SPI Japan 2025

大規模なLowCode開発を実践し得られたノウハウ・教訓

2025年10月22日

株式会社NTTデータグループ　ITマネジメント室
システム開発担当　伊丹　淳

Agenda

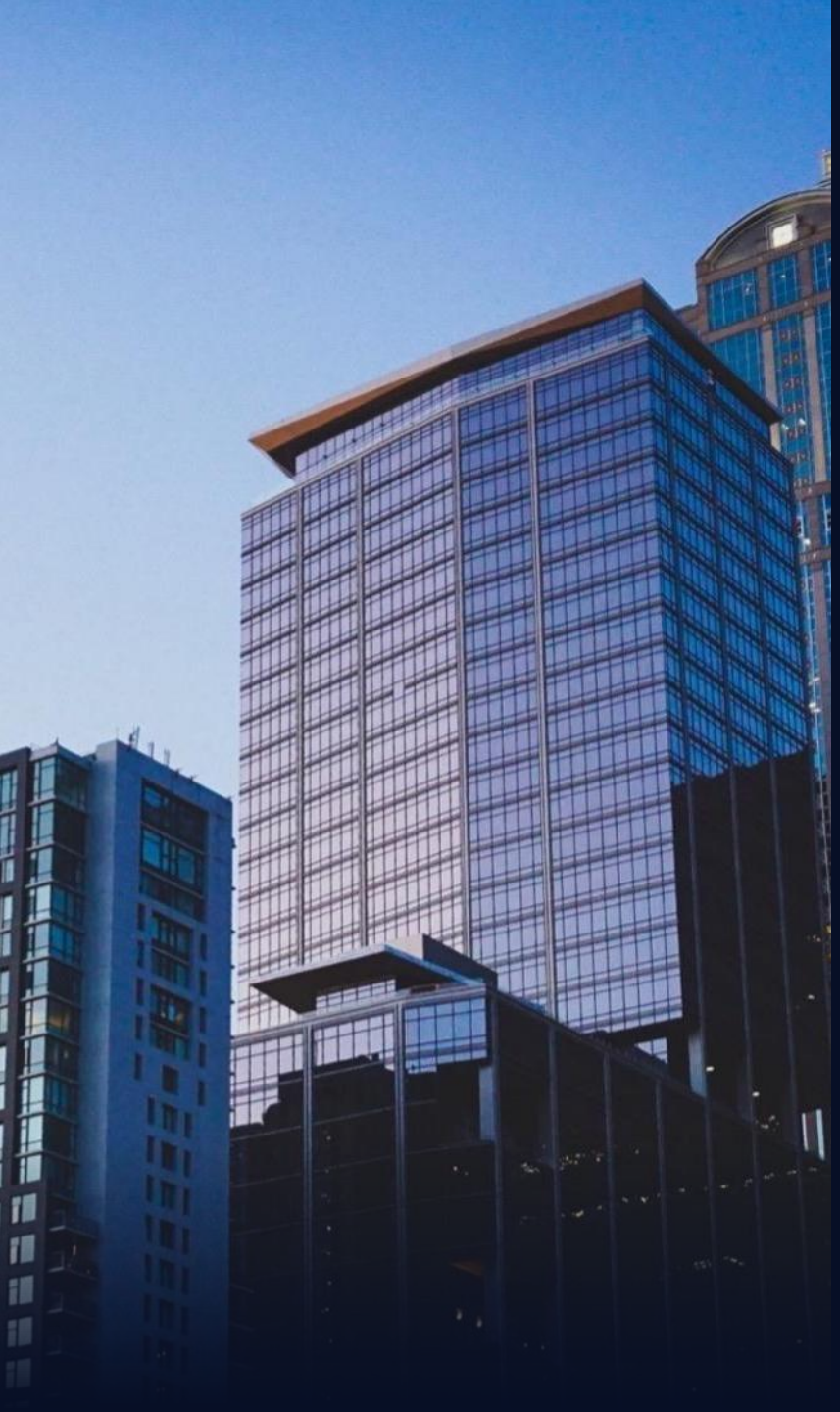
第1章　前提

1. プロジェクト概要
2. 高難度特性
3. 新技術・ローコード適用の理由

第2章　本編

1. 開発プロセス定義
2. 技術系の取り組み
3. 開発プロセス系の取り組み
4. 適用効果
5. まとめ

本発表は、主にServiceNowにおける設計、および実装について説明しています。



第1章 前提

1. プロジェクト概要
2. 高難度特性
3. 新技術・ローコード適用の理由

1. プロジェクト概要

基幹系システムの更改にあたって新技術・ローコード開発を適用した大規模なプロジェクト。

大規模

- 現行17のサブシステムを統合する基幹システム
- 開発体制は1,500名以上

高難度

- プロジェクト単位の個別原価制度
- 複数の申請の一括化
  （意思決定システムと会計システムとのシームレスな連動）
- 現行システムとの連携
  （経営分析、購買、人事等）

新技術・ローコード開発

- ServiceNowを積極活用
  （導入時点で大規模適用事例などのノウハウは少ない状況）
- 複数のクラウドサービスとの連携

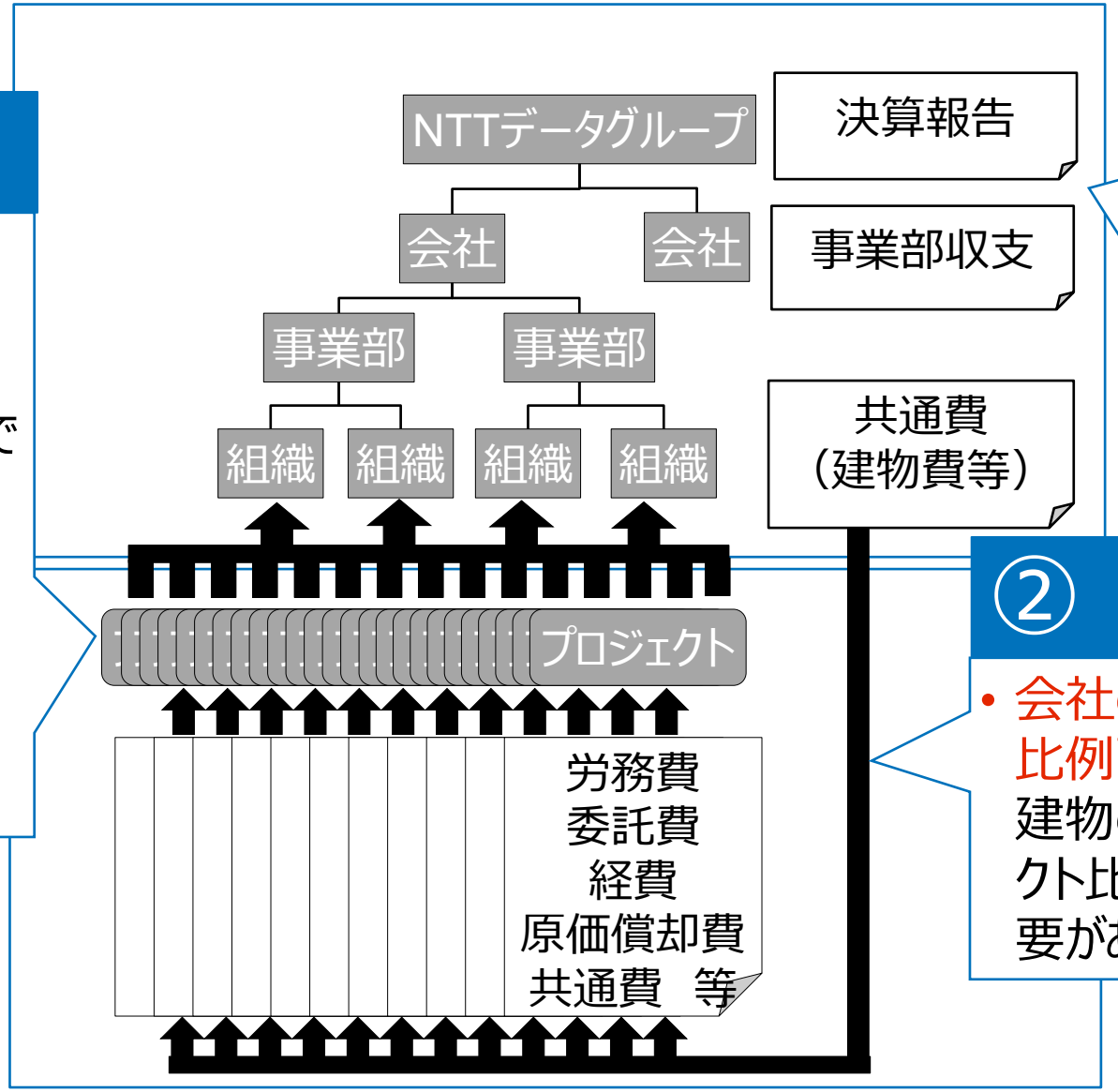
（参考）
- 利用範囲：NTTデータグループ会社(70社)
- ユーザ数：最大50,000ユーザ
- アプリケーション製造規模：44,000FP
- レコード数：最大20臆
- サブシステム数：17

