レガシーモダナイゼーション実践に向けた技術開発

2014年 9月 2日

株式会社　NTTデータ
技術開発本部　ソフトウェア工学推進センタ　　坂田祐司

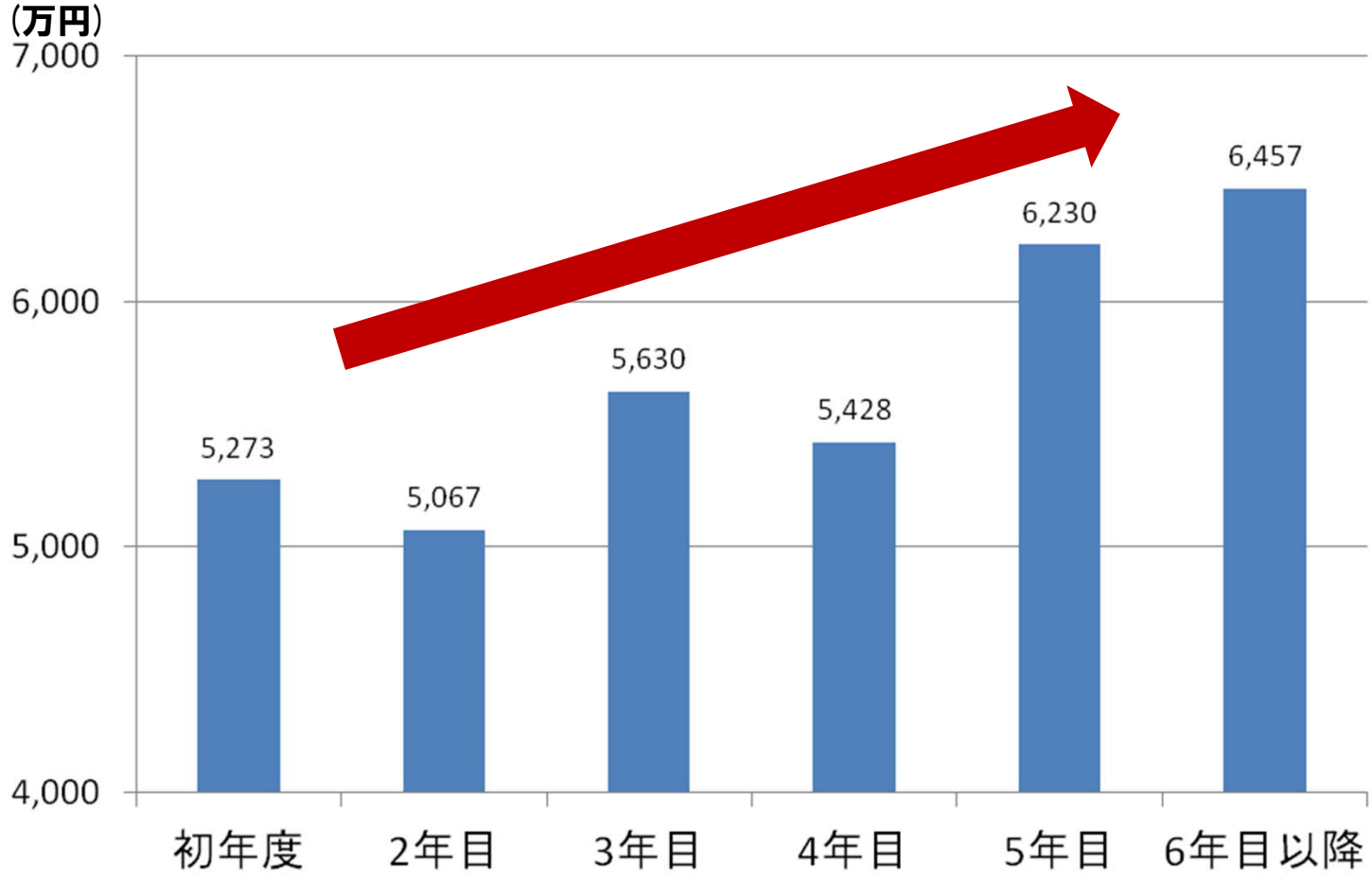
NTT DATA


レガシーシステムに内在する問題

導入後の運用・保守コスト

保守開発による複雑化や、ハードウェアの老朽化で運用・保守費用は年々増加する

図．自社開発の稼働後の年間保守費用の平均（単位：万円）



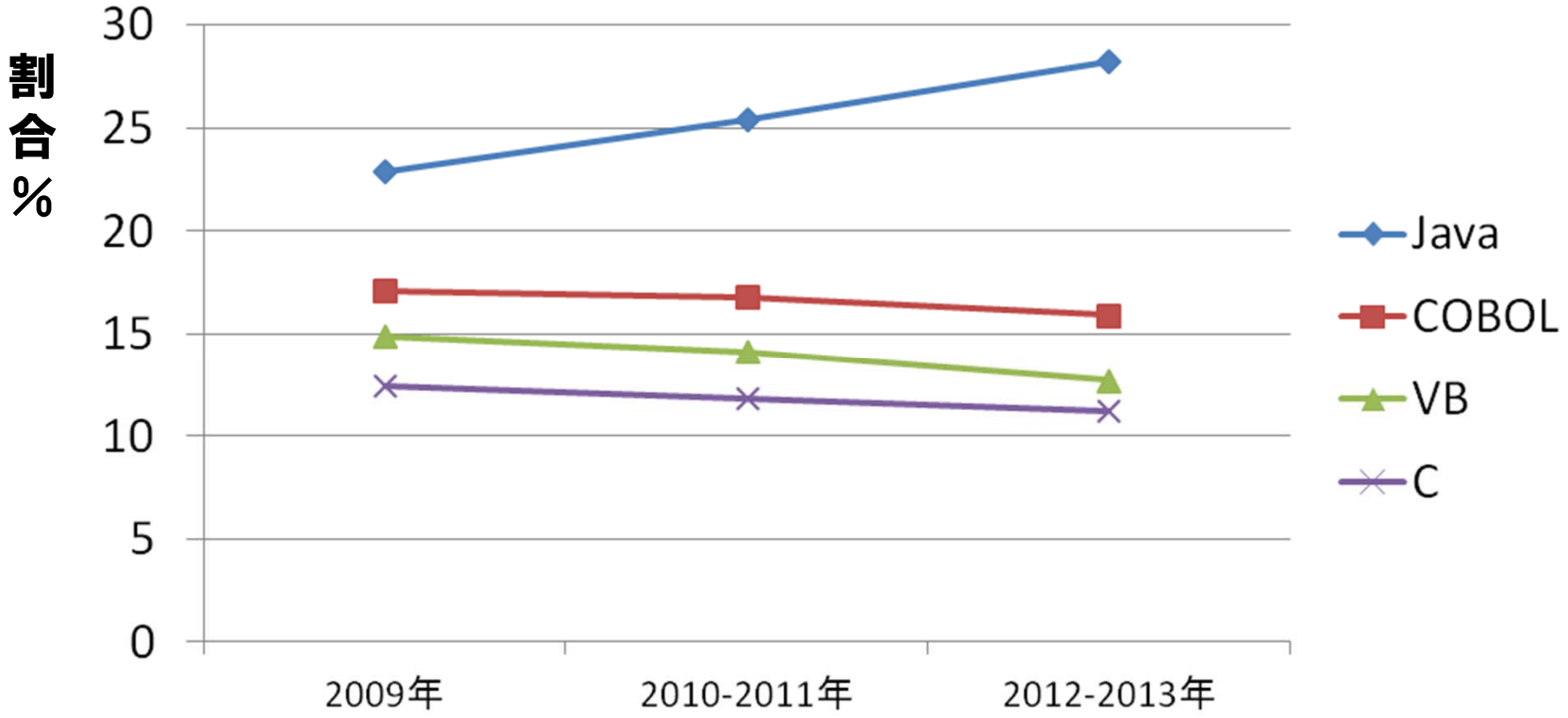
※ 出典：JUAS「「ソフトウェアメトリックス調査2012」より作成

開発に用いられる開発言語の変化

NTT DATA

Java言語の開発の増加に伴い、Java技術者が増加傾向
一方で、COBOL技術者はゆるやかな減少傾向にある

図．開発に用いられる言語の割合の変化

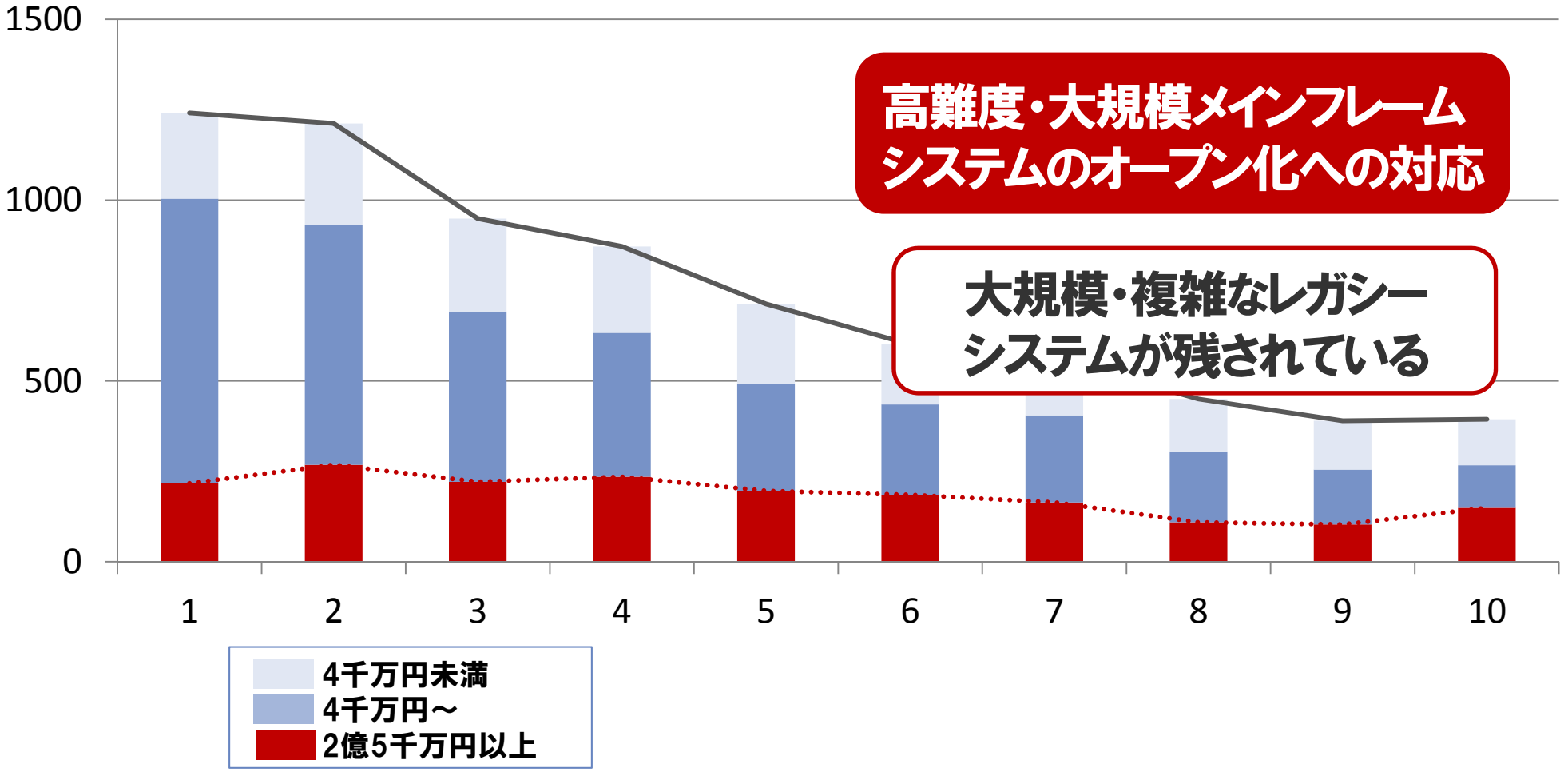


※ 出典：IPA「ソフトウェア開発白書2009」「ソフトウェア開発白書2010-11」「ソフトウェア開発白書2012-13」より作成

