2.1.2 外部インタフェース
2.2 ソフトウェアの機能
2.3 ソフトウェアの対象ユーザ
2.4 制約事項
2.5 仮定および依存性
2.6 開発ロードマップ
3 個々の要求
3.1 モジュール共通の外部インタフェース
3.2 モジュール共通の機能要求
3.3 モジュール共通の性能要
3.4 モジュール共通の設計上
3.5 モジュール共通のソフトウ
3.6 モジュール別の要求
3.6.1 モジュールA
3.6.1.1 概要
3.6.1.2 外部インタフェース
3.6.1.3 機能要求
3.6.1.4 性能要求
3.6.1.5 設計上の制約
System A
当該ソフトウェア
SubSystem1
App1
App2
MW
HW1
HW2
SubSystem2
App1
HW1
サンプル図
項目
性能指標
公差
記述漏れを減らす!
わかりやすく!

システム全体像の把握

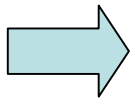
システム全体を見通して如何に矛盾のない“形“を作り上げるか

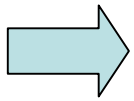
システムの性能＆信頼性側面から

機能＆サービスの競合/並行動作
割り込み条件/タイミング
機能毎のデータ連携
ハード/ソフトの連携ポイント抽出
エラー処理の方針

再利用性・移植性側面から

機能＆サービスレベルでの
再利用/機能変更の可能性
バリエーション展開の考え方
周辺ハードの変更可能性やその影響





- 機能＆サービス関連図（機能ブロック図）
- システム共通処理ユニット＆共通データ一覧
- 並行動作＆動作タイミング整理表
- 機能優先度表
- 関連データ＆センサーデーター覧
- 割り込みパターン一覧
- 個別機能＆サービス動作条件/制約一覧
- 再利用可能性検討表
- エラーハンドリングポリシー

