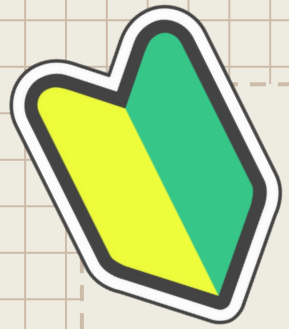
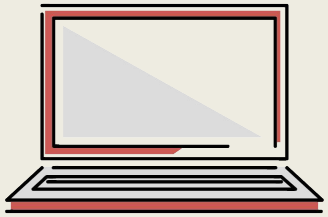


プログラム開発
未経験の方向け



ノーコード／ローコード開発

スタートガイド

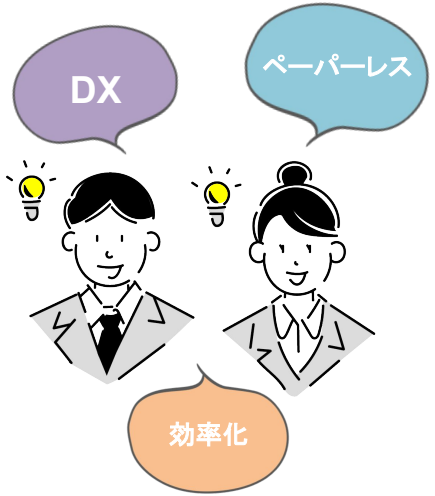
チーム・ノコロコ

TEAM NOCOLOCO

はじめに

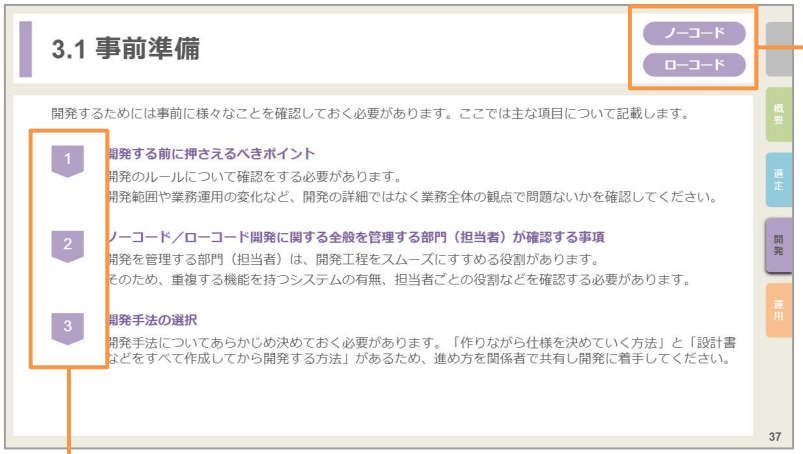
プログラミングの必要のないノーコードツール、プログラミングの必要が少し出てくるローコードツールはユーザー部門が自分たちでシステム開発を行うことができる夢のようなツールですが、事前に知っておくべきことやおさえておくべきポイントがいくつもあります。

＼ スタートガイドを読む！ ／



このスタートガイドでは、ツールの選定から開発、運用まで様々なポイントを解説しています。
また、専門的なIT知識を持たない担当者（組織）が、小規模の業務改善用のシステムを開発できるようになるよう支援することを目的としています。

ガイドラインの見方



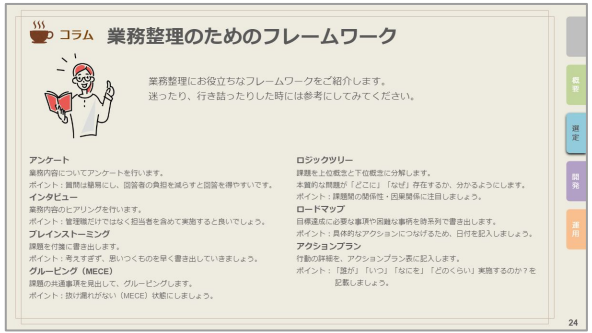
各章の詳細な説明ページの前に、内容を簡潔にまとめたページを用意しました。
気になる内容があれば、そこから読んでいただいてもOKです。

ノーコード
ローコード

ノーコードにも当てはまる、全ての方に
読んでいただきたい内容になっています。

~~ノーコード~~
ローコード

ローコード向けの少し難しい内容です。
システム開発について詳しく知りたい方は
読んでみてください。



より詳細な内容として、コラムを用意しました。
もっと詳しく知りたいという方は、ぜひ読んでみてください。

目次

1.ガイドラインの概要

1.1 ノーコード／ローコードの定義

一般的にはツールの提供会社がノーコード／ローコードツールのどちらなのかを謳っています。

ローコードツールでも、プログラミングの機能を利用することなく、ノーコードツールのような使い方で開発することができるものもあります。

ツールが「ノーコード／ローコードのどちらに属すか」ではなく、「プログラミングが発生するか否かの開発」に焦点を合わせ、本ガイドラインではノーコード／ローコードを以下のように定義します。

ノーコード
プログラミングの機能を利用しない開発のこと

ローコード
プログラミングの機能を利用した開発のこと

1.2 歴史と背景

歴史

ノーコード／ローコードの関心が高まってきたのは近年のことですが、歴史を遡ると、1979年にリリースされた「VisiCalc」というApple IIに搭載された表計算ソフトウェアが最も古いツールであるなど、最近でてきたものではありません。
また、たくさんのエンジニアやプログラマーたちが、システム開発を省略するために、プログラミングを省略しWebサイトやアプリを作る方法はないか試行錯誤し、少しずつツールが増加していきました。

背景

近年ではIT人材の不足が問題視されている中、国内企業のDX（デジタルトランスフォーメーション）推進が後押しし、人材不足が加速したため、ノーコード／ローコードを利用部門が使用する必要性が高まっています。
その結果、IT人材を持たない中小企業やこれまでITと無縁だった企業が導入するケースが増加しています。

1.3 ノーコード／ローコードの種類と導入率

主な種類

- アプリ開発
- Webサイト／ECサイト制作
- データ連携ツール
- 業務効率化（情報一元管理／業務自動化／AI開発）

国内企業での導入率（IDC Japanが実施した調査※1）

- 2020年8月の調査（回答社数435社）　導入率　8.5%
- 2021年9月の調査（回答社数485社）　導入率 37.7%（約30%増加）
- 2023年（予想）　導入率 60.0%

市場規模（ITRが実施した調査※2）

- 2019年415億円
- 2022年824億円
- 2025年1539億円（予想）

国内企業での導入率や市場規模を見ると、近年で急激に増加しているのがわかります。国内だけではなく、世界有数の調査機関Gartner（ガートナー）では、2024年までに新たに開発されるアプリケーションの65%はローコードによって開発されるようになると予想（※3）しています。

※1 国内ローコード／ノーコードプラットフォームの市場動向を発表（　IDC Japan）
※2 ITR Market View：ローコード／ノーコード開発市場 2022（ITR）
※3 Low-Code Is the Future（Gartner）

1.4 主な開発手法の特徴

従来の開発手法ではスクラッチ開発（下記参照）でシステム開発が行われてきましたが、ノーコード／ローコードツールによる開発と比べてそれぞれのメリットやデメリットが存在します。それぞれの開発手法の特徴を理解しましょう。