レガシーシステムに求められる対応

難易度の高いレガシーシステムへの対応が必要

図．メインフレームの規模別出荷台数の推移



【出典】JEITA(電子情報技術産業協会)の集計よりグラフを作成

レガシーシステムに内在する問題

- 運用・保守コストの増大
- レガシー技術者の減少
- 残された大規模なレガシーシステム

これらの問題をふまえた、
レガシーシステムの再生技術が求められる

レガシーシステム再生への開発手法

レガシーシステムから新システムへの開発目的

NTT DATA

開発目的や現行資産の状況により マイグレーションとモダナイゼーションに大別される

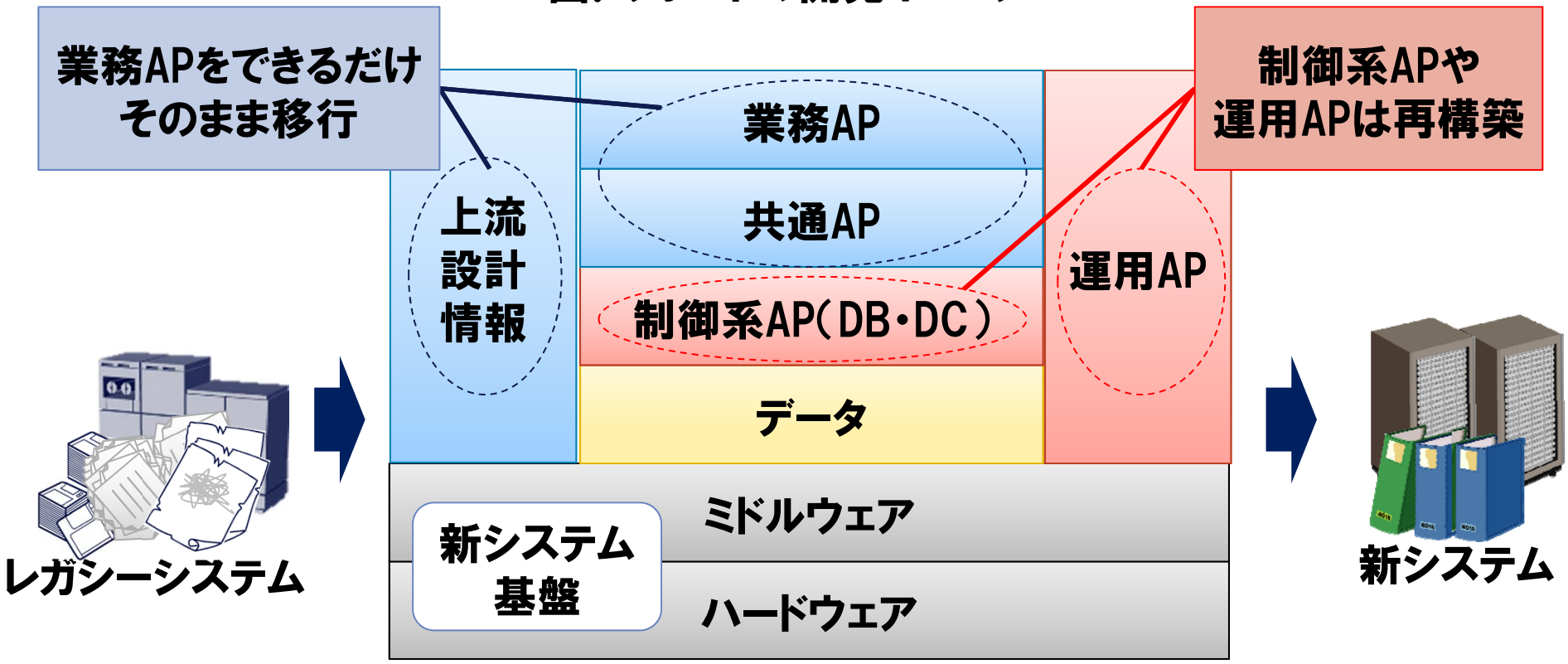
	マイグレーション	モダナイゼーション
目的	新しいシステム基盤への移行	システムを近代化しビジネス課題への対応や競争力強化
対象	業務仕様はそのまま 古いシステム基盤 → 移行 → 新しいシステム基盤	旧システム全て or 一部 → 見直し → 新システム