機能ユニットの整理と優先順位付け

Sub1:
音声通話

Sub2:
メール

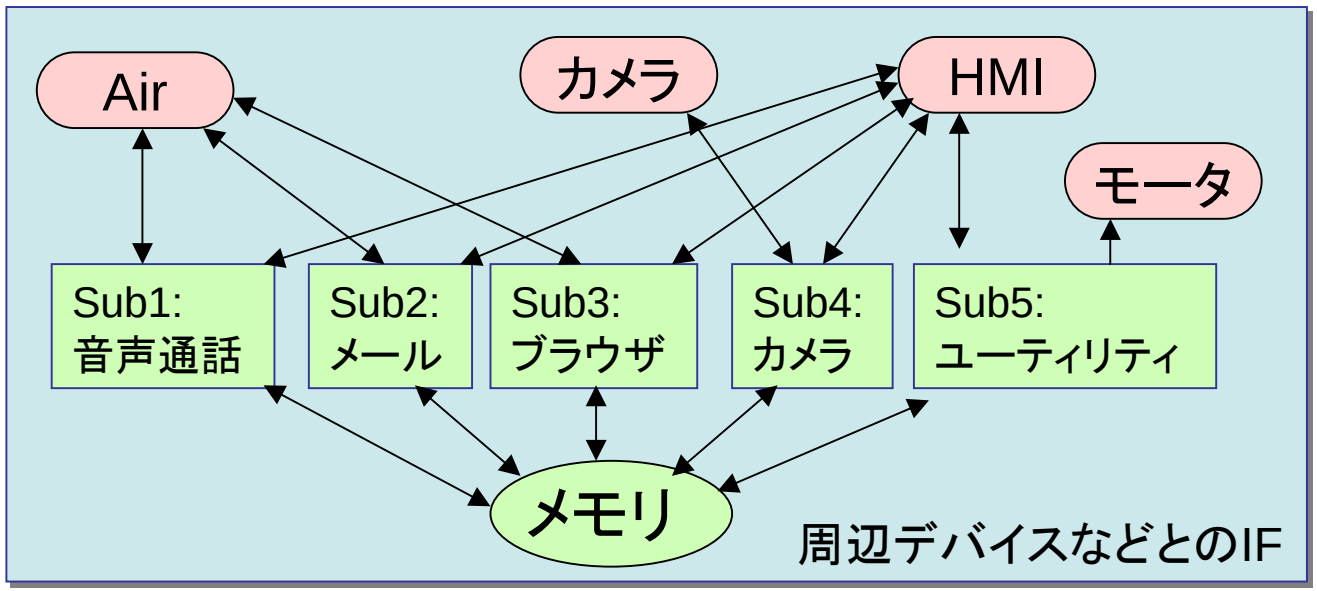
Sub3:
ブラウザ

Sub4:
カメラ

Sub5:
ユーティリティ

サブシステム分割図

No.	サブシステム名	優先順位	中機能分類	概要	変更可能性
Sub1	音声通話	1	待ちうけ		なし
			通話		なし
Sub2	メール	2			
Sub3	ブラウザ	4			
Sub4	カメラ	5			あり
Sub5	ユーティリティ	3			



並行動作整理＆エラーハンドリング

			SUb1		Sub2			Sub3	Sub4	Sub5
			音声通話		メール			ブラウザ	カメラ	ユーティリティ
			待ちうけ	通話	送信	転送	受信			
Sub1	音声通話	待ちうけ								
		通話	○							
Sub2	メール	送信	○							
		転送	○							
		受信	○	○	○					
Sub3	ブラウザ		○				○			
Sub4	カメラ		○				○			
Sub5	ユーティリティ		○				○			