2. 高難度特性（1/3）

本プロジェクトが高難度である理由は、NTTデータグループの会計制度にある。

① 会計管理はプロジェクト単位

- 個々のプロジェクト単位で全収支を管理
  例えば、組織間で要員支援する場合、稼働を時間単位で付け替え処理が必要
- 膨大なプロジェクト件数
  NTTデータグループ全体で社員数3万人に対し、発生するプロジェクト件数は毎年10万件以上



② 会社の共通費も全プロジェクトに分配

- 会社の共通費もプロジェクトに比例配分
  建物の賃貸料等も個々のプロジェクト比例配布し、利益管理する必要がある

③ 経営分析は会社単位で集計

- 膨大な会計情報を誤差なく集計
  10万件のプロジェクトの会計情報を1円の誤差なく組織単位で集計、報告する必要がある

2. 高難度特性（2/3）

17のサブシステム間で情報連携し、整合性を確保する必要がある。

サブシステム群
（17サブ）

- 案件管理
- 意思決定
- ・・・
- 請求管理
- 経費精算　等

各サブシステムは
開発体制
20～100名の
規模感

現行システムの課題

①複数のサブシステムへの申請
プロジェクト開始時に複数申請作業が必要
- ・受注の意思決定
- ・使用する勘定科目の設定
- ・プロジェクト情報の申請　等

②添付ファイルの目視チェック
データ登録時にファイルを添付する個所があり、記載内容の目視確認が必要

今回の改善ポイント

①一括申請の実現
一括登録を可能にすることでユーザの負荷を軽減する

②チェックの自動化
システムにデータとして登録、チェックを自動化する

これを実現するため
サブシステム間での
情報連携、
整合性を確保する
ことが必要

2. 高難度特性（3/3）

ServiceNowを用いた開発において、世界最大規模として紹介されています。

＜ServiceNowのご紹介＞

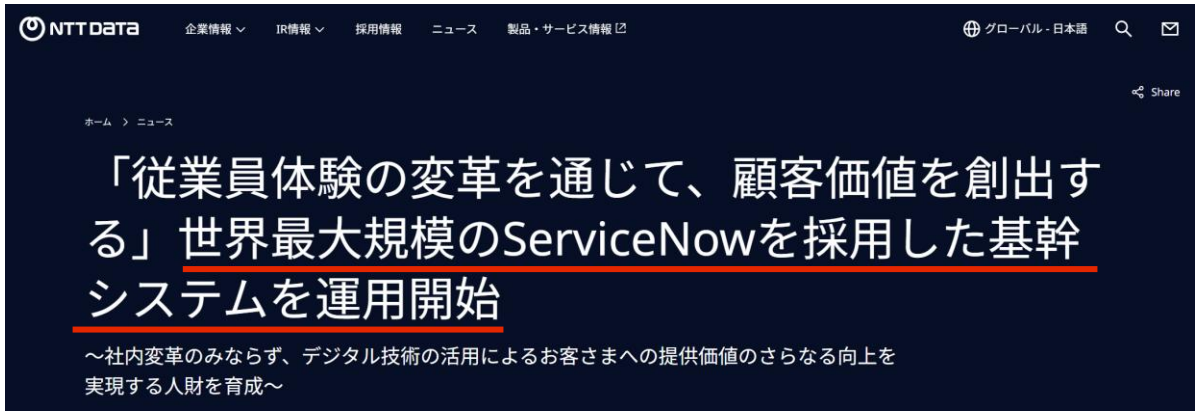
https://www.servicenow.com/jp/customers/nttdatagroup.html

国内大手のシステムインテグレータ（SIer）として、公共・金融・法人の各分野で数多くのシステム開発を手掛けるNTTデータグループ。急速なデジタル化の進展を受け、自社システムにもデジタル技術を適用し、業務改革を目指す全社プロジェクト「Project GAIA」を展開しました。中でも注目されるのが、ServiceNowによる開発では世界最大規模※となる巨大な基幹システム、「GAIA.fin」（ガイア フィン）の構築です。

会社全体の決算はもちろん、各プロジェクトのマネジメントや意思決定のためにも用いられる非常に重要なシステムのため、これまではスクラッチで開発してきました。それをなぜ、NTTデータグループはSaaSであるServiceNowで構築することにしたのでしょうか？

＜NTTデータグループのニュース＞

https://www.nttdata.com/global/ja/news/topics/2024/042601/



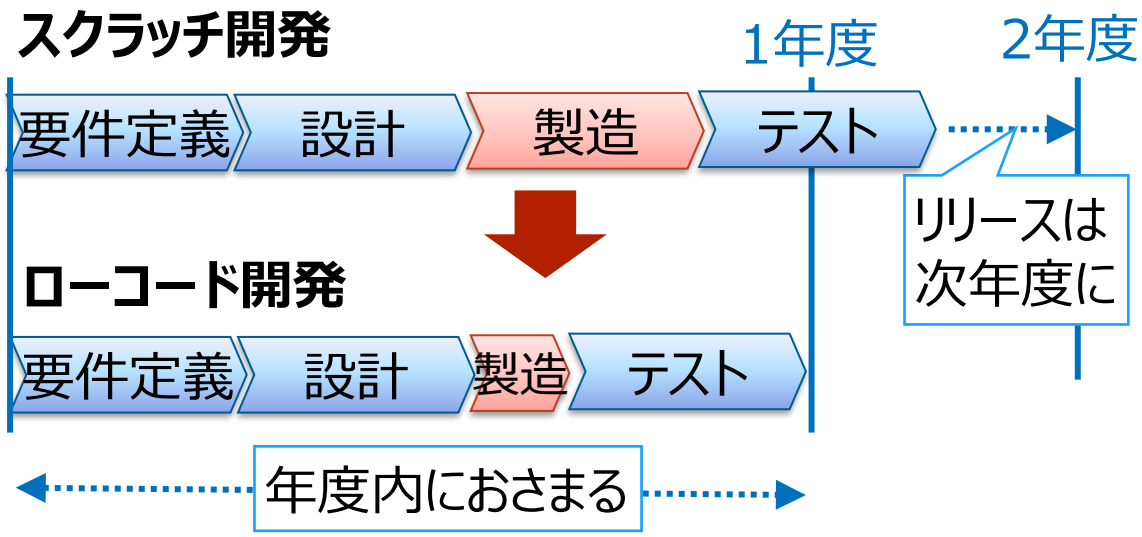
2024年4月26日
株式会社NTTデータグループ

株式会社NTTデータグループ（以下：当社）は、Project GAIA （以下：当プロジェクト）[注1]の業務プロセス変革の中心として位置づけられた、GAIA.fin[注2]において、ServiceNowを採用した基幹システム（以下：当システム）を運用開始します。
ServiceNow社が提供するローコードプラットフォーム「App Engine」を活用し、社内業務システムを統合した一元的なフロントエンドを構築し、効率的な業務プロセスを実現しました。当システムは、ServiceNowを活用したシステムにおいて開発規模および実行環境として世界最大規模です。
2024年4月初旬に一部サービスを先行して運用開始し、5月初旬に全面的な運用開始を予定しています。今後、国内グループ会社にもサービスを展開する予定です。