	ノーコード開発	ローコード開発	スクラッチ開発
特徴	プログラムコードを書かずに開発することができる	プログラムコードを書かなくても開発できるが、書く必要がある場合もある	ゼロからすべてプログラムコードを書いて開発する
開発期間	短期間	短期間～中期間	長期間
ITスキルの重要性	低	中	高
ツール提供機能への依存度	高	中～高	低
機能の豊富さ	低（※）	中（※）	高
試しやすさ	高	中	低

※一般的な特徴として表現していますが、機能の豊富なノーコード／ローコードツールも存在します。

1.5 ガイドラインの活用目的と活用方法

本ガイドラインでは、利用部門がノーコード／ローコードツールを導入するための、ツールの選定から開発、運用について解説しています。

活用目的
システムに精通していない担当者でも、ノーコード／ローコードツールを導入し、業務改善ができるようになることを目的としています。

活用方法
本書の各章を参考にしながら、導入検討を進めてください。

- 利用を始めたいが何から手をつけたらよいか？ → 『2.1 知っておきたいツールの特徴』へ
- どんなツールを選定したらよいか？ → 『2.2 ツール選定の流れ』へ
- 開発に着手する前に必要なことはあるのか？ → 『3.1 事前準備』へ
- 作りっぱなしを防止したい！ → 『4.1 管理方法』へ

1.6 必要なスキル

ノーコード／ローコードツールによる開発を行う上で必要となるスキルについて記載します。

ノーコード
ローコード

- **業務分析やロジック設計といったビジネスに即したスキル**

- **パソコンの基本操作**

  キーボードによる文字入力や、インターネットに接続し、ブラウザ操作など、日常使用できることが前提です。
  【例】SNSサービスの操作／動画配信サイトでの視聴／通販サイトでの購入など

- **Excel、Word、PowerPointの知識**

  ExcelやWord、PowerPointのような操作感のツールも多いです。これらの操作経験は役に立ちます。

- **プログラミングに関する知識**

  ローコード開発では、プログラミング（コーディングなど）を行う場合があります。
  ツールにより使用する言語は異なります。

- **データベースに関する知識**

  データベースを使用するツールが多いです。適切な設定を行うための知識が必要です。

コラム

基礎的なアルゴリズム

ノーコード／ローコードツールに関わらず、
システム化したい業務を「処理」として流れを考える必要があります。
その際に「条件分岐」「繰り返し」を用いて処理の流れを整理します。

【条件分岐】

Aさん
10時までに総務部からのデータが欲しい

10時

データが来なかった

総務部に催促する

データが来た

データ入力作業をする

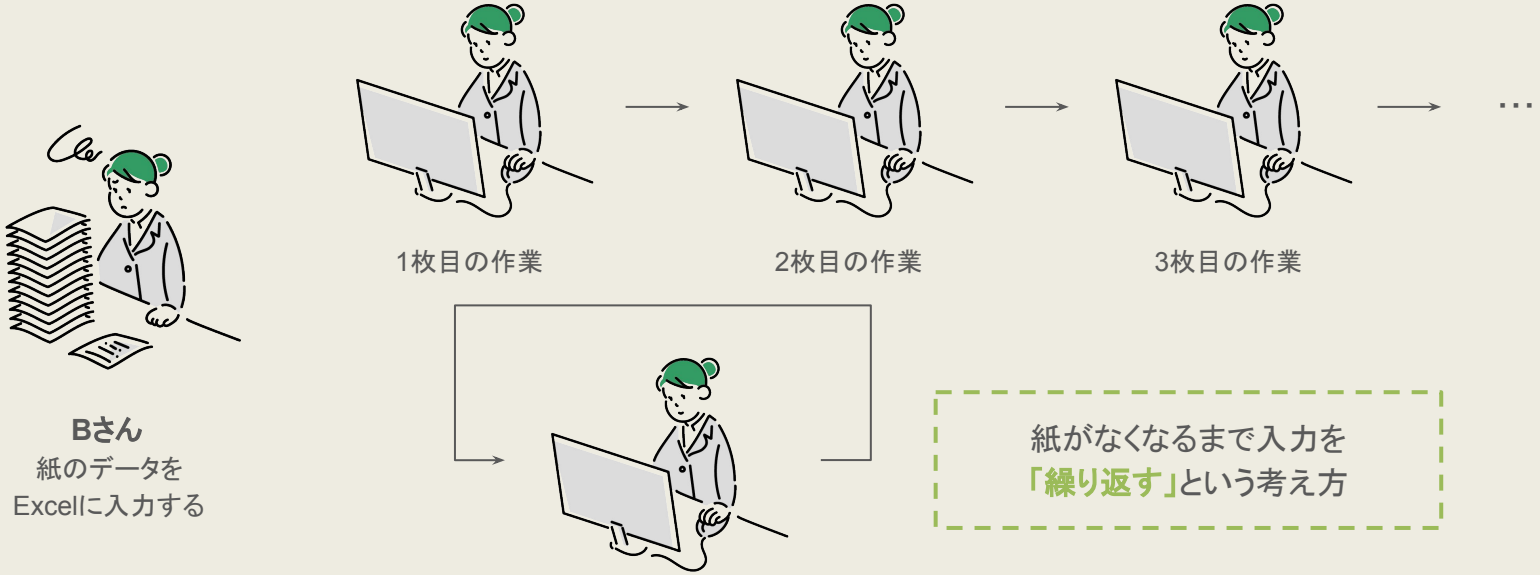
「10時にデータが来ていたか」
という**「条件」**で
Aさんのその後の行動が
「分岐」する

コラム

基礎的なアルゴリズム

【繰り返し】

「繰り返し」というものはIT知識を持たない人にもなじみがあるものです。
業務フローや処理の流れを考える際、下記の図における「1枚目の作業、2枚目の作業…」と考えると紙が増えるほど大変です。作業するタイミングによっても枚数が違うかもしれません。
そこで、「紙がなくなるまで繰り返し入力作業を行う」という整理の仕方をします。



2.選定

2 選定

ノーコード／ローコードツールは、様々な領域で展開されています。さらにその領域ごとに汎用性のあるものから、特定の用途に特化したものまで、多種多様なツールが展開されています。

この章では、次の流れで説明をしていきます。

1 **知っておきたいツールの特徴**

ツールを選定する前に、ノーコード／ローコードツールにはそれぞれどのような特徴があるのかを説明します。

2 **ツール選定の流れ**

業務整理～ツール選定の流れを説明します。
後半には、事例へのリンクを含めた選定フローがありますので、参考にしてみてください。

3 **ツールの導入体制**

ノーコード／ローコードツールのメリットの一つである「スピード感」を持った開発を行うための体制とは？
ツールの乱立を防ぎ、利用状況をしっかりと把握するためにはどうすればいいのか？
について説明します。