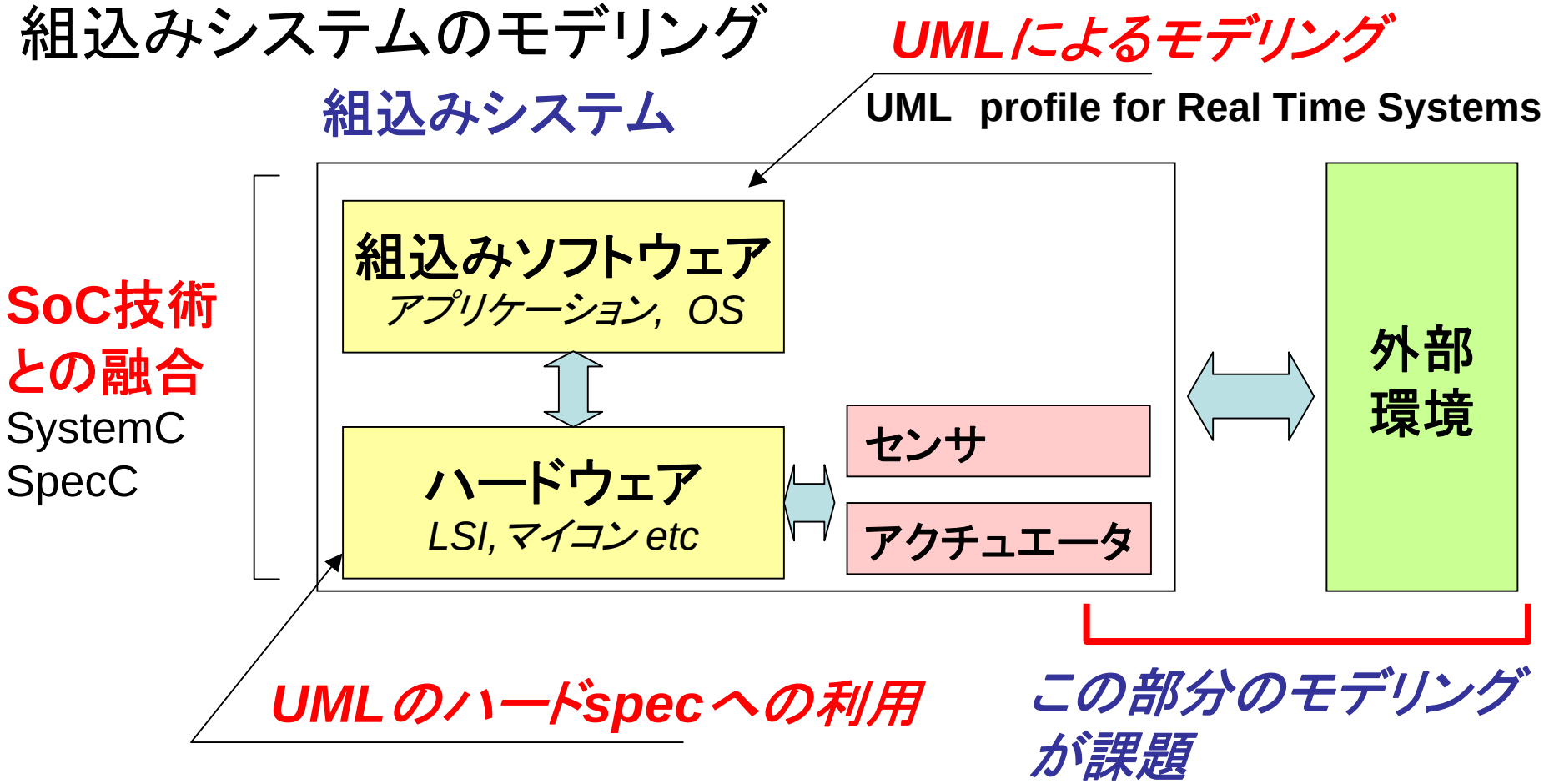
各機能サービス間の
並行動作や割り込み関係
などをマトリクス状に整理

機能サービスごとに
異常動作時のエラー
ハンドリングのレベルを
検討する

レベル		エラー処理	処理内容
1	軽度	通常メッセージ	通常のメッセージを表示し動作処理を中断
2		警告メッセージ	警告メッセージを表示し動作処理を中断
3		サブシステム再起動	当該のサブシステムのサービスを停止再起動
4		システム再起動	システム全体をいったん停止し再機能
5	重度	システム停止	重大エラー発生を表示しシステム全体を終了

Formal Specification

組込みシステムのモデリング



Point-1: ソフトウェアのモデリング
Point-2: ハード＆ソフト＆外部環境の接点を明確化
Point-3: 信頼性/安全性を実現する信頼性保証手法の導入