3. 新技術・ローコード適用の理由

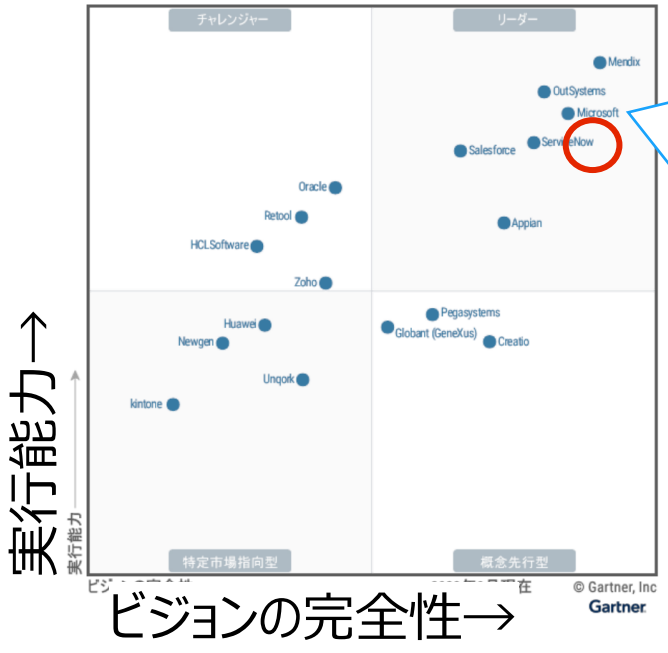
現行の技術が老朽化、中長期的な視野で見た新技術の導入が必要。

製造期間の圧縮のため、ローコード開発を採用



- 会計システムは年度単位のリリースに収めることが必須条件
- 仕様が複雑であることから、要件定義/設計/テストフェーズは従来同様の時間を要する

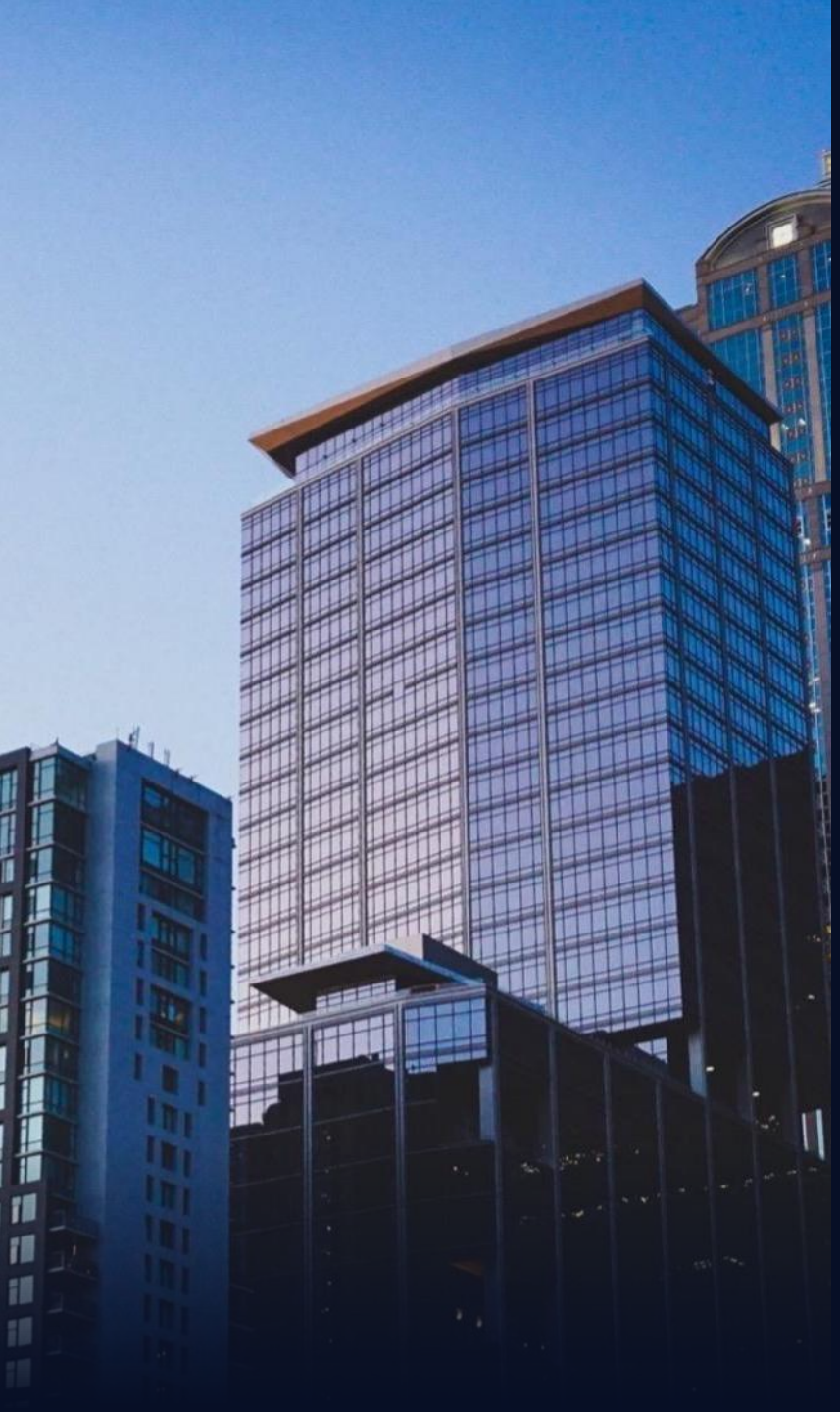
ベンダからの長期サポートの継続が見込まれる製品を採用



ガートナーレポートで
ローコード開発において
ServiceNowが今後
リーダポジションをとると
言われ、現在、海外・
国内でも様々な開発
事例が増えている

出典：ガートナー「エンタプライズ・ローコード・アプリケーション・プラットフォームのマジック・クアドラント」2023年8月発行

- 大規模なシステムのため、更改後は長期間に渡り継続した運用が必要
- 現行システムはSAPを採用していたが、更改から10年以上経過、バージョンアップによる保守期限間近

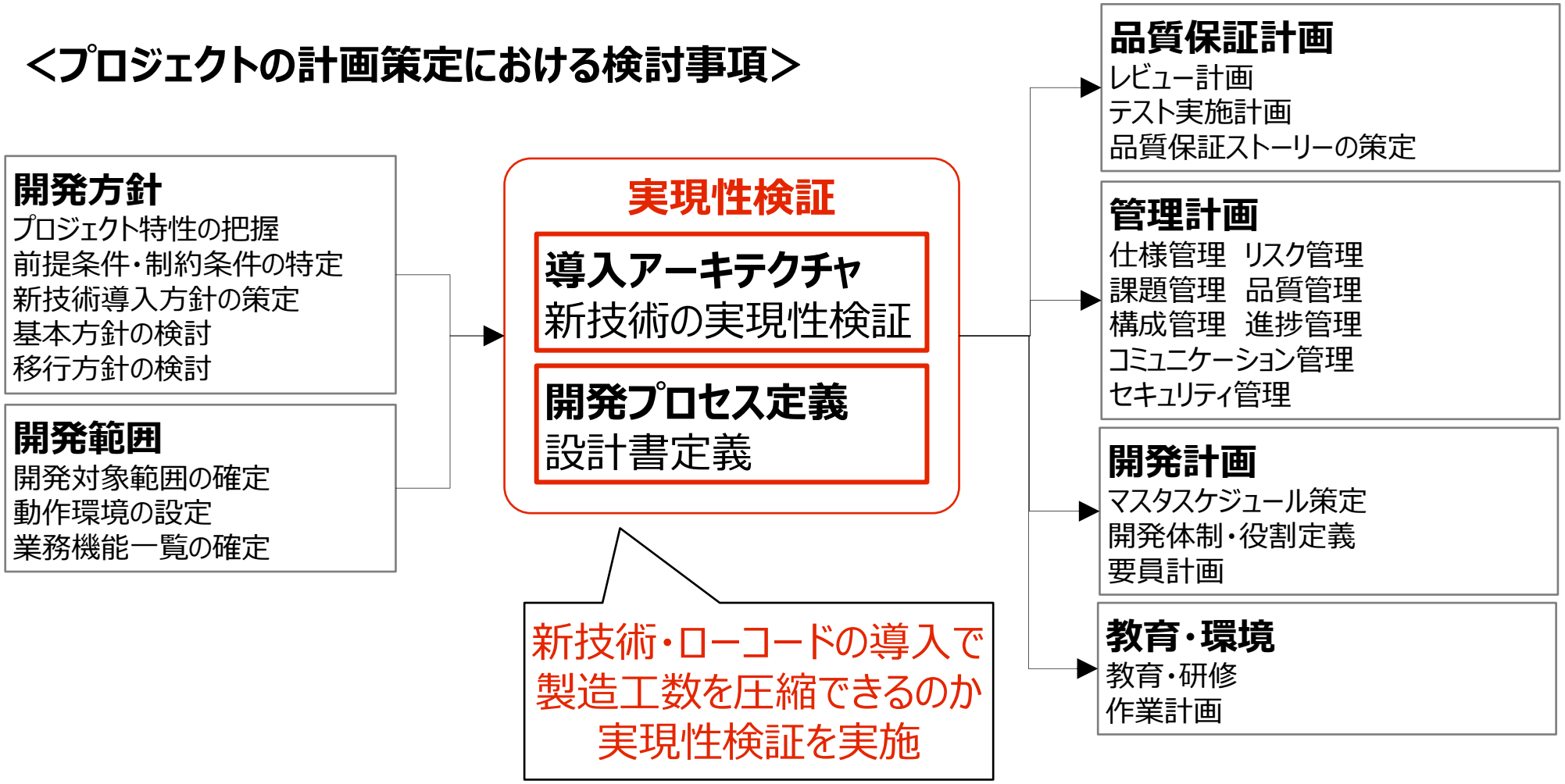


第2章　本編

1. 開発プロセス定義
2. 技術系の取り組み
3. 開発プロセス系の取り組み
4. 適用効果
5. まとめ

1. 開発プロセス定義

プロジェクト計画の策定にあたり、開発プロセスの定義は、他の検討事項の前提条件となるため、
特に新技術を用いる場合は、計画のフィジビリティ確保のために、実現性検証が必要になる。