マイグレーションを実現する「リホスト」

NTT DATA

アプリケーション資産を流用して新しいシステム基盤に移行するアプローチ

図. リホストの開発イメージ

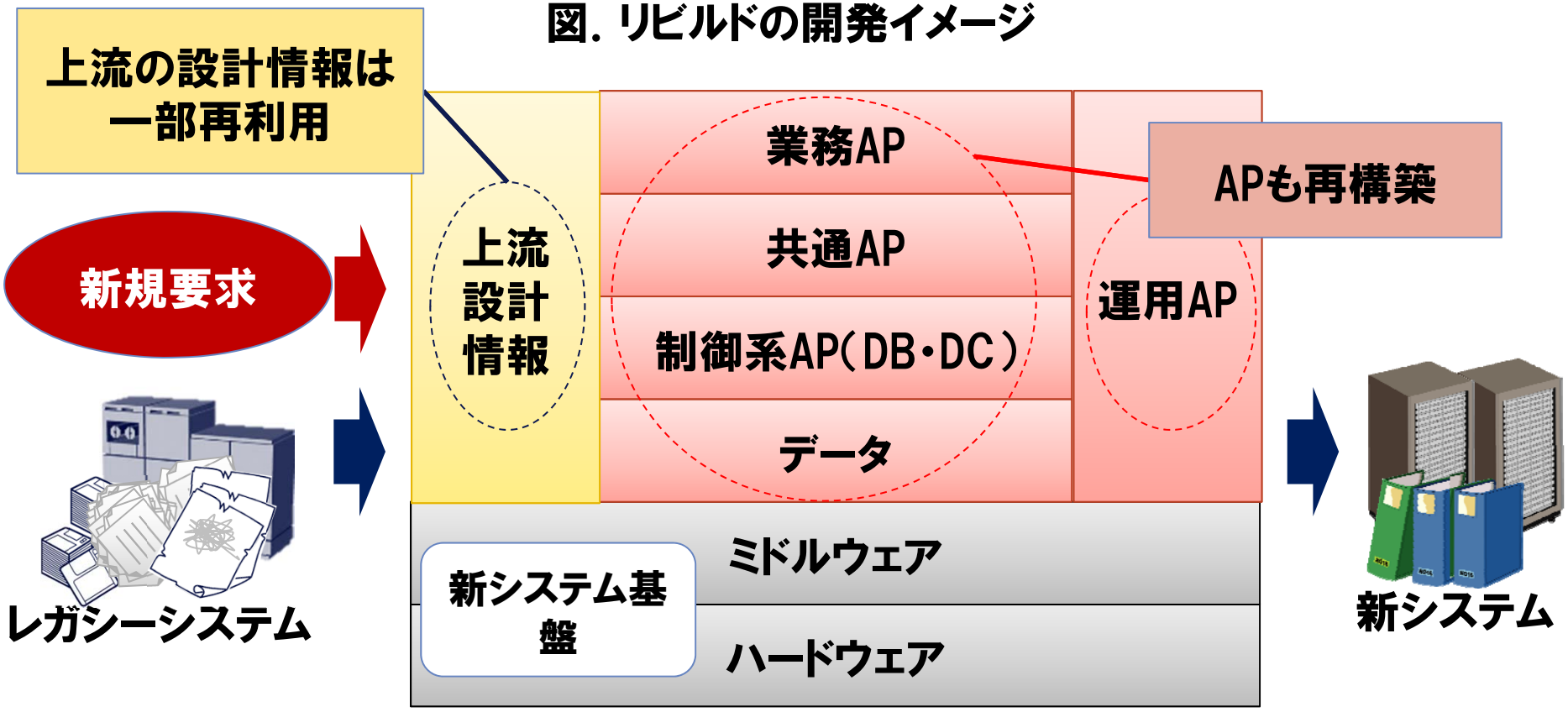


メインフレームのオープン化に使われることが多い

モダナイゼーションを実現する「リビルド」

新規要求の実現や現状の非効率な面を改善し システムの価値を高めるアプローチ

図. リビルドの開発イメージ



オープン化だけでなく、競争力強化にも使われる

「リホスト」と「リビルド」の比較

NTT DATA

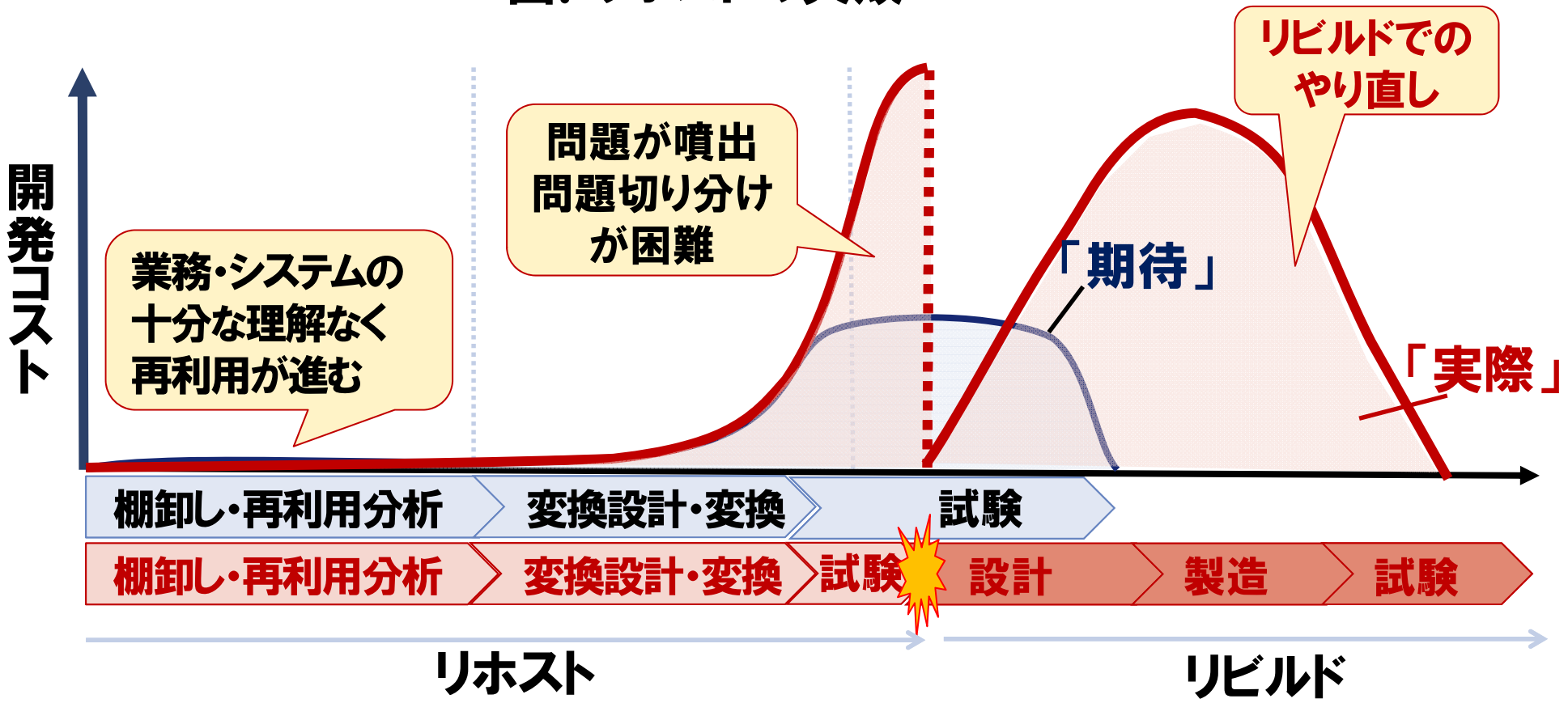
開発手法のメリット、デメリットが一般的に認識されている

比較項目		リホスト	リビルド
移行内容	コード	変更小、自動変換	変更大
	業務ロジック	変らない	見直す
移行コスト		コスト小～中	コスト大
移行後の効果	保守・運用コスト	削減できる	削減できる
	業務効率	変らない	向上が期待できる
	新規業務への迅速な対応	あまり変らない	向上が期待できる