モデリング手法の課題

① モデルの記述能力
-リアルタイム性/リアクティブ性
-ハードウェア依存要素
-外部環境要因の表現

⇒ UML2
UML Profile
コデザインでのUML利用

② 利用者側の課題
-設計スキル（抽象化する能力）

⇒ 要求工学

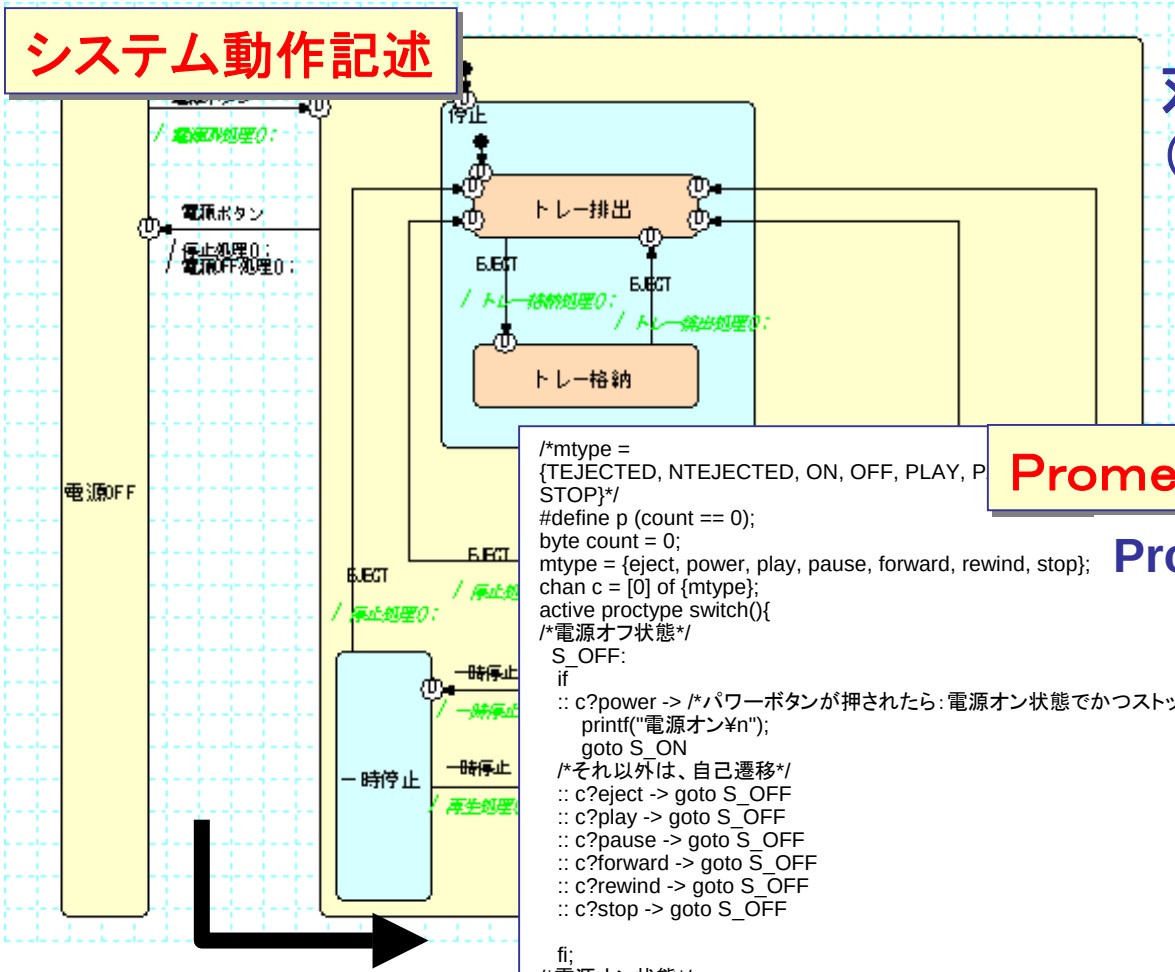
技術者教育
（UMLロボコンなど）

特に
ハードウェア/ソフトウェア両面の記述
⇒ System-Cなどコデザイン関連
非機能用件の表現
⇒ ユーザビリティ、セキュリティなども

結構使いこなすの難しい

Model Checking の適用事例

システム動作記述



対象：CDプレーヤ
（トレーユニット）

Promela記述

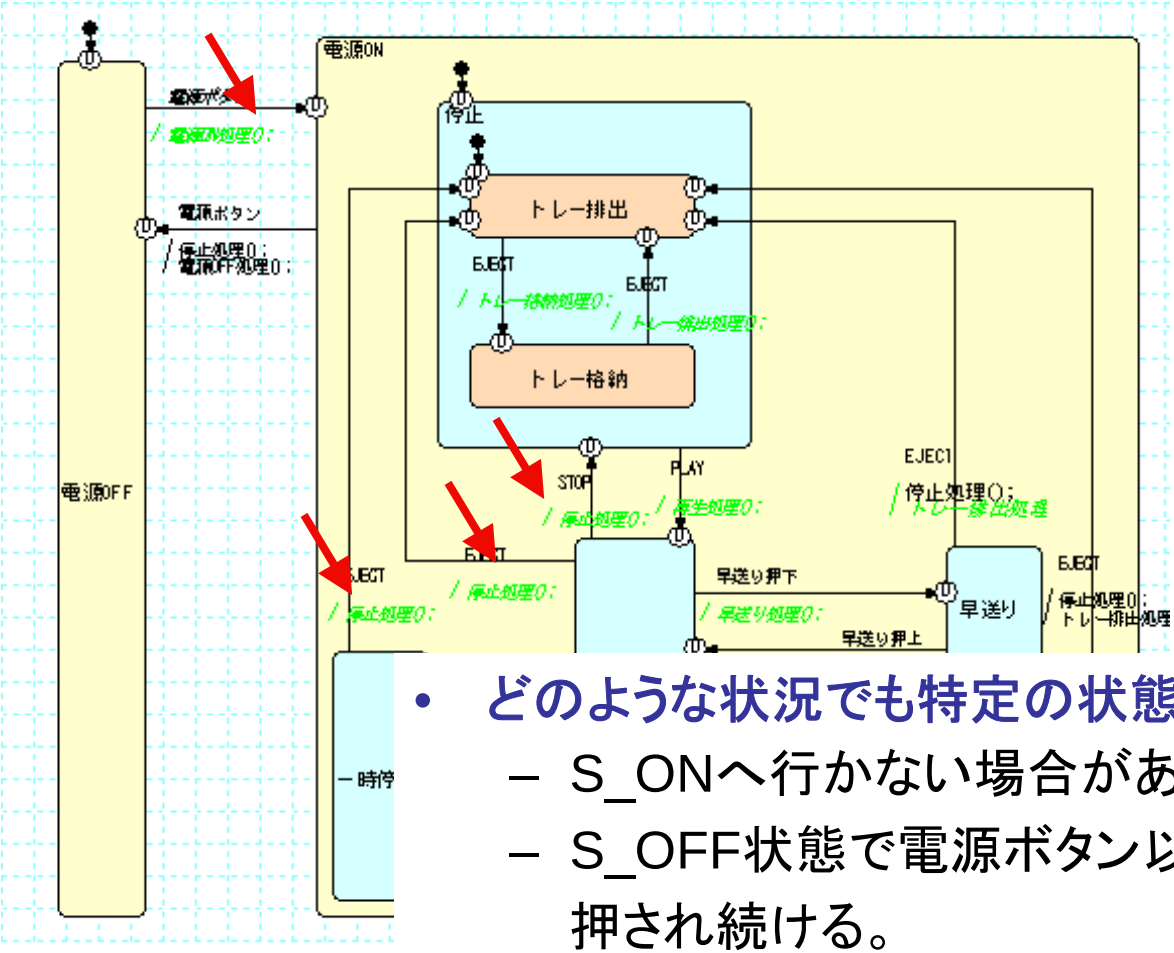
```
/*mtype =
{TEJECTED, NTEJECTED, ON, OFF, PLAY, P
STOP}*/
#define p (count == 0);
byte count = 0;
mtype = {eject, power, play, pause, forward, rewind, stop};
chan c = [0] of {mtype};
active proctype switch(){
/*電源オフ状態*/
  S_OFF:
   if
   :: c?power -> /*パワーボタンが押されたら：電源オン状態でかつストップ状態*/
       printf("電源オン¥n");
       goto S_ON
   /*それ以外は、自己遷移*/
   :: c?eject -> goto S_OFF
   :: c?play -> goto S_OFF
   :: c?pause -> goto S_OFF
   :: c?forward -> goto S_OFF
   :: c?rewind -> goto S_OFF
   :: c?stop -> goto S_OFF

   fi;
/*電源オン状態*/