1. 開発プロセス定義　実現性検証（1/2）

ServiceNowはコーディングすることで様々な要件に柔軟に対応することができるが、ローコード開発のメリットを活かすためには、不必要なコーディングを避け、ServiceNow標準機能を活用できるよう仕組みを整備することが重要になる。

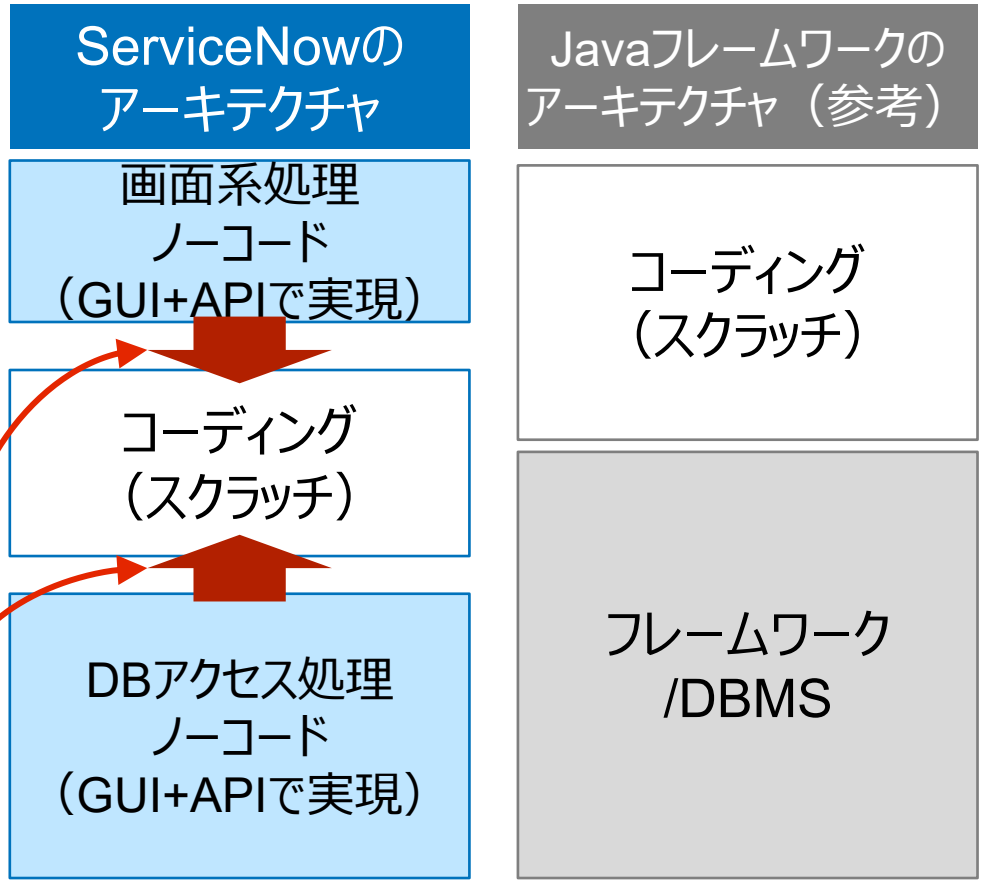
ServiceNowの特徴

ServiceNowを使うメリット

- ServiceNow標準機能を活用することでコーディングを削減できる
- 一方、ServiceNow標準機能が要件を満たさない場合、コーディングすることで柔軟に要件に対応できる

メリットを活用するための注意事項

- 実現性検証の結果、メンバがServiceNow標準機能を知らないと、これを使わずにコーディングで対応してしまうため、コーディング量が増える傾向になることが見えてきた
- できる限りServiceNow標準機能を使い、不必要にコーディングしないようにすることが重要



1. 開発プロセス定義　実現性検証（2/2）

ServiceNowはクラウドサービスを前提とするプラットフォームであるため、従前のJavaフレームワークによるWebアプリケーションとは設計思想が異なっているため、フレームワークレベルのコーディングを含む仕組みを整備することが重要になる。

ServiceNowの特徴

ServiceNowを使うメリット

- APサーバを後から容易に拡張、スケールアウトできる
- 複雑になる性能問題をスケールアウトで回避することができる

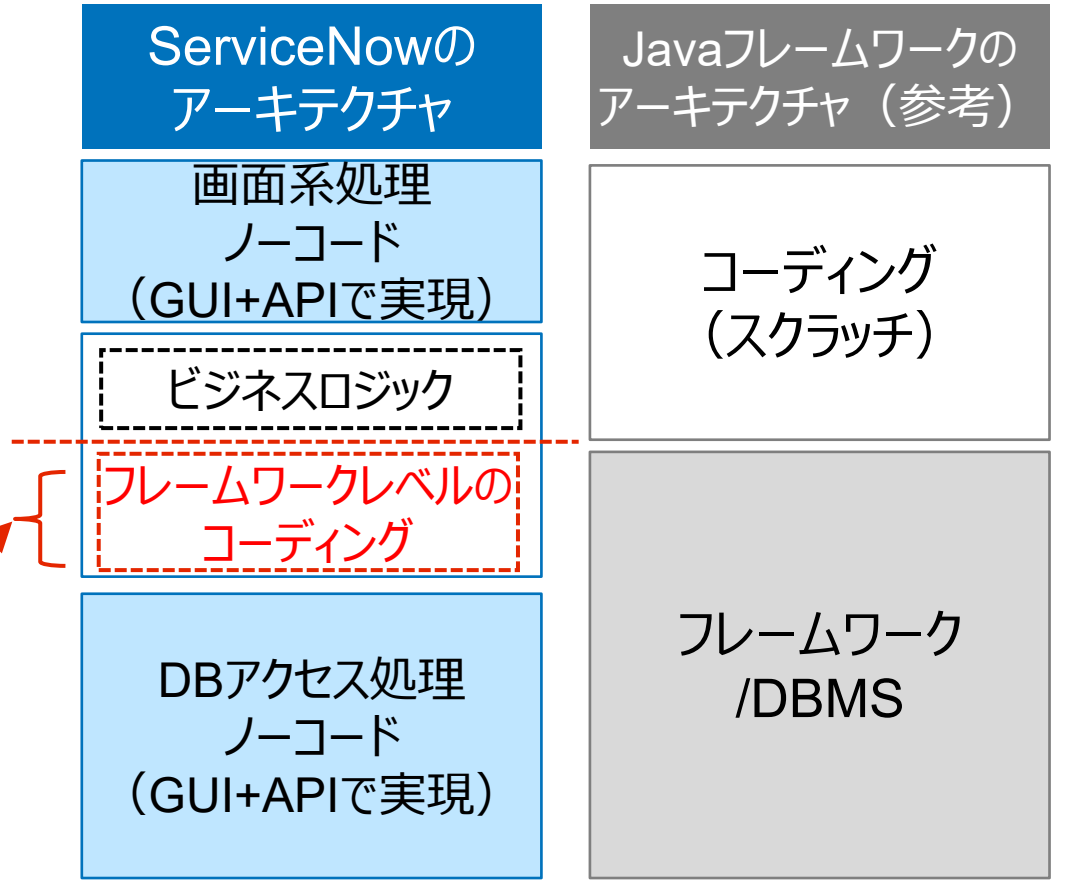
メリットを実現するための手段

- マイクロサービス化
  - ➢ 大量のトランザクションを短時間で処理するために、ショートトランザクションでの実装を推奨
- DB依存からの脱却
  - ➢ DBは基本単一インスタンスでスケールアウトの制約になるため、DBに依存する機能は回避、AP側でトランザクションを制御する仕組み