リホストが失敗した大規模システム更改

リホストが失敗したケースでは、試験工程で問題が一気に噴出し、大幅な手戻り開発コストが膨れ上がってしまう場合がある

図. リホストの失敗



レガシーシステム再生への開発手法

- マイグレーションを実現する「リホスト」
- モダナイゼーションを実現する「リビルド」
- 大規模レガシーシステムへの適用の壁

大規模・複雑化したレガシーシステムには、従来の判断基準、手法では立ち行かない

大規模レガシーシステムへのアプローチ

リビルドへの期待

リホストや保守の問題から、リビルドが再び評価されはじめている

保守の問題

- 運用・保守コストが高い
- 技術者の減少
- 属人性が高い
- 俊敏な対応ができない
- 品質が低下している

Before

システムの特徴

- 機能追加によりシステムが複雑化
- 影響分析やテストに稼働がかかる

リホストの問題

- 開発手順が複雑
- 見積りが難しい
- 保守性や俊敏性が改善しない

リビルド

After

リビルドへのお客様の期待と現実

リビルドにも解決すべき問題がある

After

期待
と
現実

リビルドへの期待

- 見積り精度の向上
- システム上の問題解決
- 保守性や俊敏性の改善
- 最新技術や開発ツール導入による生産性向上
- 潤沢な開発者の確保

リビルド開発の現実

- 業務仕様の把握が困難
- 開発に時間がかかる
- 保守性低下の繰り返し

業務仕様の把握が不十分

保守運用時は部分的な仕様把握で、全体把握がされていない

図. 保守運用時の業務仕様理解の範囲

実際の30M規模のシステムでは、年1.7%の追加・変更しか発生していなかった。

設計書の更新漏れなども発生

（変更点の記述のみで、最新状態の把握が困難なケースも）

システムのブラックボックス化、有識者に依存した現状把握

業務仕様の把握が不十分

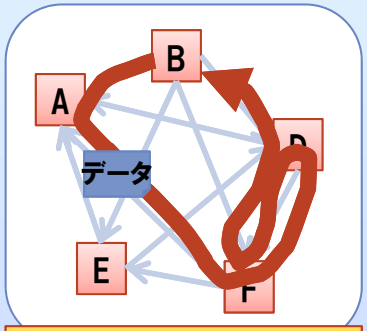
リビルド時には、数倍～十数倍の新規メンバーが参入

有識者ボトルネックを防ぐ取組みが求められる

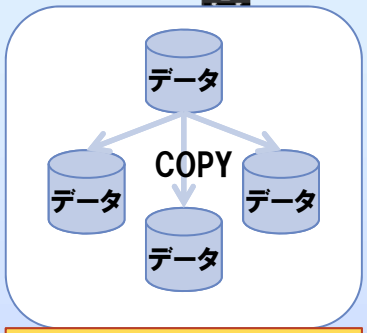
開発に時間がかかる

保守性の改善やシステム価値向上には、設計に時間が必要

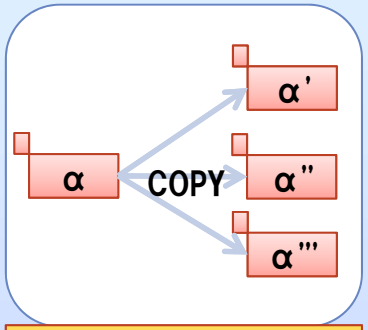
複雑化・肥大化の原因例



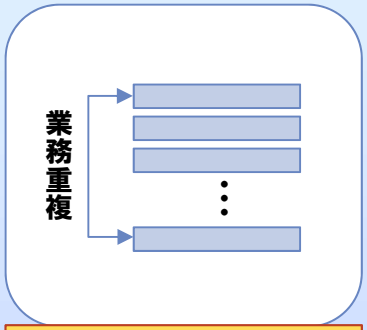
複雑なI/Fや データフロー



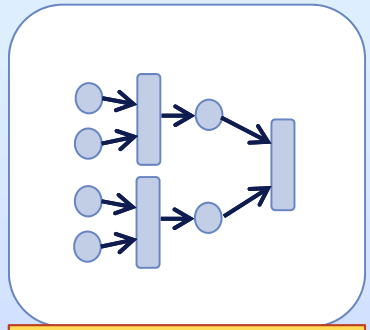
重複・類似した データや帳票



機能重複や コードクローン



業務重複や 操作性不統一



肥大化した バッチ処理

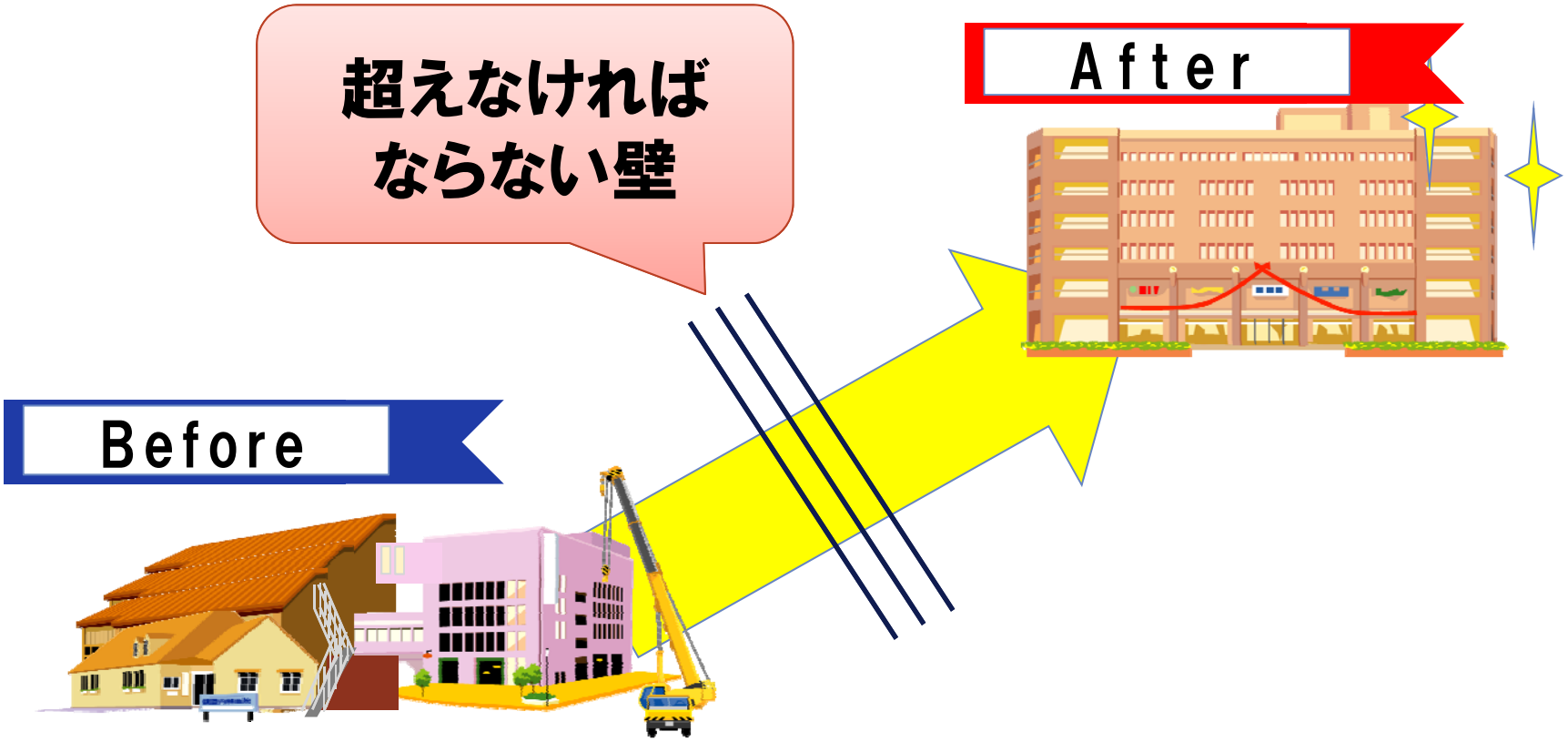
←―― 保守性・俊敏性 ――→　　<効率>　　<時間>

リビルドによるメリットの享受を期待したが
設計工期の時間切れで、改善ができないケースも

開発に時間がかかる

NTT DATA

新システムによる大幅な改善により
データ移行の設計・開発・実施の難易度が上がる

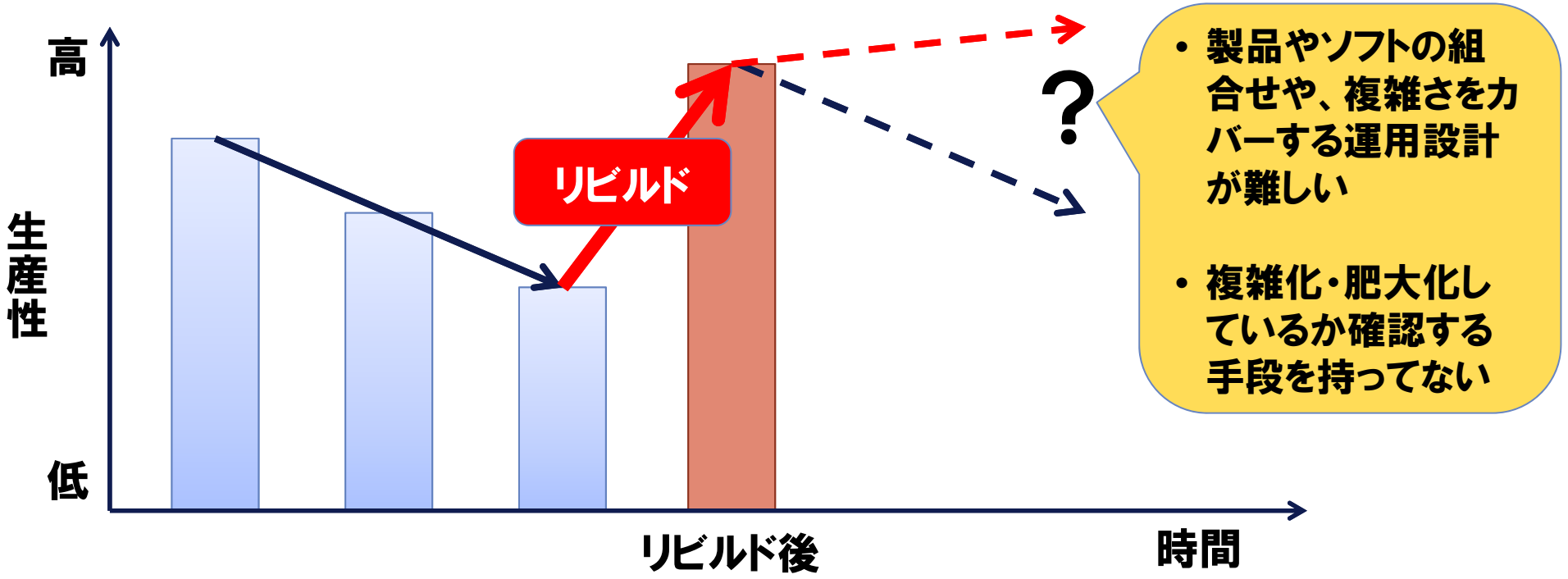


改善の効果を事前に検証し、注力すべき点を明確化

保守性低下の繰り返し

NTT DATA

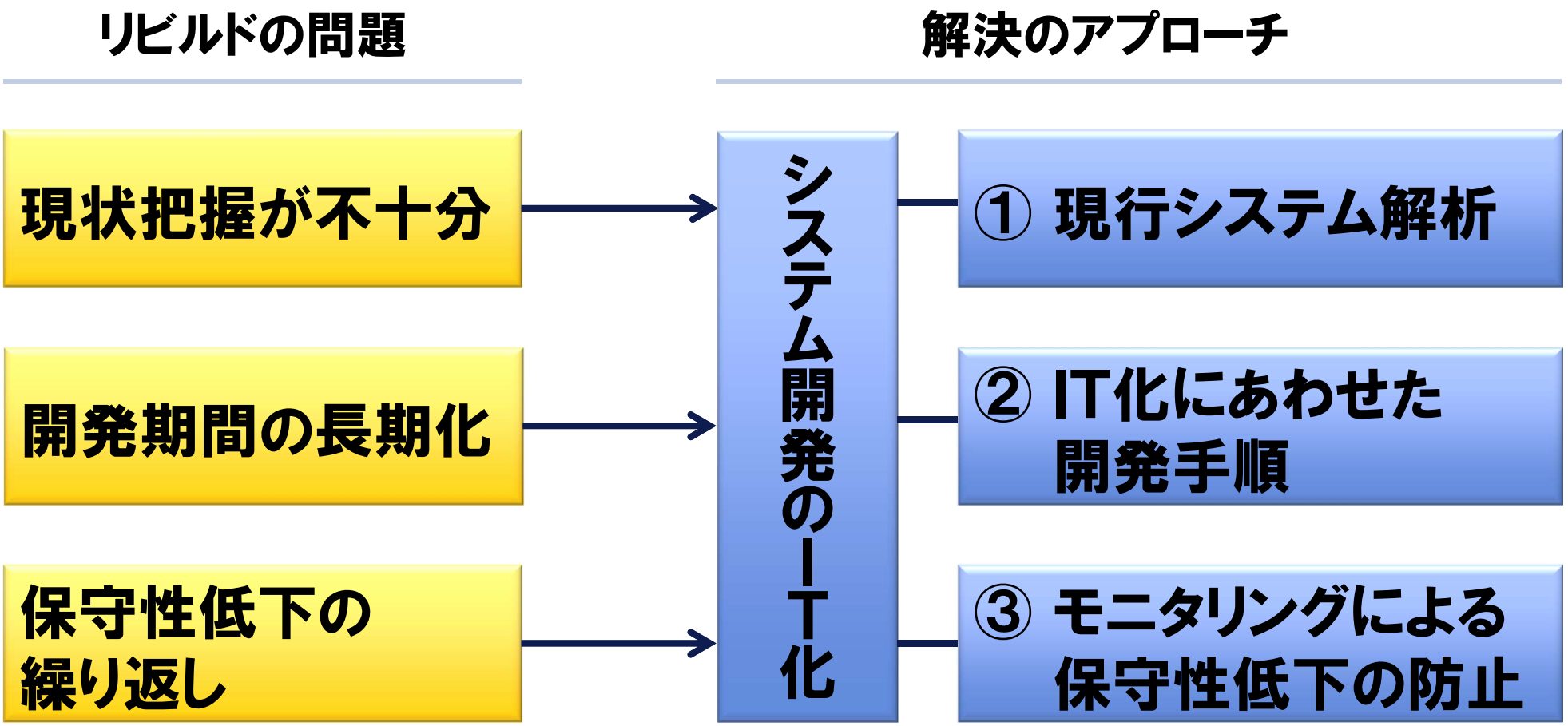
リビルドにより保守性が向上するも
対策を取っていないと、再び複雑化・肥大化してしまう