  S_ON:
   if
   :: c?power -> /*パワーボタンが押されたら：電源オフ状態*/
       count ++;
       printf("電源オフ¥n");
       count --;
       goto S_OFF
            ▪
            ▪
            ▪
```

Promela記述：約300行

検証システム
(SPIN)
により動作検証

適用結果



- どのような状況でも特定の状態に到達するか？
  - S_ONへ行かない場合がある。
  - S_OFF状態で電源ボタン以外のボタンが押され続ける。
- 特定のボタンが押されると期待する状態に到達するか？
  - 「電源ON・停止・トレー排出状態で再生ボタンが押されると、いつかは再生状態に行く」が成立しない。

Model Checking 適用に関する考慮点

SECでのトライアルの場合

被験者：3名（開発経験は5年前後）

教育フェーズ

JAIST教育講座受講 ……
(3日間)

- プログラミング経験
- 状態遷移の考え方を理解していることが必須

トライアルフェーズ

対象システムの状態図整備
Promela記述作成
SPINによる検証

Promelaは対象システムの設計モデルのすべてを記述するには記述能力が低い。

⇒ モデル検査で扱える部分は、設計の一部。
ハードウェアとのインタラクションの部分などは弱い

検証のためのモデル（検証モデル）を作成しなければならない

⇒検証したい部分の特定。
検証可能なように厳密化。
検証したい性質が取り扱えるよう変形や埋め込み。

Today's Topics

1. 組込みシステム開発の現状
2. ソフトウェアエンジニアリングの活用戦略
3. 「要求仕様の変更」への対応
4. エンジニアリング手法導入の考え方

4. エンジニアリング手法導入の考え方

ソフトウェアエンジニアリング手法

様々なものが出回っている

↓

プロダクトライン,MDD,
アスペクト指向,CMMI. . . .