ServiceNowを使う上での制約

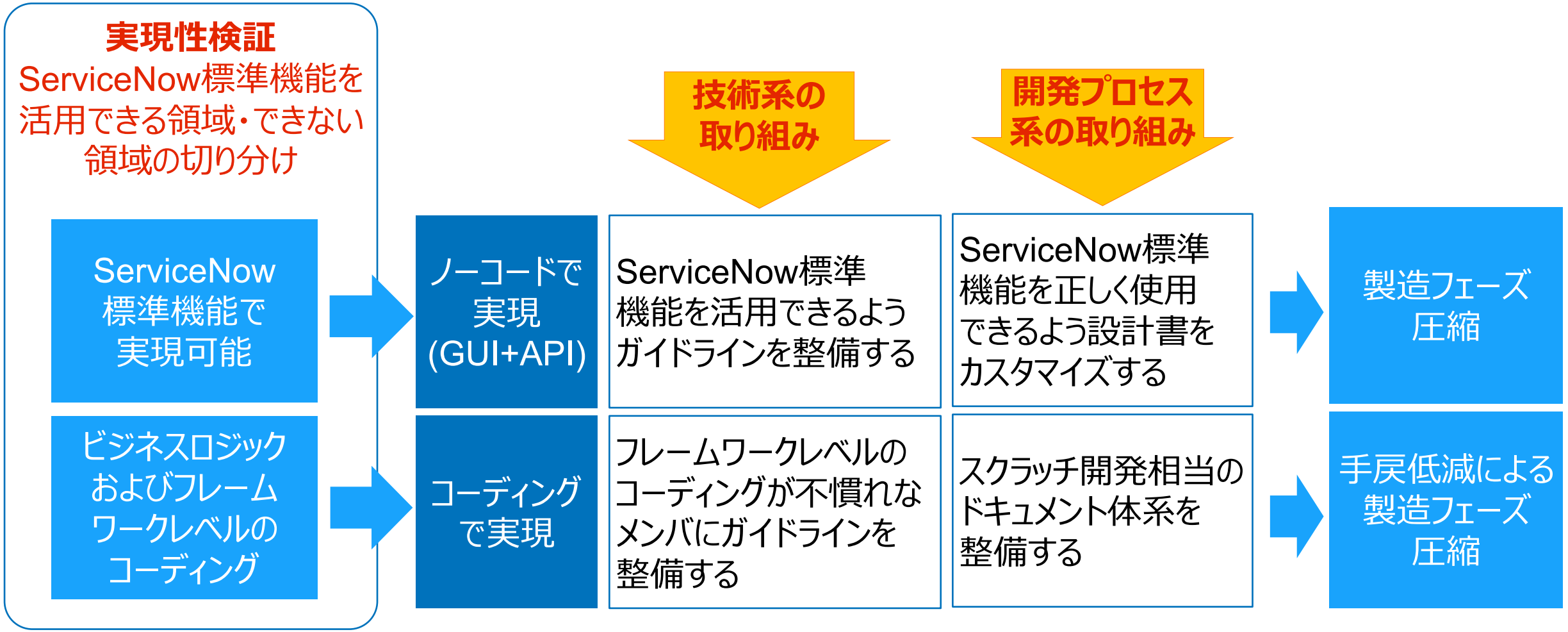
- 実現性検証の結果、DBに依存する複雑なトランザクション処理等はコーディングが必要であることが見えてきた

（例）select for update、not null制約が使えない等



1. 開発プロセス定義　実現性検証を踏まえた取り組み

複雑な業務要件に対し、ServiceNow標準機能を活用できる領域・できない領域を実現性検証によって見極め、技術系の取り組みとしてガイドラインの整備、開発プロセス系の取り組みとしてドキュメント体系を整備した。



2. 技術系の取り組み

大規模プロジェクト、複数チームが並走する体制、高スキル要員確保が困難であるため、
ServiceNow標準機能を活用、コーディングを統制できるようガイドラインを整備、一定品質を確保、手戻りを低減した。

技術系の取り組み

ノーコードで実現(GUI+API)	ServiceNow標準機能を活用できるようガイドラインを整備する
コーディングで実現	フレームワークレベルのコーディングが不慣れなメンバにガイドラインを整備する

ガイドライン目次

- 標準画面遷移パターン
- ネーミング規約
- プロパティ管理
- メッセージ管理
- 基本画面要素
- 表示制御

- 排他制御
- バッチ処理
- トランザクション制御
- エラーハンドリング
- 性能
- 削除方式

コーディングガイドライン整備し
チケットとして全チームに展開、
一定品質を確保、手戻りを低減した

2. 技術系の取り組み　①排他制御

ServiceNowは、APサーバの拡張による容易なスケールアウトの実現のため、その制約となるDBMSの排他制御（select for updateによるレコードロック等）は使えない仕組みとなっている。**排他制御が必要な場合、業務の入力チェック仕様を用いて排他の仕組みを実現する。**

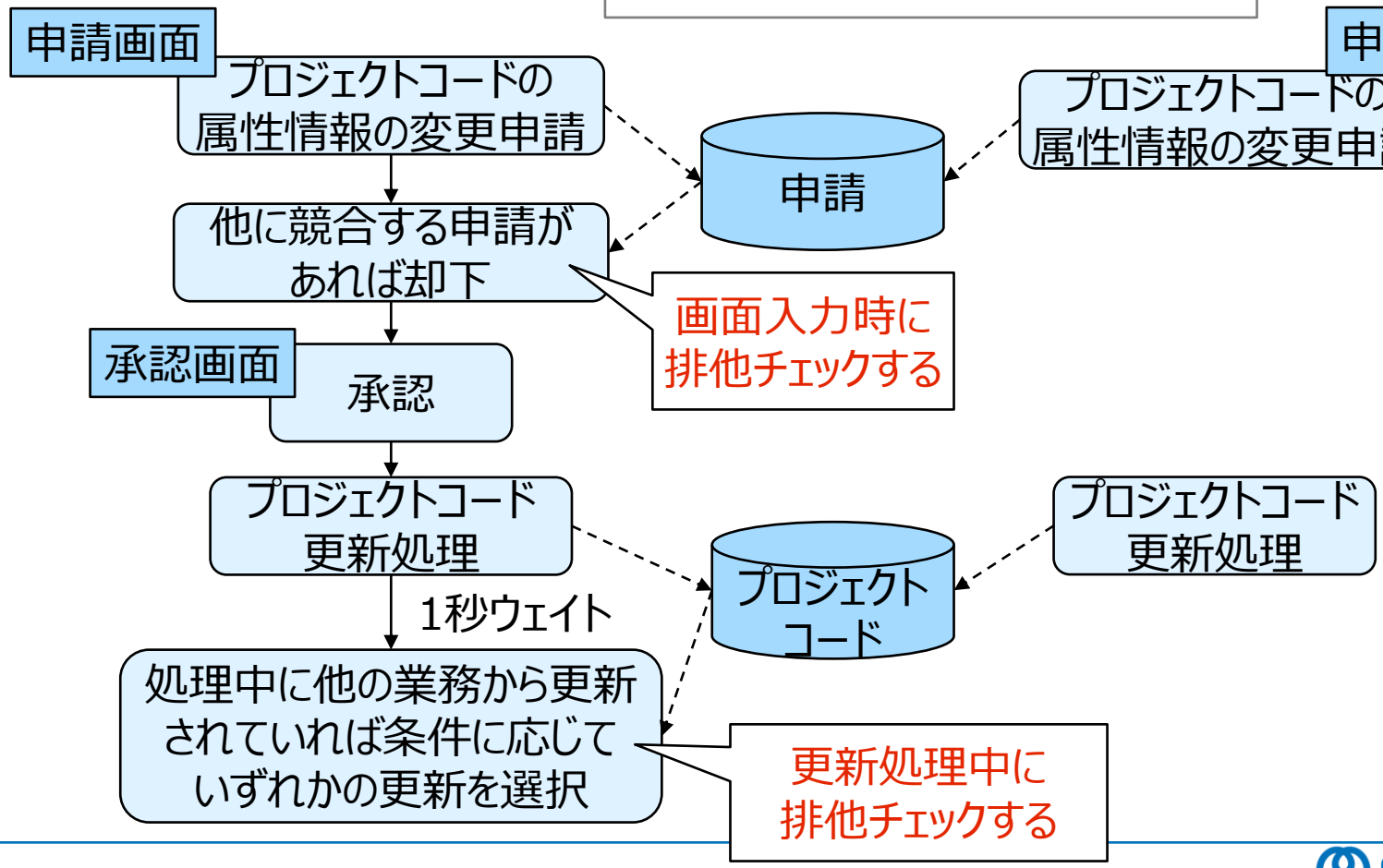
ガイドライン目次

- 標準画面遷移パターン
- ネーミング規約
- プロパティ管理
- メッセージ管理
- 基本画面要素
- 表示制御
- **排他制御**
- バッチ処理
- トランザクション制御
- エラーハンドリング
- 性能
- 削除方式