新システムの中に、複雑さ・肥大化を防ぐ仕組みが必要

リビルドの成功に向けて

リビルドによるシステム最適化のためには
システム開発をIT化して生産性を高める必要がある



システム開発のIT化

NTT DATA

開発・管理・運用の各領域で自動化が取り組まれている。

システム開発のIT化（自動化） 8領域

現行解析の自動化	設計の自動化	コーディングの自動化	テストの自動化
レガシーコードからの正確な仕様解析	設計段階での整合性自動検証	複雑・多様なロジックの完全自動生成	試験項目の自動生成と自動実行

プロジェクト管理の自動化	ライブラリ管理の自動化	運用の自動化	システム基盤構築の自動化
管理情報の集計・可視化	ビルドやテスト環境へのリリースの自動化	RunBookAutomation	システム基盤の自動インストール・設定

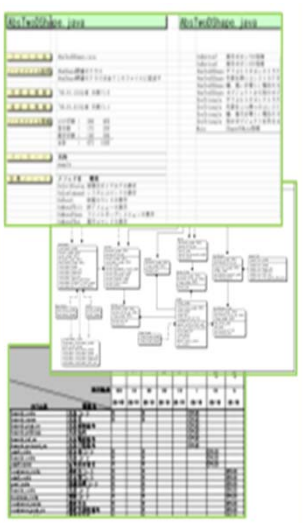
① 現行システム解析

現行システム解析にリバースエンジニアリングを活用し大規模なシステムを、早く、正確に分析できる

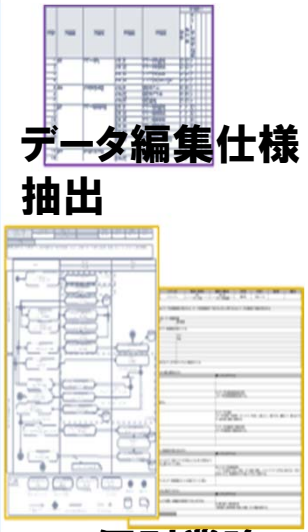
確実に信頼できる資産

- ソースコード
- 実働システムの
  画面、帳票、
  動作ログ、
  電文キャプチャ
  実データ
- システム設定

リバースエンジニアリング



業務処理ロジック抽出



データ編集仕様抽出

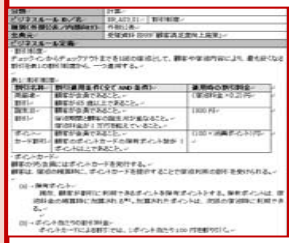
個別業務ルール抽出

「処理」

「機能、仕様」

人手による把握手順

処理設計書



システムの正しい把握

- 改善要求から要件を定義できる
- テストの設計ができる

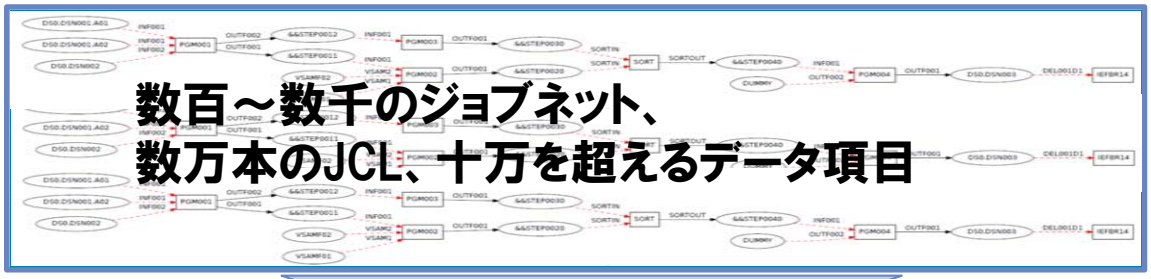
「業務」

「要件」

① 現行システム解析

大規模システムのバッチ処理、DB、帳票等の最適化にもITの力が必要

大規模システムのバッチ処理の例



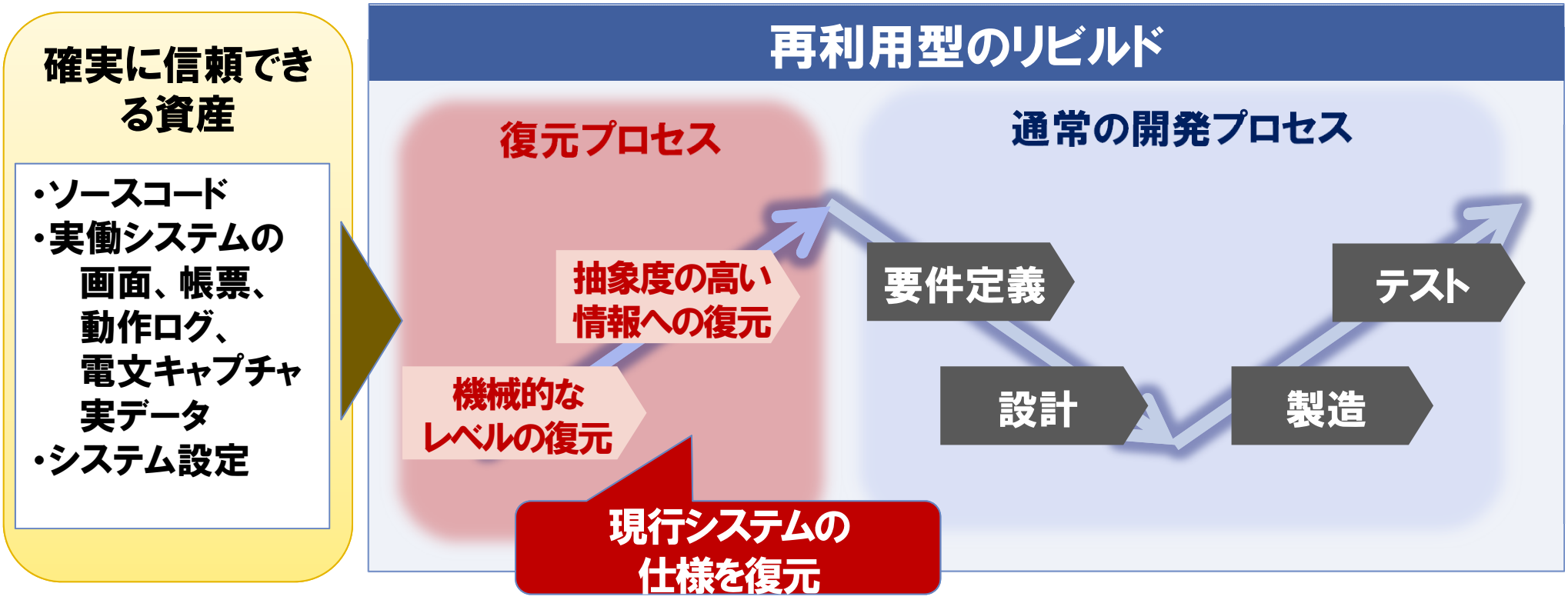
大規模バッチの改善検証

- ファイルベース(バケツリレー)処理からRDB処理への改善
- 重複データや重複処理をしている点の改善
- 並列処理が実現可能な点を改善
- 処理の重いバッチ処理の抽出と改善
  →HadoopやオンメモリDBなどの適用　　等