2.1 知っておきたいツールの特徴

ノーコード／ローコードツールの以下の項目について、特徴を説明します。
ツールの特徴を理解し、自社で利用するならどのようなツールが向いているかを検討しましょう。

1. ツールの利用形態
2. 対応デバイス
3. 料金プラン
4. サポート
5. テンプレート／外部連携

2.1.1 ツールの利用形態

ツールの利用形態は、インストール型とSaaS型の二つに大別できます。
それぞれの特徴を理解しておきましょう。

分類	インストール型	SaaS型
特徴	● インストール先が必要 ● 解約時にアンインストールが必要	● ブラウザのみで始められる ● 気軽に試すことができる
初期コスト	● インストール先の環境準備が必要 その分のコストが必要	● 基本的にはインターネット接続環境があれば その他のコストは不要
バージョンアップ	● 自分で行う必要がある ● 操作性や見た目が突然変わることはない	● 自動的に行われる 操作性や見た目が突然変わることがある
セキュリティ	● 自分で設定を行う必要がある ● 高度な設定を行うことができる	● ツールに依存するため、自分で設定はできない
解約	● 手続きとアンインストールが必要	● 手続きのみ

2.1.2 対応デバイス

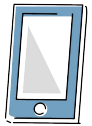
ツールによって、開発した機能が使用可能なデバイスが異なります。
対応しているデバイスを確認した上で、ツールを選択するようにしましょう。
ここでは、主なデバイスと想定される利用シーンを記載します。

パソコン

- 自席での利用がメインの場合
- 利用者が操作することなく、時間起動など自動で処理が実行される場合

タブレット

- 持ち運びながら利用したい場合
- 入力／確認項目が比較的多い場合



スマートフォン

- 持ち運びながら利用したい場合
- 入力／確認項目が比較的少ない場合

2.1.3 料金プラン

料金プランには、「**ライセンスの買い切り**」と「**サブスクリプション（月額／年額）**」があります。
拡張機能は別途費用が掛かる場合も多いです。想定外の費用が発生しないよう、詳細を確認する必要があります。

料金プランの考え方はツールによって異なりますが、以下の要素が代表的です。

- 開発／本番利用
- 利用ユーザー数
- 利用データ容量
- 問い合わせ可否
- 各種機能拡張

無料プランや、無料トライアルの期間が提供されているツールも多くあります。
本格的な導入をする前に、無料プランを試したり、無料トライアル期間に利用したりするといいでしょう。
これにより、目的にあったツールを選びやすくなります。
将来的に有償プランの利用を検討していたとしても、まずは無料で利用できる機能を十分にチェックしておきましょう。

2.1.4 サポート

ツールのサポートやマニュアルについても事前に確認しておきましょう。
充実したサポート体制は、ツールを効率よく利用していく上では重要なポイントとなります。
【主な確認ポイント】

ポイント	確認事項
サポート契約の有無	製品とサポートのライセンスが独立している場合があります。 利用回数に制限がある場合もあります。契約内容をしっかりと確認しましょう。
サポート内容	以下の内容を確認し、自社の運用に合った内容かを確認しましょう。 サポート利用可能者　：サポート利用者が限定される場合があります。 問合せ方法　：電話、メール、専用サイトからの問合せなどがあります。 受付時間　：平日のみかつ時間指定、年中無休など様々なパターンがあります。
マニュアルの有無	分かりやすいマニュアルが提供されていることは、ツールを活用する上で重要なポイントとなります。 マニュアルの他にも、書籍やサポートサイトが充実しているかなどを確認するといいでしょう。
対応言語	海外のツールを利用する場合は、日本語での対応が不可の場合もあります。 海外のツールを利用する場合は、サポートやマニュアルが英語対応のみの場合もあるため、 注意しましょう。

2.1.5 テンプレート／外部連携

テンプレート

一般的な業務のテンプレートが提供されているツールがあります。
テンプレートを利用することで、簡単な設定作業だけでアプリケーションを作成することができます。

外部連携

システムの開発では、新しく機能を作るだけではなく既存のシステムを利用することも視野に入れると、生産性が向上します。
そのため、ツールが既存システムと連携可能かを確認するといいでしょう。

以下のシステムは連携が必要になることが多いため、連携可否をあらかじめ調査しておくと良いでしょう。

- 顧客管理
- 決済
- プロジェクト管理
- SNS
- カレンダー

2.2 ツール選定の流れ

様々なツールが存在する中で、業務課題の解決が可能なツールを選択するのは非常に大変です。
ここでは、ツールを選定するまでの流れについて説明します

1

業務課題の整理

ツールのことは一旦脇に置いて、まずは現状の業務課題を把握することから始めることを推奨します。

2

ツール適用範囲の検討

解決したい課題が何かを把握したところで、課題を解決するために業務のどの部分にツールを適用すればいいのかを検討します。

3

ツールの選定

ツールを利用したい業務を絞り込んだところで、ツールの選定に入ります。

選定フロー

導入事例へのリンクを含む選定フローを用意しました。
ツール適用範囲の検討、ツールの選定の参考にしてみてください。

2.2.1 業務課題の整理(1)

1

現状の実施業務の棚卸し

まずは現状の業務内容を整理しましょう。
各業務の担当者へのアンケートやインタビューなどを通して、業務内容のヒアリングを行います。

棚卸し方法

Step1　タスクリストの作成

大まかな「行動ベース」のタスクリストを作成します。
業務の流れに沿って、どのような作業を行っているかを整理しましょう。

Step2　作業の細分化

作成したタスクリストの作業を細分化します。
細分化することで、より具体的に業務内容を把握することができます。

Step3　業務量の調査

それぞれの業務内容に対して、業務量を調査します。

- 業務にかかる時間(1回あたりの時間)
- 業務の発生頻度
- 業務に必要なスキルなど

ポイント

以下を明確にしましょう。

- いつ／どこで／誰が／何を／どのように
- 業務の順番
- インプットとアウトプット
  使用データ、ファイル、紙帳票の有無など
- 各作業で使用している業務システム

参考：業務改善につながる「業務の棚卸し」とは？実践方法もご紹介 | 業務改善ノート　| 株式会社TMJ

2.2.1 業務課題の整理(2)

2

業務の見直し

可能であれば、ツールの適用範囲を検討する前に、以下のような観点で業務の要／不要を含めた見直しを実施することを推奨します。ツール導入の効果をより大きなものにすることができます。

本当に必要な業務か？

不要な業務は廃止することも検討しましょう。
作業全体、部分的な工程、帳票の項目など、作業の目的と照らし合わせ必要なものだけ残します。

共通化はできるか？

類似している業務は、違いを整理し可能な限り共通化しましょう。
ただし、無理に共通化を行う必要はありません。

手順の見直し

順番を入れ替えることで業務を効率化できる場合もあります。

3

業務改善に向けた検討

業務量調査の分析を通して、現状業務における課題を明確にし、業務改善に向けた優先順位や方向性を定めます。

コラム

業務整理のためのフレームワーク

業務整理にお役立ちなフレームワークをご紹介します。
迷ったり、行き詰ったりした時には参考にしてみてください。

アンケート
業務内容についてアンケートを行います。
ポイント：質問は簡易にし、回答者の負担を減らすと回答を得やすいです。

インタビュー
業務内容のヒアリングを行います。
ポイント：管理職だけではなく担当者を含めて実施すると良いでしょう。

ブレインストーミング
課題を付箋に書き出します。
ポイント：考えすぎず、思いつくものを早く書き出していきましょう。

グルーピング（MECE）
課題の共通事項を見出して、グルーピングします。
ポイント：抜け漏れがない（MECE）状態にしましょう。

ロジックツリー
課題を上位概念と下位概念に分解します。
本質的な問題が「どこに」「なぜ」存在するか、分かるようにします。
ポイント：課題間の関係性・因果関係に注目しましょう。