本プロジェクト

多人数による申請/承認フロー

＜排他制御の実装例＞



2. 技術系の取り組み　②バッチ処理

ServiceNowにはジョブスケジューラのような仕組みはないため、**バッチ処理における処理の多重化、進捗状況の監視、リトライ等の制御を設計・実装する必要がある。**

ガイドライン目次

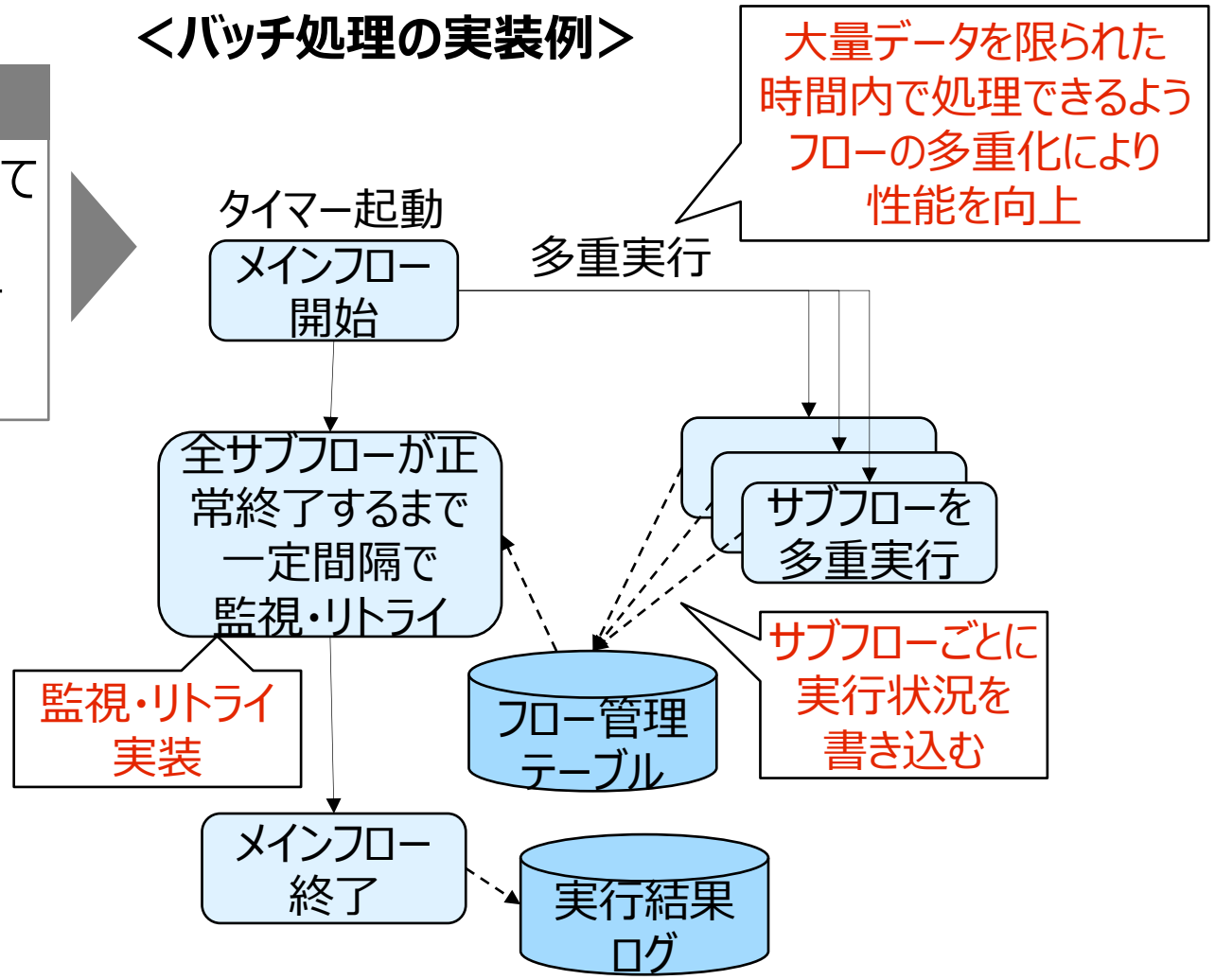
- 標準画面遷移パターン
- ネーミング規約
- プロパティ管理
- メッセージ管理
- 基本画面要素
- 表示制御

- 排他制御
- **バッチ処理**
- トランザクション制御
- エラーハンドリング
- 性能
- 削除方式

本プロジェクト

既存システムとの連携にて膨大なデータを限られたスケジュールで一括集計する必要あり

＜バッチ処理の実装例＞



2. 技術系の取り組み　③標準画面遷移パターン

本プロジェクトでは大量の画面を並走して設計するため、手戻りにならないよう**ユースケースに応じた画面遷移パターンをガイドラインに定義し、UIを統制を図るようにした。**

ガイドライン目次

- **標準画面遷移パターン**
- ネーミング規約
- プロパティ管理
- メッセージ管理
- 基本画面要素
- 表示制御

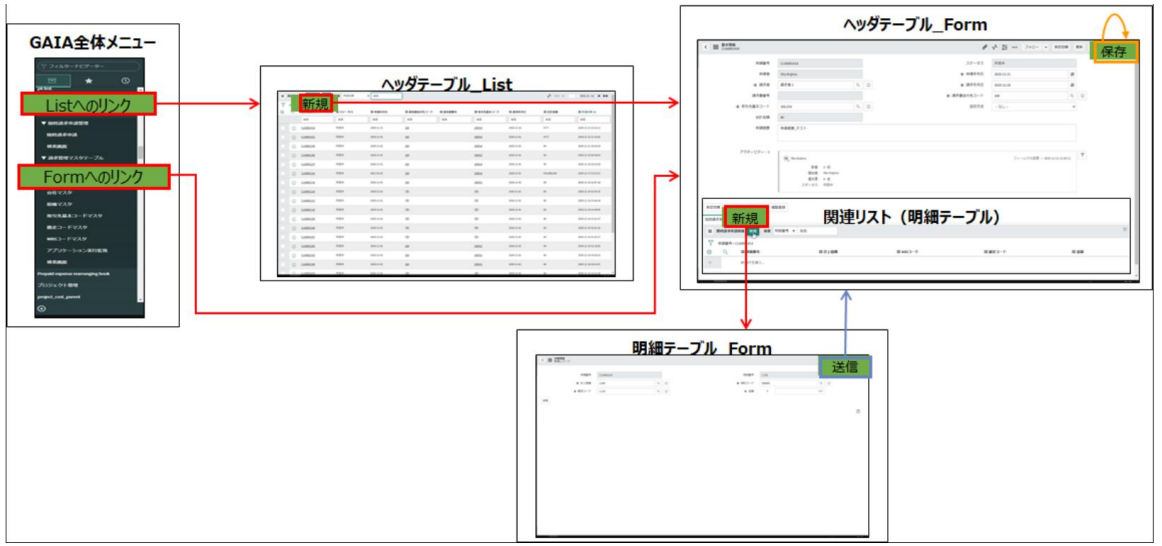
- 排他制御
- バッチ処理
- トランザクション制御
- エラーハンドリング
- 性能
- 削除方式