「可視化」「問題把握」「最適化・チューニング」「検証」の仕組み

② IT化にあわせた開発手順

確実に信頼できる資産から再利用を行うリビルド

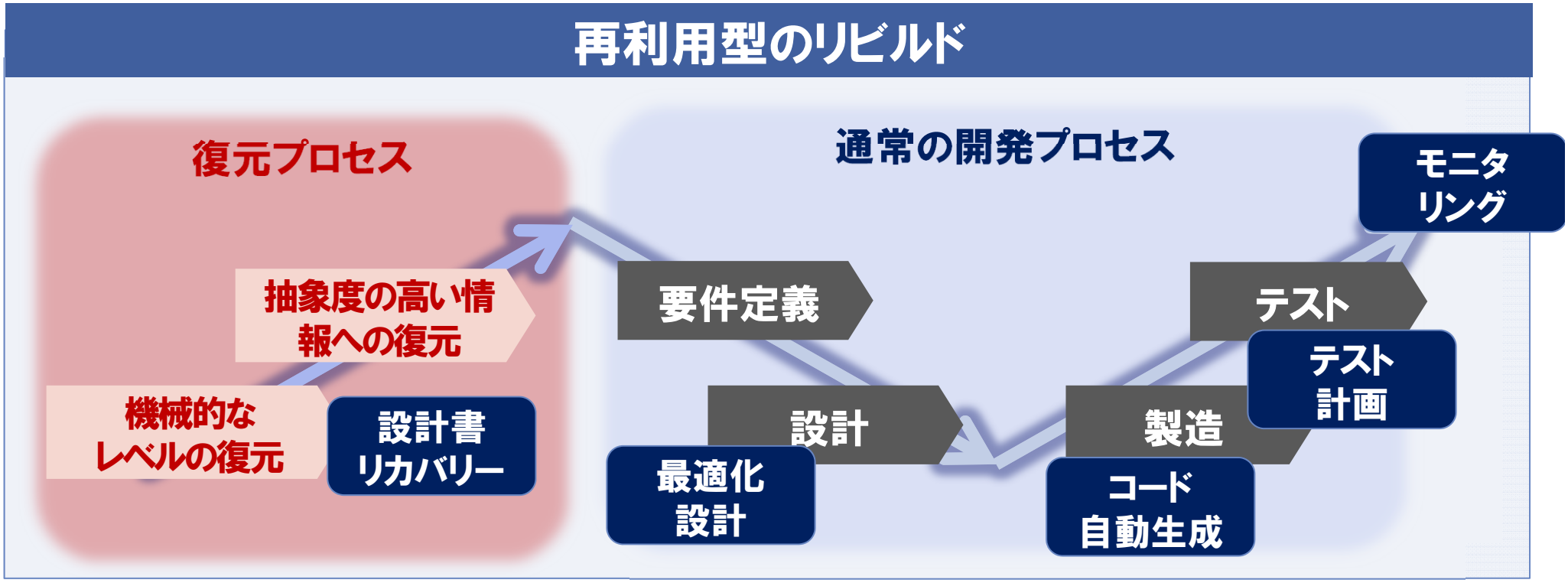


確実性が確保されるものの、はやりコストを要する

② IT化にあわせた開発手順

NTT DATA

様々な工程でIT化を進め、リビルドを短期間で実現する



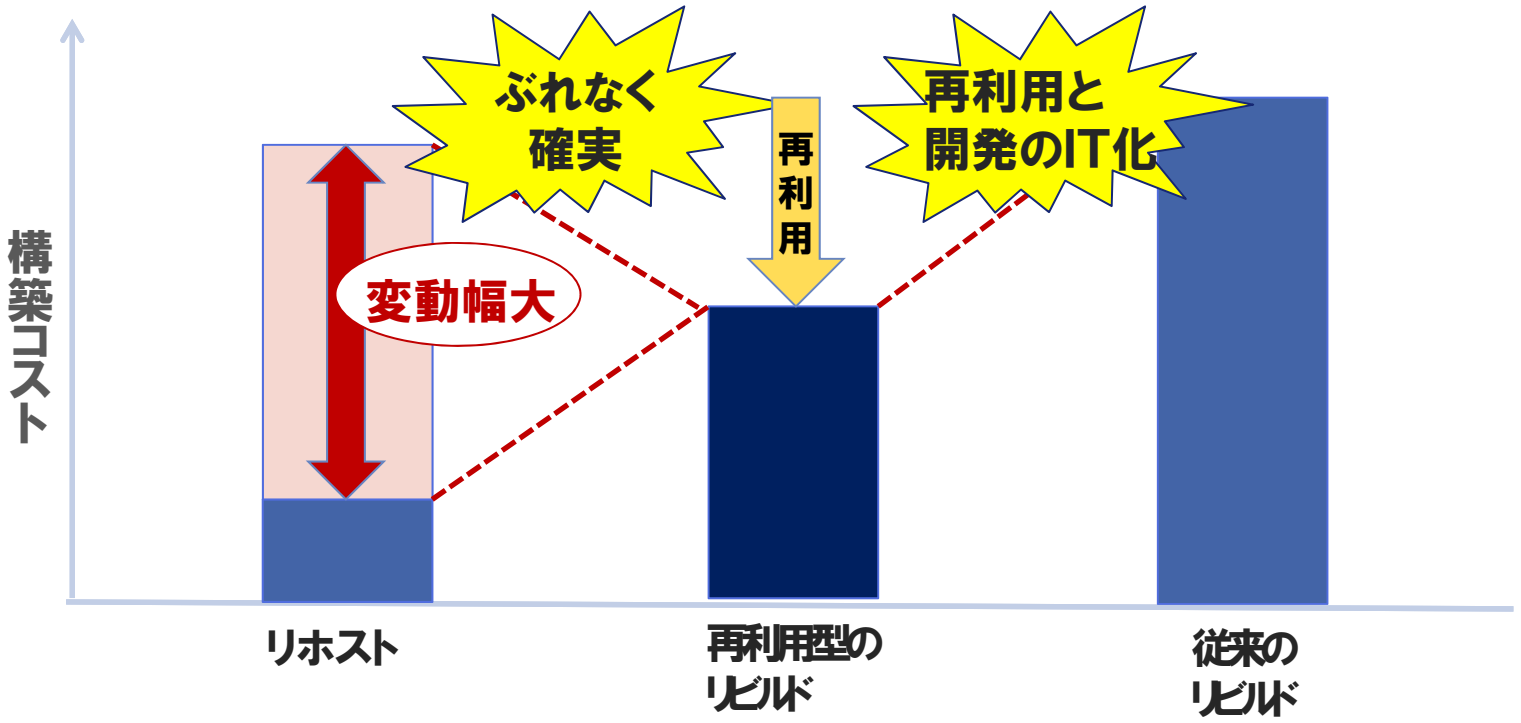
ソフトウェア生産技術の適用

開発自動化ツール、フレームワーク、開発手順、ノウハウ、、、etc

② IT化にあわせた開発手順

NTT DATA

リホストよりも変動幅を抑え、保守性も向上し
従来のリビルドよりも生産性を高める



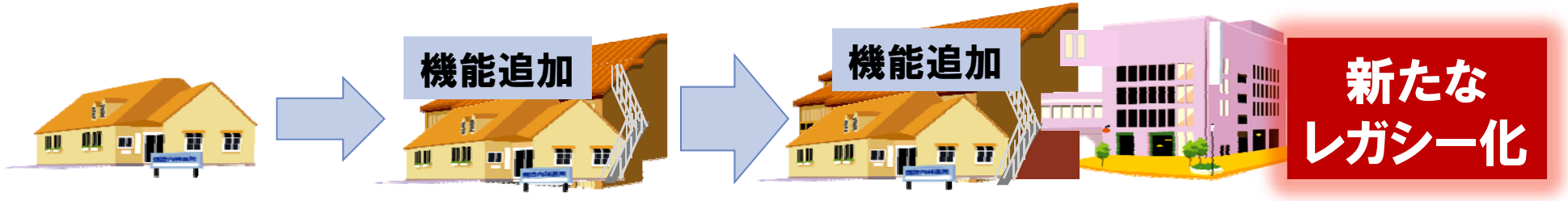
再利用型のリビルドは、確実性の確保と再利用のメリットを享受