ロードマップ
目標達成に必要な事項や困難な事柄を時系列で書き出します。
ポイント：具体的なアクションにつなげるため、日付を記入しましょう。

アクションプラン
行動の詳細を、アクションプラン表に記入します。
ポイント：「誰が」「いつ」「なにを」「どのくらい」実施するのか？
を 記載しましょう。

2.2.2 ツール適用範囲の検討

業務課題を解決するために、業務のどの部分にツールを適用するかを検討します。
ただし、ツールにより対応可能なことが異なるため、ガチガチに適用範囲を決めることは推奨しません。
判断基準例を以下に記載します。

向いている業務

- 定型業務である
- 単純作業だが、時間が掛かっている
- 紙やExcelで運用している
- 難易度が低いが、業務改善効果が得やすい
- 複数システムへの重複入力が発生している
- 本社と現場など遠隔地での情報共有に課題がある
- システム間のデータ連携

向いていない業務

- 実現したい内容が複雑
- 求められる機能が多い
- 堅牢なセキュリティが求められる
- 最先端の技術や特殊な機能が求められる

2.2.3 ツールの選定

ツールを適用する範囲を決めたら、要件に合うツールの選定を行います。

1

要件と合う領域のツールに絞る

【領域の一部例】

ビジネスアプリ開発ツール

仕事に役立つアプリを作成することができるツール。
営業管理／生産管理／請求書管理など

モバイルアプリ開発ツール

モバイルアプリ作成に特化したツール。
製造現場や工場、店舗など現場業務向け。
在庫管理／作業日報など

業務自動化ツール

人の行っていた作業を機械化するツール。
定型業務の自動化／データ連携など

ECサイト構築

ECサイトの構築から、決済や売上管理まで、
一通りできるツール。

2.2.3 ツールの選定

2

トライアル利用をしてみる

無料プランや、無料トライアル期間が設けられているツールが多くあります。
本格的な導入をする前に、目的にあった使い方ができるか試してみるのもおすすめです。

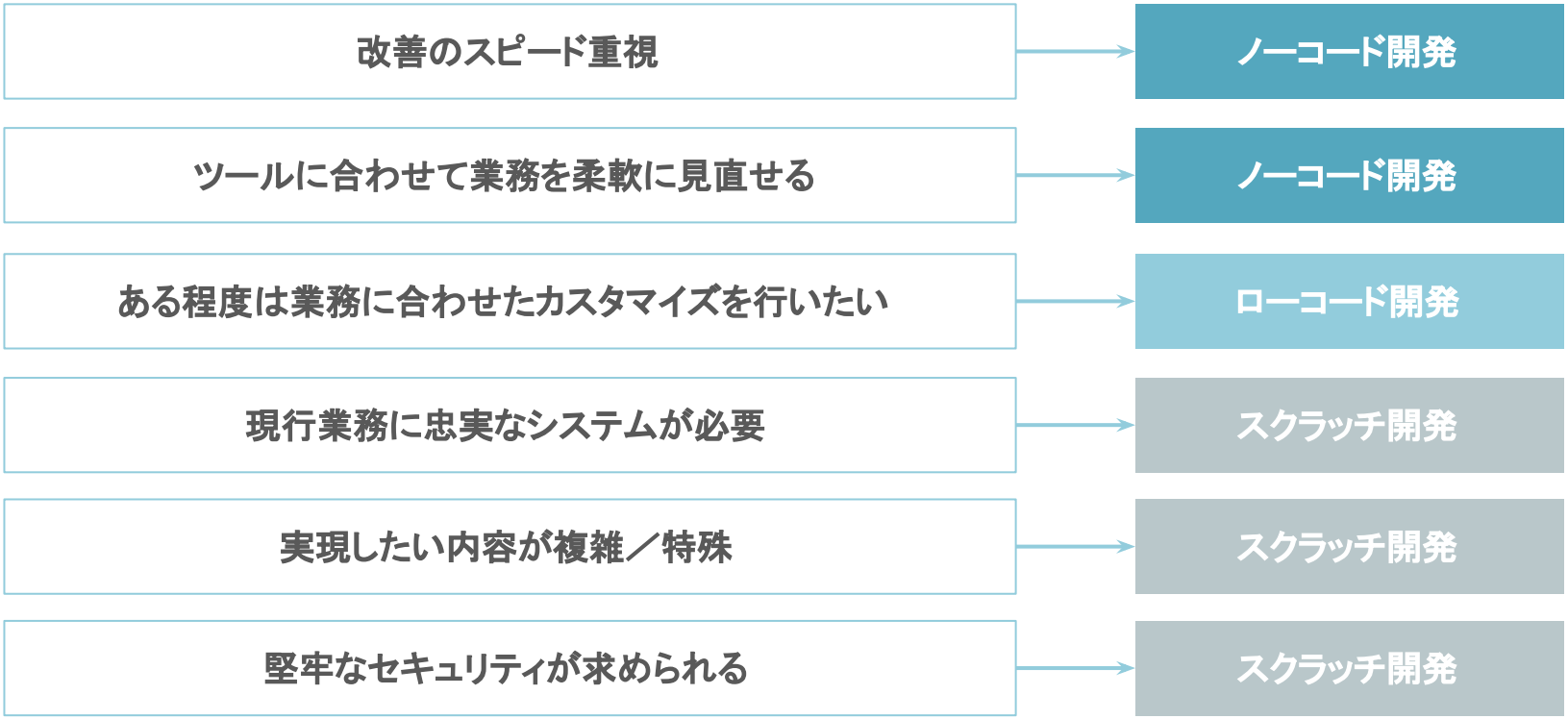