ユースケースごとに標準画面遷移パターンを設計、ガイドラインに記載、
事前に展開することで、UIの統一感を確保し、手戻りを防いだ

ユースケース

- 一般単票
- 複数テーブル
- 申請/承認
- 一括登録　　等

＜複数テーブルにおける標準画面遷移イメージ例＞

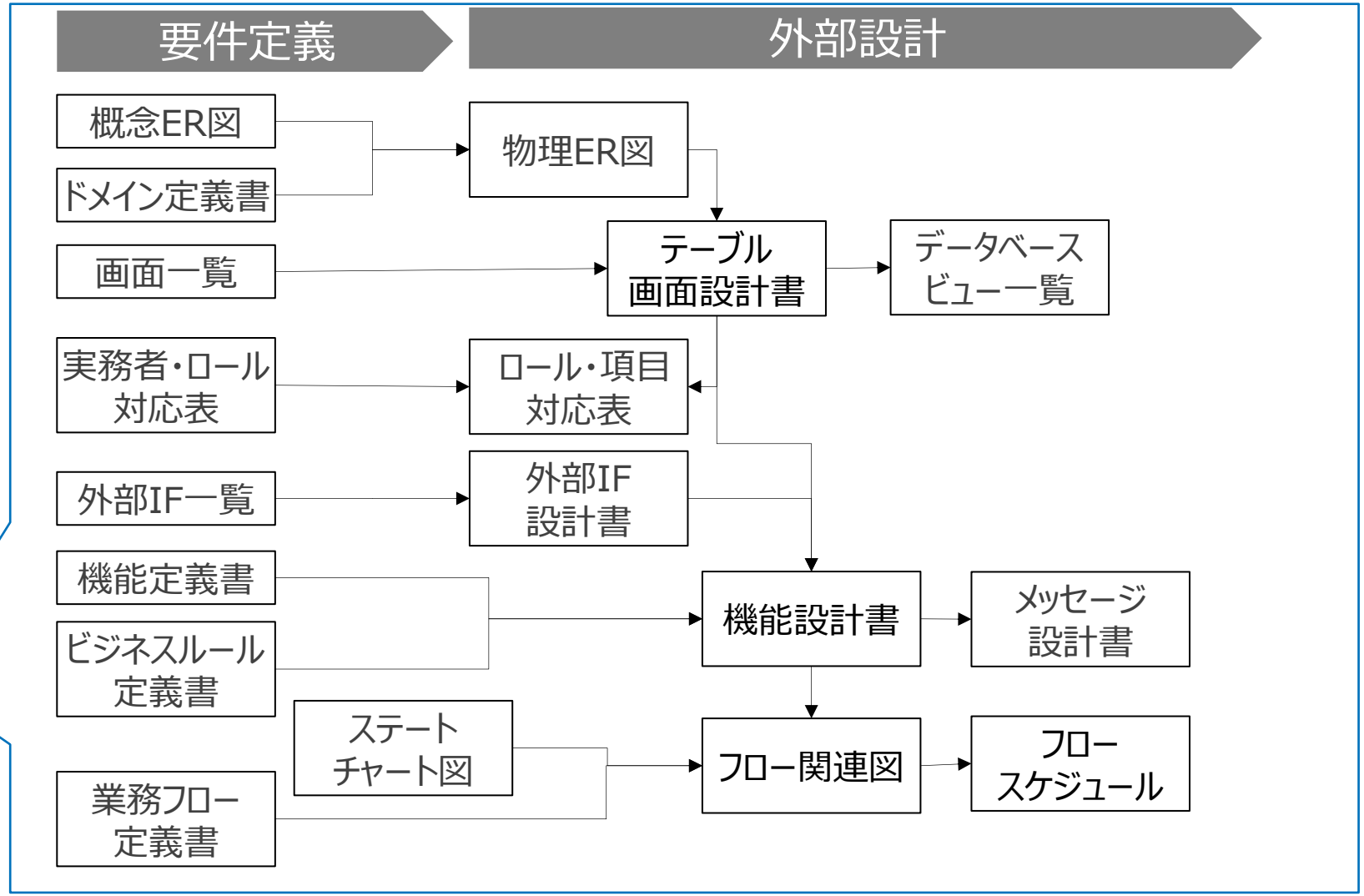


3. 開発プロセス系の取り組み（1/2）

大規模プロジェクトにおいては意思疎通のため設計書が必要になるため、
スクラッチ開発相当のドキュメント体系をベースに整備した。

開発プロセス系の取り組み

ノーコードで実現(GUI+API)	ServiceNow標準機能を正しく使用できるよう設計書をカスタマイズする
コーディングで実現	**スクラッチ開発相当のドキュメント体系を整備する**

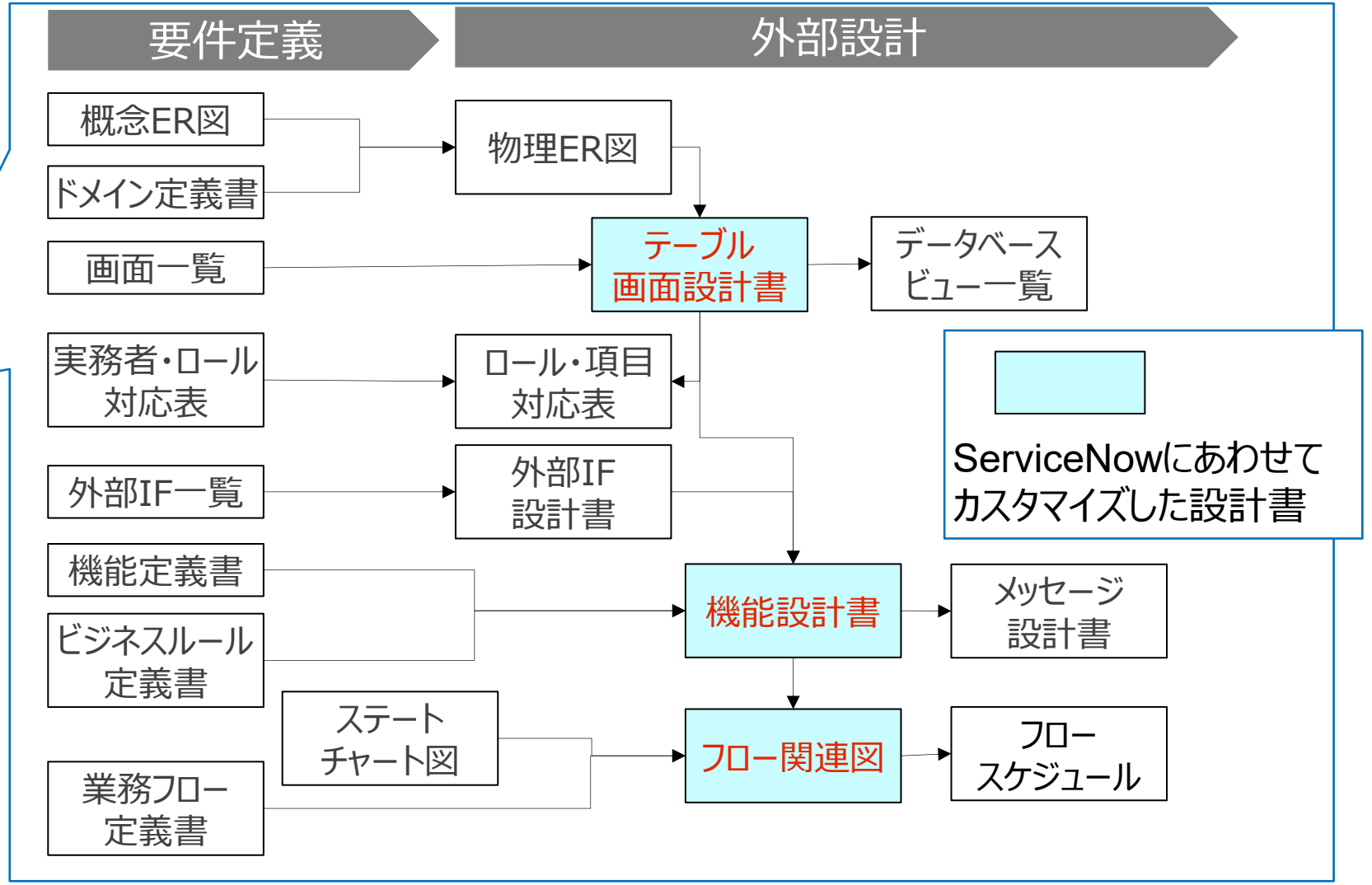


3. 開発プロセス系の取り組み（2/2）

ServiceNow標準機能を正しく使用できるようスクラッチ開発相当のドキュメント体系をベースとしつつ
ServiceNow特性に応じてテーブル、画面、機能に関わる設計書についてカスタマイズした。

開発プロセス系の取り組み

ノーコードで実現(GUI+API)	**ServiceNow標準機能を正しく使用できるよう設計書をカスタマイズする**
コーディングで実現	スクラッチ開発相当のドキュメント体系を整備する



3. 開発プロセス系の取り組み　①機能設計書（オンライン処理）

数百名の体制でスキル差がある要員が誤った実装により手戻りを招くリスクを回避するため、
機能設計書を作成し、排他制御・トランザクション制御等の設計を事前にチェックした。

作成理由

- 規約遵守チェック
  - 大規模開発では高スキル者の大量確保は難しく、要員のスキル幅が大きいため、機能設計書を作成し、事前チェックすることで手戻りのリスクを回避することができる

工夫点

- 排他制御の実装個所が一目で分かるよう該当行に「排他制御」と明記した

副次効果

- 運用フェーズにて他チームの実装内容が規約遵守しているか解析できた
- チーム間で、テーブル画面設計書等の他の成果物とあわせて参照することで、整合性を確保することができた