③ モニタリングによる保守性低下の防止

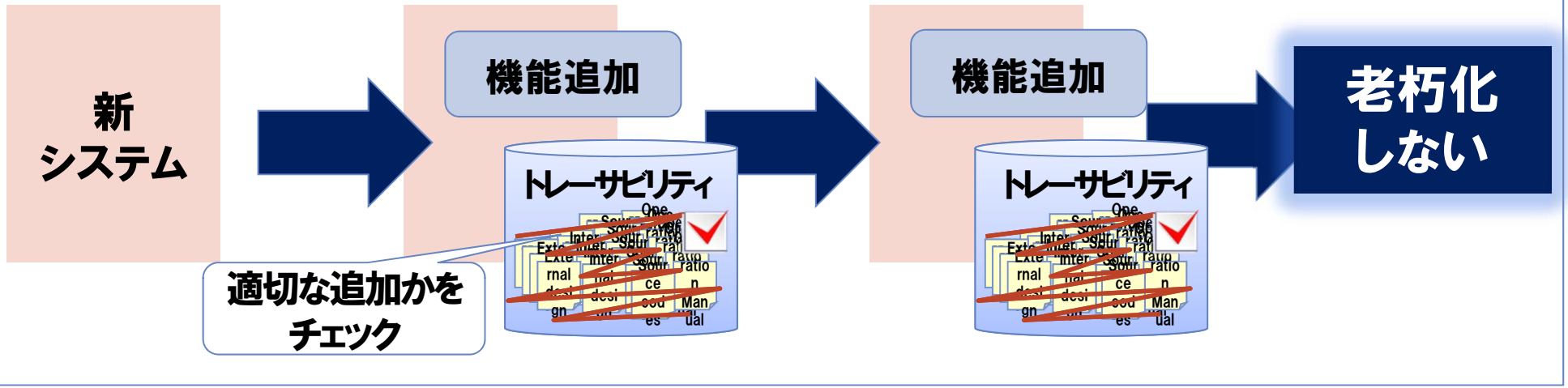
NTT DATA

運用・保守フェーズでのモニタリングとトレーサビリティの実現で、変化をチェックし、新たなレガシー化を防止

開発後に何もせず機能追加をくりかえすと・・・



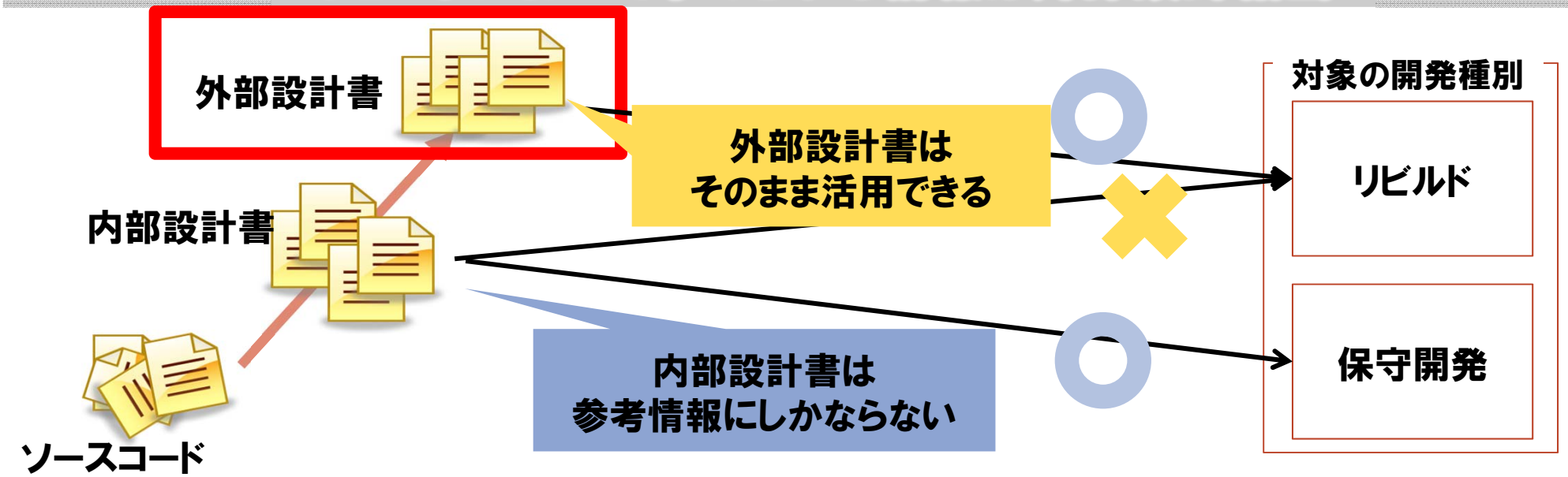
トレーサビリティ機能によりモニタリング




業務仕様の把握に向けて

ソフトウェア生産技術への期待

リビルドにおいて求められる情報は外部設計相当

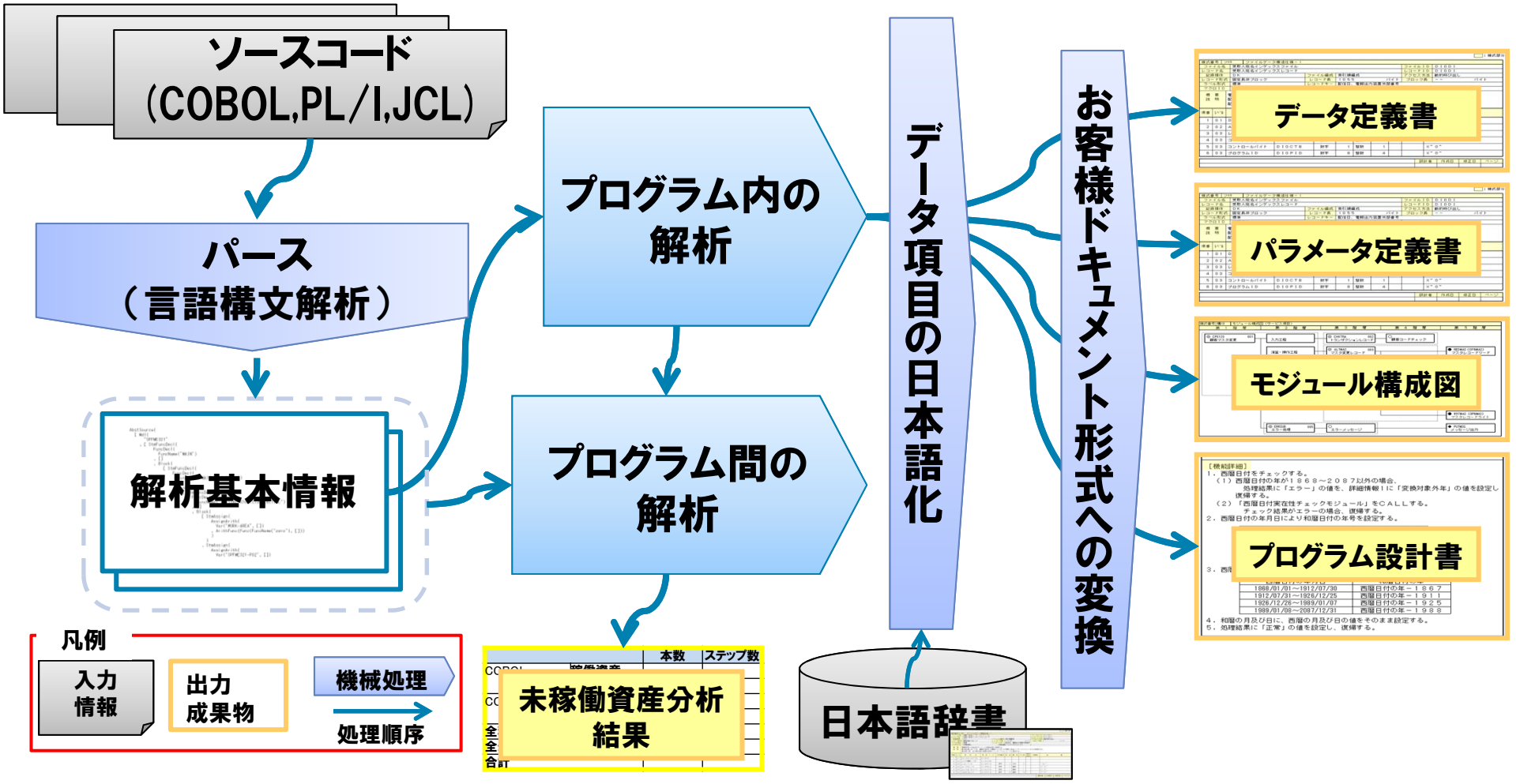


既存システム資産は膨大かつ複雑

ある基幹システムのCOBOLソース（15.5M）

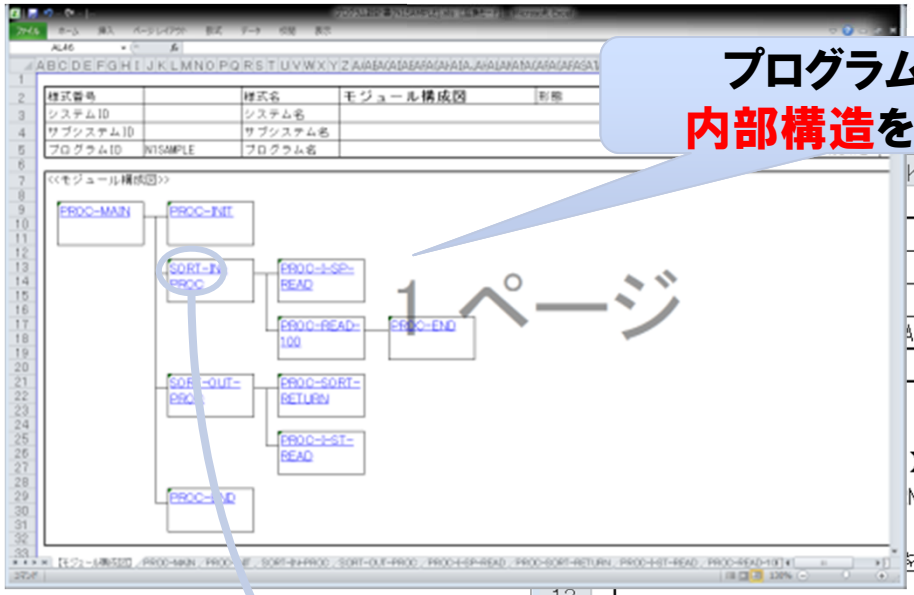
全ソース中の条件の数：76万
全ソース中の命令文の数：420万
全ソース中の変数の利用回数：924万