3

業務の見直しを検討／他のツールを検討

最初から業務にぴったりと合うツールに巡り合うことは難しいです。
合わないからとツール導入自体を諦めるのではなく、次の点を検討してみるといいでしょう。

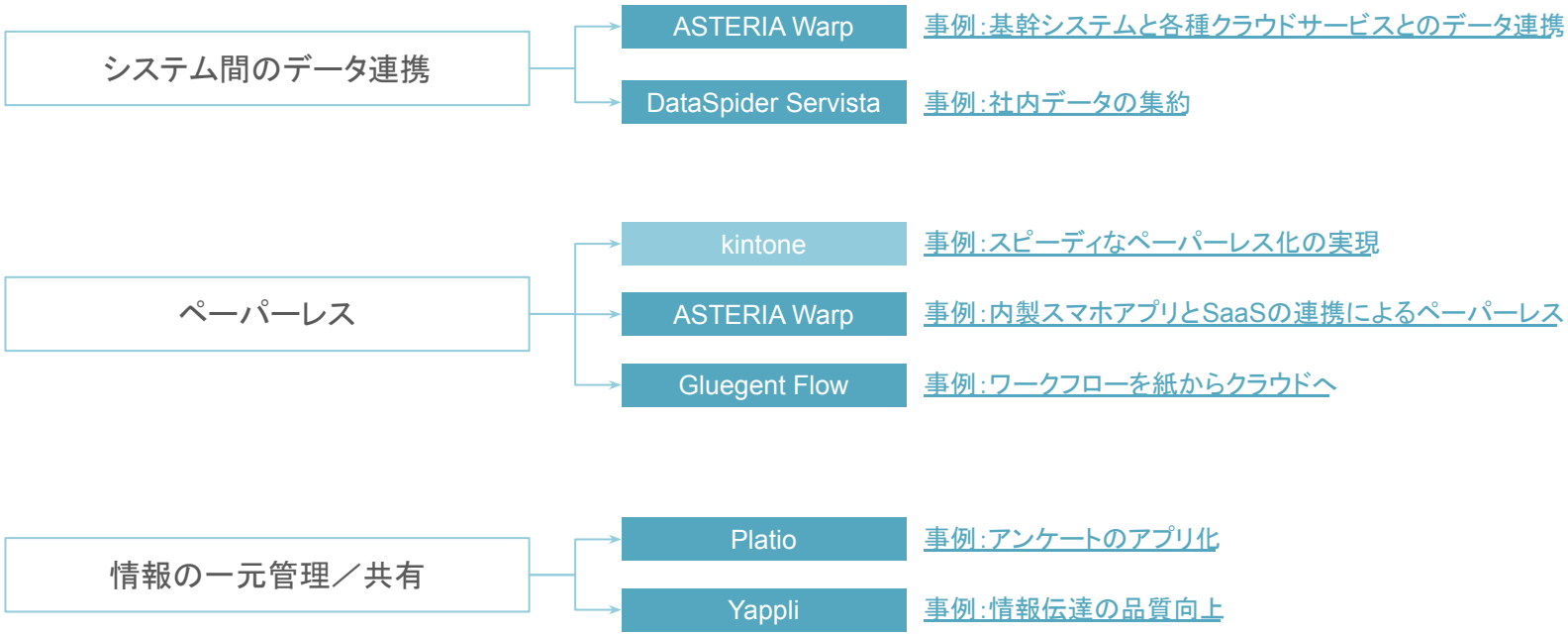
- ツールに合わせて、業務を見直すことができるか検討
- 合わなかった点を選定条件に含め、他のツールを検討

2.2.3 ツール選定フロー

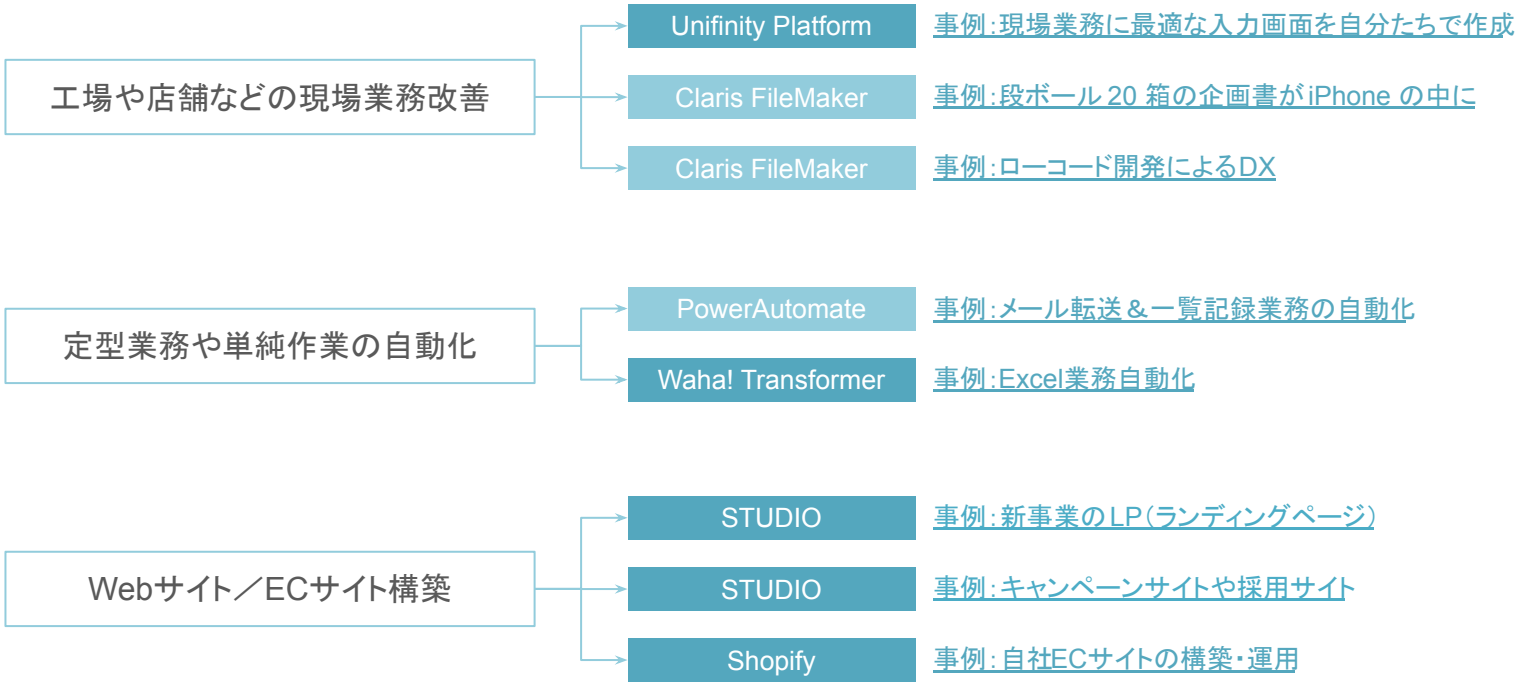


2.2.3 ツール選定フロー（事例）

改善したい内容別の導入事例を紹介します。業務選定～ツール選定の参考にしてみてください。



2.2.3 ツール選定フロー（事例）



コラム

まだまだ可能性は未知数…！

ここまでの選定フローに出てきたもの以外にもノーコード／ローコードツールは
たくさん存在します。その中でも海外製のノーコードツールをいくつかご紹介します。

これらのツールはSaaS型で提供されており、無償プランが存在します。
ノーコードツールでつくられたとは思えないほど本格的なアプリばかりです。
ノーコードツール＝簡単なものしか出来ない、とは言い切れないですね。

- Glide

  事例：農業現場をデジタル化しよう！農家の自作アプリ・活用サービスの紹介　| Qiita
- Adalo

  事例：SPOTTO | 新卒採用のマッチングアプリ
- Bubble

  事例：ノーコード Bubbleの開発事例 - TripBook

参考書籍：『基礎から学ぶ ノーコード開発』C&R研究所

2.3 ツールの導入体制

ツールを導入するにあたって、どのような体制を組めばいいのかについて説明します。

1

導入体制の検討

ツール利用者が自ら開発を行うのか、システム部門が存在する場合はシステム部門が開発を行うのかなど、
開発の主体者を検討しましょう。
外部に支援を依頼する場合は、外部とやり取りを行う窓口となる担当者を決める必要があります。