必ずしも効果が立証されていないものも少なくない

手法の利用者（開発技術者）は忙しく、保守的

いきなり高価な手法（High Cost / High Return) を目指すのは？
簡単な手法でも大きな効果が得られる場合もある

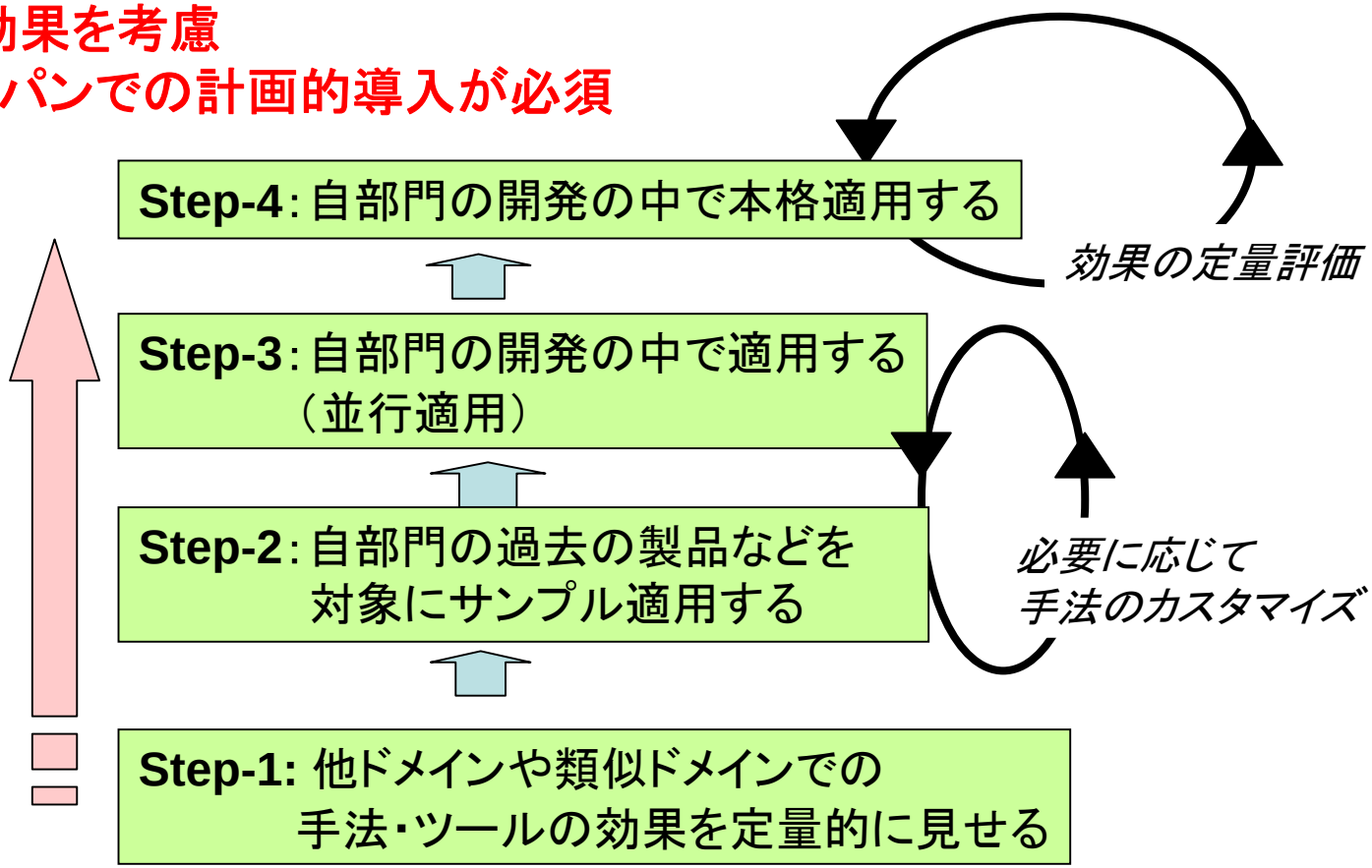
⇨ どのような“治療“を目指すかの戦略が必要

外科手術
投薬治療--- 西洋医学 vs 東洋医学

エンジニアリング手法導入のステップ

Point-1

- 投資対効果を考慮
- ロングスパンでの計画的導入が必須



Point-2

- 技術導入計画を実行する
  人材（技術者,マネージャ）の育成が前提