＜機能設計書イメージ＞

機能設計書

ED機能群ID	xxxxxxxxxx
ED機能ID	xxxxxxxxxx
機能名・イベント名	WBSコード変更申請情報編集

処理番号					処理名	終了/次処理	処理概要
1					編集ボタン押下		-
2					会議体意思決定WBSコード変更申請情報の削除中チェック		会議体意思決定WBSコード変更申請情報の削除中チェックを行う。 ・削除中=false
2	1				正常	3	-
2	2				エラー表示	終了	エラー表示
3					排他チェック		
3	1				会議体意思決定WBSコード変更申請情報の排他チェック		編集中、且つ、作業者が編集者か排他チェックを行う。
3	1	1			正常	4	-
3	1	2			エラー表示	終了	エラー表示
4					排他制御（ロック）		
4	1				会議体意思決定WBSコード変更申請情報へのデータ更新	5	編集中および編集者を更新し、ロックする。
5					画面遷移	-	画面遷移しない。

従来のJavaフレームワークの開発：

- 排他制御はシステム全体として制御するため、個々の設計書には記載しない

ServiceNowの開発：

- 各機能で排他制御を実装する必要があるため、設計書に「排他制御」と明記する必要がある

3. 開発プロセス系の取り組み　②フロー関連図（バッチ処理）

ServiceNowにはジョブスケジューラのような仕組みはないが、既存システムとのIF連携のため、ServiceNow上にバッチ的な処理を構築する必要が生じた。そのため、フロー定義のみでなく、フロー関連図を作成し、バッチ処理における処理の多重化、進捗状況の監視、リトライ等の制御等の設計を可視化した。

作成理由

- ServiceNowにおいては長大なバッチ処理は複数のフロー（⇒ジョブ）に分割して実装することが望ましい
  - ServiceNowはショートトランザクションを推奨
  - 処理の多重化、進捗状況の監視、リトライ等の単位でフローを分割するほうが保守性が高くなる
- 分割したフローをどのような順序・条件・関連で実行するのか、フロー関連図を用いて整合性を見る必要がある（⇒ジョブスケジューラ）

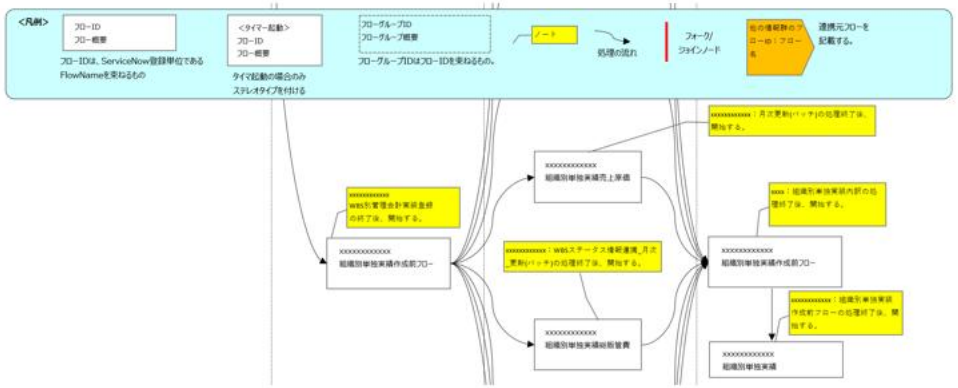
工夫点

- バッチ開始時間、多重化、監視、リトライの個所が一目で分かるように明記した

副次効果

- 性能要件が満たせなかった際の改善ポイントを絞り込むことができた

＜フロー関連図イメージ＞



3. 開発プロセス系の取り組み　③テーブル画面設計書

ServiceNowではDBトランザクション制御ができないため、画面に紐づくテーブルは原則1つの制約が課されている（例外あり）。そのため、1画面で1テーブルを扱う設計となり、テーブル項目の設計が、画面の入出力項目の設計に合致する。そこで、テーブル設計書を拡張し、画面設計情報を統合した。

作成理由

- テーブルの項目、画面の入出力項目の網羅性を確認するため

工夫点

- 各画面でどの「標準画面遷移パターン」を適用するか記載することで、画面レイアウトの記載は省略した
  - これにより、記載工数の削減に加え、UIの統一性の確保も図ることができた

副次効果

- 仕様変更が発生した際に、他チームのどのテーブルの項目をどの画面が参照しているか検索することで、影響範囲の確認ができた

<テーブル画面設計書イメージ>

4. 適用効果

達成できた点

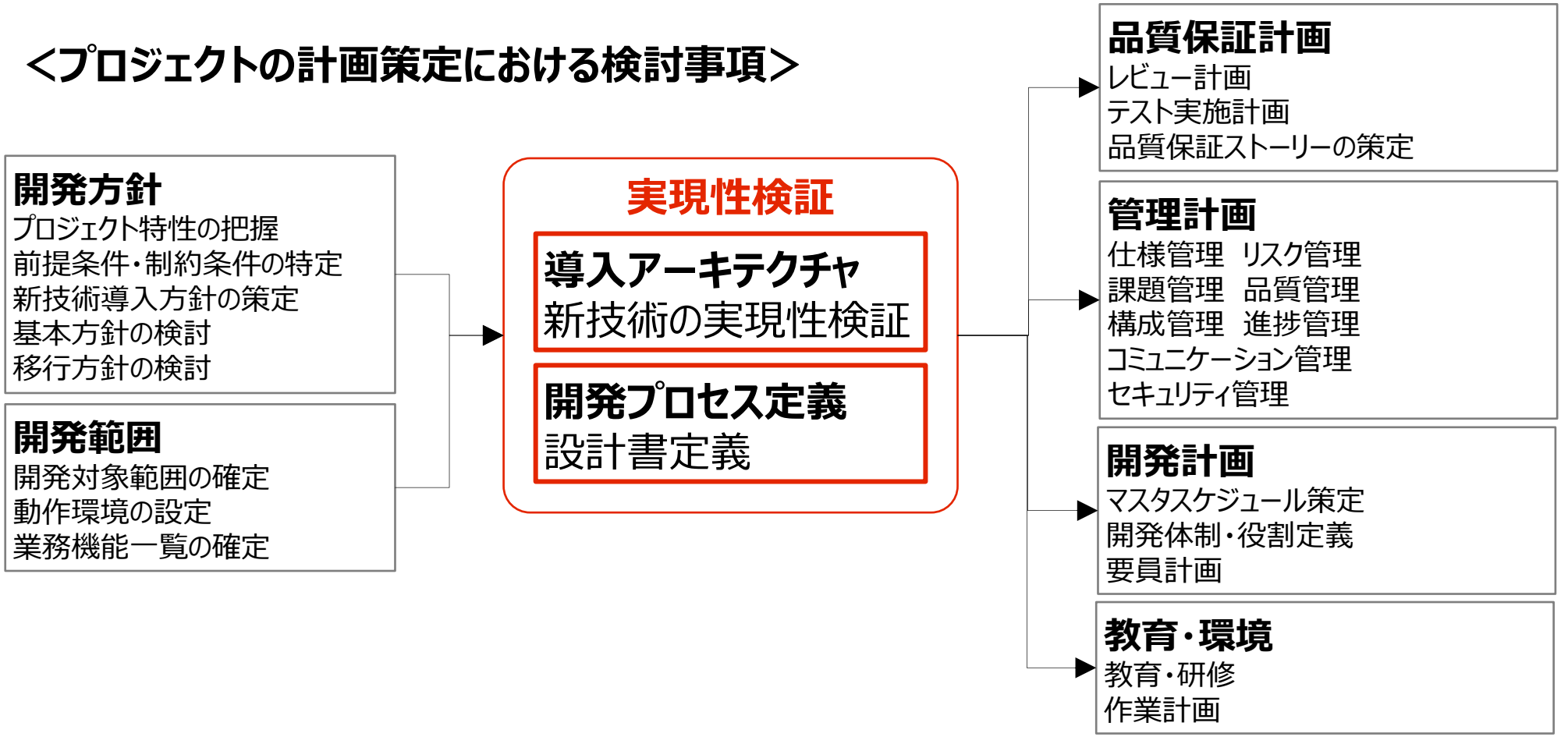
✓【製造フェーズ】大規模・高難度案件に新技術・ローコード開発を採用した結果、
ガイドライン、開発プロセスの整備といった準備・工夫をすることで、製造期間を計画通り圧縮することができた。

今後の課題

- 【要件定義・設計フェーズ】一般のServiceNow・ローコード開発では設計書は作成しないが、高難度・大規模開発では意思疎通のため設計書を作成する必要があり工数は圧縮できていない。
- 【テストフェーズ】ServiceNowにはテスト自動化の仕組み（ATF：Automated Test Framework）が提供されているが、本プロジェクトでは準備不足のため活用できず、テストフェーズの工数は圧縮できていない。
- 【今後に向けて】作成した設計書から、ディシジョンテーブル/試験項目/ATFを効率的に生成する仕組みを準備できれば、さらなる工数圧縮が期待できる。

5. まとめ

特に新技術導入といった不確定要素が大きい開発では、事前に実現性検証を行うことが重要。




NTT DATA