2

外部支援の検討

自分たちだけで開発を行うことが不安な場合は、外部からの支援を受けることを検討しましょう。
外部支援にも様々なパターンがあります。

3

役割分担

設計／開発／保守など、各工程を誰が主体となって担当するのか、想定しておきましょう。

2.3.1 導入体制の検討

自社の組織機構と開発するシステムの規模や難易度から、どんな体制を組むことができるのかを想定してください。
利用部門の担当者や管理職をはじめ、システム部門（存在している場合）や外部支援も含めて、システム開発に必要な体制をあらかじめ想定することで、ツール選定の際の不安が削減できます。

利用部門が開発の主体となる場合

メリット　：業務を熟知しているため要望の共有がスムーズに行えます。
デメリット　：開発スキル不足により期間が長期化したり、システム設計が煩雑となる可能性があります。

システム部門が開発の主体となる場合

システム部門が存在する場合は、開発を依頼してもいいでしょう。

メリット　：システムの知見が豊富なため、短い工数での開発が可能となります。利用部門が気づかない提案を得られることもあります。
デメリット　：業務知識が乏しい場合があり、要望に対してのずれが生じる可能性があります。

2.3.2 外部からの支援

社内の体制だけで開発を行うことに不安がある場合は、外部からの支援を検討しましょう。
支援パターンの例を以下に記載します。

- **開発全般を依頼する**
- **サポート契約を結び、開発支援を受ける**
- **技術トレーニングを受ける**

外部からの支援を受ける場合は、将来的な内製化を目指し、
「自社にノウハウを吸収するための支援依頼」という認識を持つことが大切です。
内製化することで、ノーコード／ローコードツールのメリットの一つである「スピード感」を持った開発を行うことができるようになります。
また、開発から保守までを内製化することで、コストを抑えながらも様々な業務を効率化することができます。

2.3.3 役割の分担

ツールを導入するにあたっては、設計／開発／保守など、主に誰が担うのか体制を想定しておきましょう。

設計／開発

ツール適用範囲に選定した業務内容とツールが提供する機能を照らし合わせ、設計や開発を行います。
次のパターンが考えられます。

- 利用部門が自身で行う
- 社内にシステム部門が存在しており、システム部門が主体で行う
- 外部委託し、外部の会社が主体で行う

業務を最も理解しているのは利用部門です。どのパターンであっても、設計には関わっていきましょう。

保守

ツール利用者からの問合せを受ける窓口です。
また、ツールの提供元への問合せも、保守担当が行うようにするといいでしょう。
→ 詳細は、『4.1.2 保守体制』へ

管理

社内におけるツールの使用状況を管理します。
管理者の把握していないツールを自由に使用することは重大なセキュリティリスクに繋がります。
システム部門が存在する場合は、システム部門が管理をする場合が多いです。

コラム

管理者への相談／申請は忘れずに

新たなツールの導入を検討する場合は、社内の管理者に相談／申請しましょう。
適切な管理のもとで運用することが重要です。
主な理由は以下の通りです。

- 社内で利用しているツールを一元管理するため
- ツールの乱立を防ぐため
- シャドーIT（※）を防ぐため

適切に管理することは、以下に繋がります。

- ウィルス感染や情報漏洩などのセキュリティ上の重大な問題が発生するのを防ぐ
- 重大な問題が発生してしまった場合、影響範囲の特定が困難になるのを防ぐ

※ シャドーITとは
会社や組織が使用を許可していない、または従業員が利用していることを会社や組織が把握できていない
デバイスや外部サービスのこと

開発

3.1 事前準備

ノーコード
ローコード

開発するためには事前に様々なことを確認しておく必要があります。ここでは主な項目について記載します。

1

開発する前に押さえるべきポイント

開発のルールについて確認をする必要があります。
開発範囲や業務運用の変化など、開発の詳細ではなく業務全体の観点で問題ないかを確認してください。

2

ノーコード／ローコード開発に関する全般を管理する部門（担当者）が確認する事項

開発を管理する部門（担当者）は、開発工程をスムーズに進める役割があります。
そのため、重複する機能を持つシステムの有無、担当者ごとの役割などを確認する必要があります。

3

開発手法の選択

開発手法についてあらかじめ決めておく必要があります。
「作りながら仕様を決めていく方法」と「設計書などをすべて作成してから開発する方法」があるため、
進め方を関係者で共有し開発に着手してください。

3.1.1 開発する前に押さえるべきポイント

ノーコード
ローコード

開発するにあたり、以下のような観点を押さえる必要があります。

1

会社や部門内でのルールを確認する

すでに導入しているツールの場合、開発ルールが設定されていることがあります。
要件定義を行う段階で確認いただく必要があります。

2

作り込み過ぎが起きないように確認する

他部門への展開や、異動などで運用管理担当者が変わることを想定し、最初から複雑に開発するのは避けましょう。作成する機能がブラックボックス化し、メンテナンスできなくなる恐れがあります。

3

システム外の業務運用の変化を確認する

既存の業務フローと、新たな業務フローを照らし合わせ、ツールの導入後、現在より負荷が高くならないことを確認してください。作成する機能が継続して使われなくなる恐れがあります。

3.1.2 管理者が確認する事項

ノーコード
ローコード

管理者は、ノーコード／ローコード開発に関する全般に対して、以下のような観点で確認をしてください。

1

重複の開発が起きないかの確認

仕様検討の概要や目的から、同様の開発が行われているかを確認してください。
同様の機能が既に開発されていた場合は、開発コストを抑えられる可能性があります。

2

問い合わせ窓口の確認

利用部門または担当者の問い合わせ窓口を確認してください。
複数の担当者から要望を伝えられると、開発の手戻りなどが起こる恐れがあります。

3.1.3 開発方法

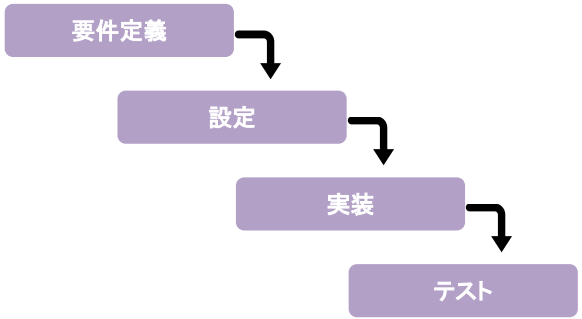
ノーコード

ローコード

開発方法には「アジャイル開発」と「ウォーターフォール開発」があります。
それぞれの特性を理解し、開発したい内容に合った手法で開発を進めてください。

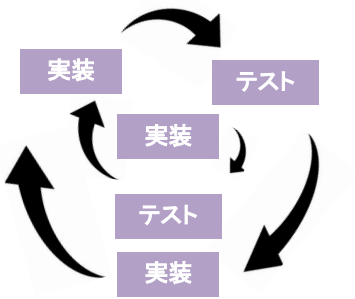
ウォーターフォール開発

- 最初に全体の機能設計／計画を決定し、
  この計画に従って開発／実装していく手法
- 開発スケジュールを設定するため、
  現状の進捗度を把握することが可能
- 問題が発生した場合、戻る工数が大きくなる