MF系のソースコードを解析し、各種設計情報を自動回復



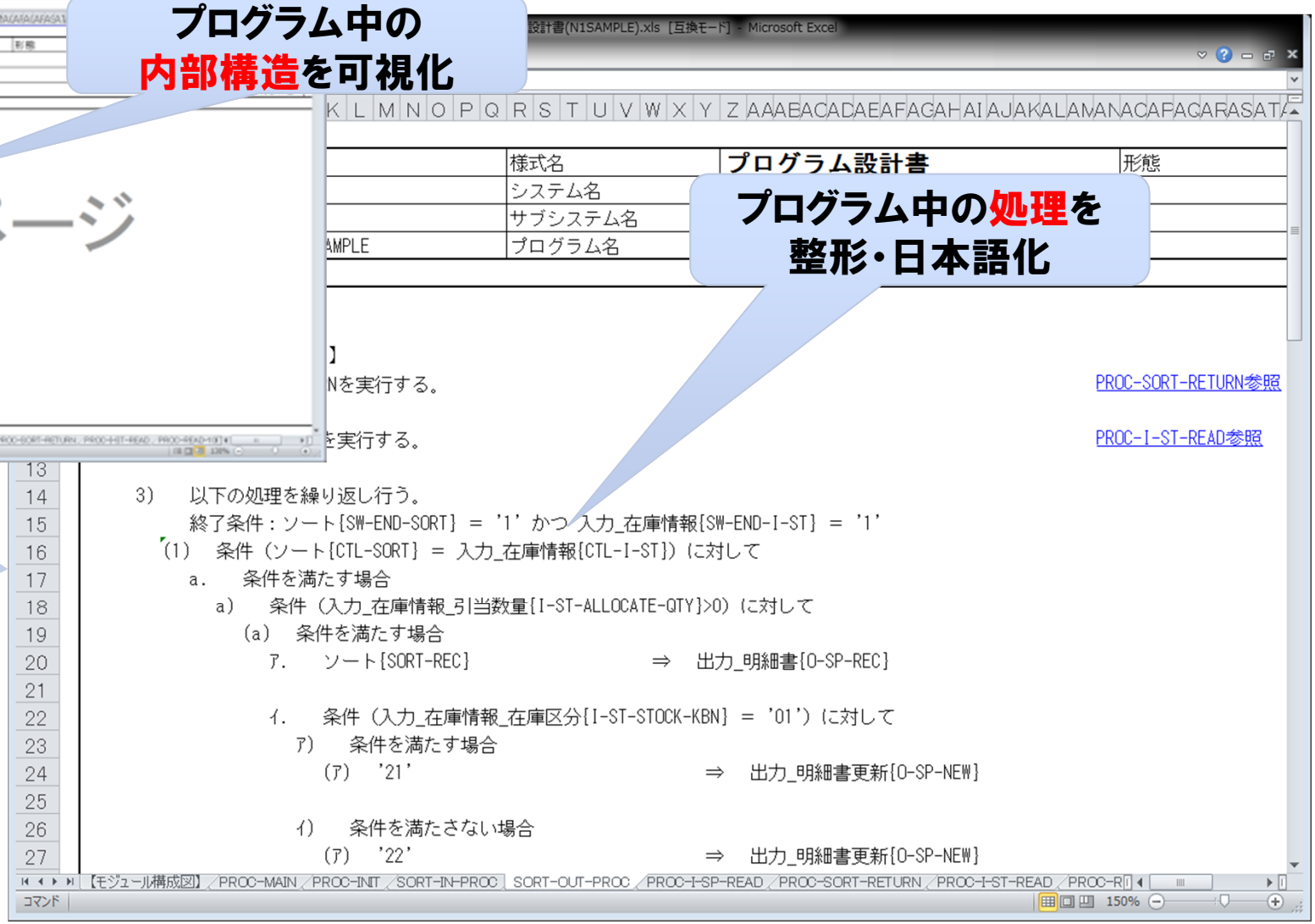
TERASOLUNA Reengineering 既存出力例

現在内部設計書相当の設計書を自動生成



1ページ

プログラム中の
内部構造を可視化



プログラム中の**処理**を
整形・日本語化

TERASOLUNA Reengineeringによる外部設計書の生成

NTT DATA

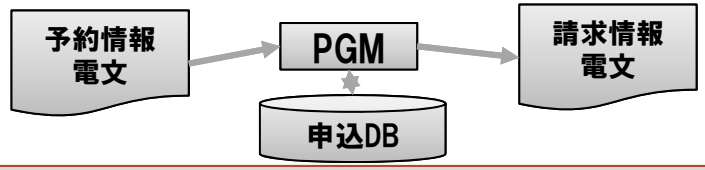
外部設計書相当の出力ツールを開発中

■ 典型的な外部設計書の記載

■ 概要

予約情報を取り込み、支払種別が請求となっている予約について請求情報を作成する

■ 入出力関連図



■ 入力値チェック

チェック条件	エラーコード
請求金額が未設定	E100
請求金額<=0	E110
申込日>請求日	E300

■ 処理ロジック

出力	更新項目	条件
請求情報	有効=0	支払種別 != 請求
	有効=1	支払種別 == 請求
申込情報	金額=予約.金額	なし

プログラムの内部構造・処理ではなく、
どのような入力により、どのような出力になるのかという外部仕様に着目する

プログラムのI/Fファイル・DBを記載

プログラムの入力に対する
チェックルールを記載

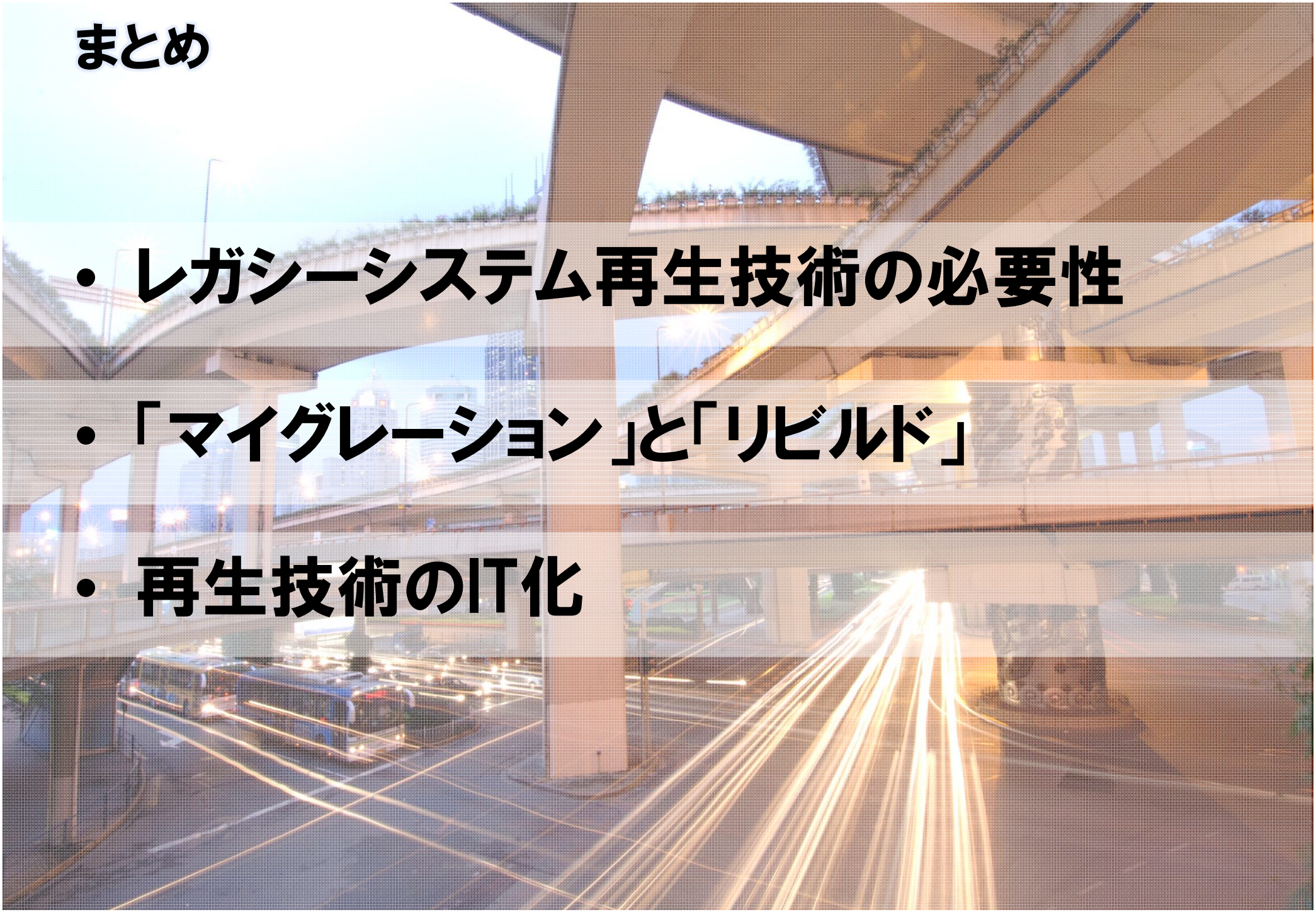
プログラムの入力に応じた
出力内容を記載

外部設計書生成の難しさ

NTT DATA

記号実行技術を中心としたアプローチを行うが研究課題は多い

1. 規模の問題
   - ✓ 1ソース10K～50Kのソースも多く存在
2. ロジックの簡略化
   - ✓ 恒真・恒偽判定、式の簡略化…
3. 業務ロジックと実装依存ロジックの分離
   - ✓ 業務起因のロジックと実装依存のロジックの見分け
4. 業務ロジックの特性の分離
   - ✓ 入力チェックと出力データ生成処理の見分け
5. ロジックの表現方式
   - ✓ 自然言語、決定表、決定木、図表…何が見やすいのか
   - ✓ どのような単位で分割すると見やすいのか
6. 。。。



まとめ

- レガシーシステム再生技術の必要性
- 「マイグレーション」と「リビルド」
- 再生技術のIT化

おわりに

・外部設計書生成など、ソフトウェア工学の活用が期待できるテーマが、レガシーシステムの再生には数多く存在します。
・ぜひ、この分野の研究に取り組んでみませんか？

NTT DATA

Global IT Innovator