アジャイル開発

- 小さな単位で実装とテストを繰り返し開発する手法
- 開発の途中で仕様の変更や追加が予想される
  プロジェクトに向いている手法
- 問題が発生した場合、戻る工数が少ない反面、
  開発の方向性がぶれやすい



3.2 開発

ノーコード

ローコード

開発では以下の作業が必要となります。

1. **要件定義**
   必要な機能を列挙します。

2. **設計概要**
   各種開発手法別に設計書を用意します。

3. **命名ルール**
   開発の際の様々な命名ルールをあらかじめ作っておきましょう。

4. **データベースの設定**
   データを保存しておく場所の設定をします。

5. **テスト**
   作成した機能が正常に動作するか確認します。

3.2.1 要件定義

ノーコード
ローコード

『2.2 ツール選定の流れ』にあるように、現行業務フローの列挙や見直しを行った上で、要件定義を行いましょう。

機能の列挙

まずは必要となる機能を検討します。

【機能例】

- データの一覧表示
- データの検索／入力フォーム
- データの出力
- メールの送信

画面が必要となる場合は、必要な画面単位で検討すると分かりやすいです。
画面が不要な場合は、データを取得する、○○を△△に加工する、処理の完了をメールで通知する、など処理の工程で考えると比較的列挙しやすいです。

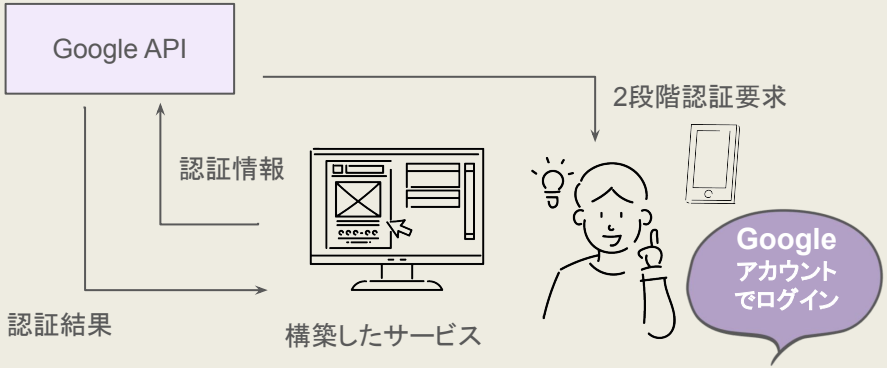
コラム

あと少しだけ機能が足りない！そんな時は…

業務に基づいた適切なノーコード／ローコードツールを選択しても、
「○○の機能だけ足りない！」という場面に遭遇することがあるかもしれません。

例えば、ユーザー登録をして利用するサービスを構築することになった際に
「2段階認証をしてログインさせたいのに機能がない！」となったとしましょう。

その際にAPI（※）を利用する、という方法で実現できることがあります。
方法の一つとしてGoogleのAPIを利用することで、Googleアカウントを使って
構築したサービスの「2段階認証をしてログインする」機能を実現することができます。



※APIとは「Application Programming Interface」の略で
アプリケーション同士やサービス同士を繋ぐためのものを指し、
自ら開発することなく様々なデータの取得や、処理の実行が可能です。

APIの利用する場合、提供元との契約の必要の有無、
有償／無償提供など様々あります。必ず確認しましょう。

3.2.2 設計概要（ウォーターフォール開発）

ウォーターフォール開発による設計は、以下の流れで行います。

1 **基本設計（外部設計）**

- 各種利用するツールの決定（ノーコード／ローコードツール、データベースなど）
- ユーザーインターフェースの設計（操作画面の設計）
- モックアップ（※）を作成し、要件定義などと相違ないか確認

※ 実際の処理は実装されていない、画面の遷移や操作性のイメージをしやすくするためのアプリケーション

2 **詳細設計（内部設計）**

- データベース設計
- 入出力表の作成（おもに画面単位での入力値と出力値の一覧を作成）
- テスト設計書

ウォーターフォール開発は、前の工程には戻らないことを原則とする手法です。
各工程ごとに承認を受けてから、次の工程に進みます。

3.2.3 設計概要（アジャイル開発）

ノーコード
ローコード

アジャイル開発による設計は、以下のように行います。

アジャイル開発では、ソフトウェア作成が優先と考えられているため、必ずしも事前に設計書を用意する必要はありません。
しかし、要件定義との乖離やチーム内での認識の不一致を防ぐために、いずれかの（あるいはそれらを組み合わせた）方法で設計書に該当するものを用意することが望ましいです。

- タスク管理ツール（※）を用いて、チケットにメモを残す
  ※タスク管理ツール… Backlog、Redmineなど
- ウォーターフォール開発で作成するような設計書を用意し、随時更新する

アジャイルソフトウェア開発宣言 （agilemanifesto.org）

3.2.3 命名ルール

ノーコード
ローコード

命名ルールとは、開発を行うにあたり、開発者が名づける識別名（※）についてのルールです。
プログラミング言語や開発する環境などにより定められているものと、開発を行う組織で内部的に定めるものがあります。
※ クラス名、メソッド名、関数名、定数名、変数名など

ルールが必要な理由

- ソースコードの可読性が上がる
  名前から、意味や使い方などが、ある程度推測できるようになります。

- 修正や追加開発の効率が上がる
  検索や置換などが行いやすくなります。
  また、意図せずに同じ名前を別の意味で使用してしまうことによるバグを防ぐ効果もあります。

3.2.3 命名ルール（プログラミング全般）

プログラミングにおいてよく使われる命名ルール

命名ルールに沿った名前を付けることで、役割の判別や区別が容易になります。

例	記載ルール	役割	名称
USER_NAME	英大文字で単語間をアンダーバーでつなぐ	定数	アッパースネークケース
UserName	単語の頭文字が大文字	クラス	パスカルケース
userName	最初の単語以外、単語の頭文字が大文字	メソッド	キャメルケース
user_name	英小文字で単語間をアンダーバーでつなぐ	関数	スネークケース
user-name	英小文字で単語間をハイフンでつなぐ	HTML属性名	ケバブケース

3.2.3 命名ルール（変数）

変数の命名ルール

変数の命名ルールは、比較的自由度が高いです。

「キャメルケース」、「スネークケース」、「ケバブケース」のいずれかが使用されることが多いです。

しかし、ハイフンでつなぐ「ケバブケース」は、プログラミング言語やサービスによっては非推奨な場合もあるため、ハイフンのかわりにアンダーバーを使用する「スネークケース」の使用をおすすめします。

また、ローマ字を英単語に変換するか、日本語をローマ字表記にするか、可読性の観点からルールを決めておく必要があります。

【例】

項目名「利用者名」

スネークケース（英小文字をアンダーバーでつなぐ）で記載します

日本語をローマ字表記	riyousha_mei
英単語	user_name

3.2.3 命名ルール（その他）

ノーコード

ローコード

英単語を使用する際の省略形

英単語が長い場合、プログラマーの間でよく使われる省略形が存在します。それらを使うか否か規定します。
プログラミングに不慣れな場合は、省略せずにそのまま使用することをおすすめします。

【例】

省略前	省略後
object	obj
temporary	tmp
count	cnt

3.2.4 データベースの設定（1）

ノーコード
ローコード

データベースとは、システムにおいてデータを貯めておくためのツールです。
ツールによって、決められたデータベースしか使用できないもの、データベースの選択が必要なものがあります。

本スタートガイドでは「リレーショナルデータベース（RDB）」を前提に説明を進めていきます。

リレーショナルデータベース（RDB）とは

Excelの表のような形をとったデータベースで、行と列で構成されます。
一つの表をテーブルと言います。また、複数のテーブル同士の関係を
定義することで、複雑なデータの組み合わせも扱うことができます。
高度な検索を行うことができるのが特徴です。
一方で、データを扱う場合は、件数が多いほど処理に時間がかかるため、
Webのアクセスログなどの大量にあるデータの扱いには向きません。

【代表的なRDB】
PostgreSQL, MySQL, SQL Server, OracleDBなど

3.2.4 データベースの設定（2）

データベースの項目の検討方法

データベースに貯めるデータとして検討するのは、「貯めた後にデータを取り出して、利用したい」値です。
ノーコード／ローコードツールで作成したシステムに対して、
「利用者が入力するデータ」や「処理を実行するときに取得するデータ」などを対象とします。

【例】通販サイトのユーザー会員情報（「会員ID、名前、連絡先、住所」など）

これらはユーザーが通販サイトで初めて買い物をする際に「入力するデータ」であり、購入した商品を届けるために
「取り出して利用されるデータ」でもあります。
また、通販サイトで「1か月ごとに会員がどれくらい増えたか（減ったか）を集計する」という機能があった場合、件数を数えるため、
会員情報は「（処理の）実行時に取得されるデータ」となります。

3.2.4 データベースの設定（3）

ノーコード
ローコード

データベースの設定は、後から変更するのが難しい（もしくは不可能な場合も）です。
また、テーブル構成によって、処理効率に大きな影響を与える場合もあります。
そのため、設計段階で慎重に検討する必要があります。

テーブル構成

1

正規化

一つの要素から他の要素が決定される場合、別のテーブルに分ける必要があります。
これを正規化といいます。第3正規形、またはあえて正規化しない非正規形にすることが多いです。
非正規形に向いているのは、Webのアクセスログなどのログ系に多いです。
→ 詳しくは『コラム「正規化」とは』へ

2

複数テーブルでのデータの紐づけ（外部キーの活用）

外部キーとして別テーブルの値を利用すると、データ探査の処理効率が上がったり、データが存在しない場合にエラーとして検知することができます。

3.2.4 データベースの設定（4）

ノーコード
ローコード

概要
選定
開発
運用

型／サイズ

データベースはどんなデータが入るかを定義する「型」と、その列に入るデータを考慮した「サイズ」を決定する必要があります。利用できる「型」は、使用するデータベースによって名称や仕様が異なります。
使用するデータベースの仕様を確認するようにしてください。

【例】

型

型名	内容
integer	整数
text, char	文字列
timestamp	タイムスタンプ （日時を扱うための型）

サイズ（種類はデータベースの型による）

型名	内容
固定長	格納されるデータが常に同じ長さ
最大長	格納されるデータが設定した最大長までの長さで可変
可変長	PostgreSQLのtextのみ 格納されるデータが無制限の長さで可変

3.2.4 データベースの設定（5）

列名

変数と同様、列名も命名ルールを決める必要があります。
データベースの種類によっては、推奨ルールが存在するものもあります。
日本語入力できるものもありますが、バグなどの発生を考慮し、英数字で定義するのが一般的です。

命名ルールについては、『3.2.3 命名ルール』を参照してください。
英数小文字をアンダーバーでつなぐ「スネークケース」、または英数小文字をハイフンでつなぐ「ケバブケース」を使うことが多いです。

コラム 「正規化」とは(1)

正規形の例

ある通販サイトで「購入者1人が一度に1種類の品しか購入できない」という仕組みがあった場合、ある注文は以下のような情報になります。ある一つの要素から別の要素が確定する時(「りんご」は「300円」である)、情報を別テーブルに分散し、別のテーブル(注文テーブル)で使用する時は一意のID(「100」)で呼び出します。

【非正規化状態】

個数	品名	金額	購入者(会員名)	電話番号
1	りんご	300	A	090-XXXX-XXXX

りんごから300円は確定できる

購入者Aから電話番号は確定できる

【正規化状態】

注文テーブル

No	商品番号(外部キー)	個数	購入者番号(外部キー)
1	100	1	1000

商品テーブル

No	商品名	金額
100	りんご	300

会員テーブル

No	会員名	電話番号
1000	A	090-XXXX-XXXX

コラム

「正規化」とは(2)

先ほどの例の正規化のメリット

以下の操作、条件であったとします。

- Aさんの電話番号を「090-YYYY-YYYY」に変えようとする。
- 過去にAさんからの注文が100件あるとする。

非正規化の場合	正規化の場合
注文テーブルのAさんの行を100行更新する必要がある （処理負荷が重い）	会員テーブルのAさんの行を1行だけ更新すれば良い （処理負荷が軽い）

3.2.5 ログ

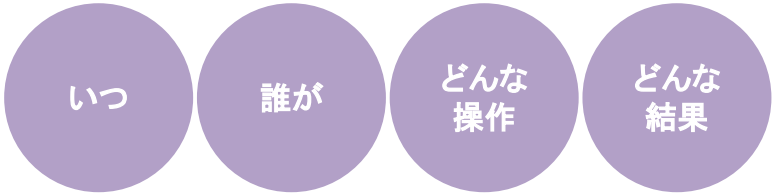
ノーコード
ローコード

システム上重要な操作については、ログを記録（ロギング）することが望ましいでしょう。
重要な操作とはログイン、ログアウト、データの登録／更新／削除／ダウンロードなど、主にデータを扱う機能のことを指します。
ロギングを行うと、予期せぬエラーが発生した場合に、いち早くバグを検知することが可能となります。
ロギングは利用サービスにより、記録方法が異なるため各種サービスを確認してください。

ロギングのメリット

- 正当、あるいは不正な操作の証拠となる
- 誤った操作や意図しない処理によってトラブルが発生した時
  迅速で正確な状況確認が可能
- 機能の使用頻度など、システムの効果検証が可能
- システムのバグの発見や処理負荷の検知が可能

ログに含む情報



3.3 テスト

ノーコード
ローコード

テストの目的、テスト設計は主に以下の観点で行います。

- 想定通りに動作するか（機能テスト）
- 想定通りのレスポンスか（性能テスト）

テスト設計
テスト設計書を作成し、レビューを実施します。

テストデータ準備
テストを実施するためのデータを用意します。

テストでは、正常動作はもちろん、異常動作の確認も行います。
例えば、想定していない操作を行ったり、値を入力するなどして確認します。

テストと合わせ、セキュリティ上問題ない動作であること、情報管理が適切に行われていることを確認します。例えば、動作自体は良好だが、機密情報が管理者以外でも閲覧可能な状態は、重大なセキュリティ事故に繋がります。

3.3.1 テスト設計（1）

ノーコード

ローコード

テスト設計

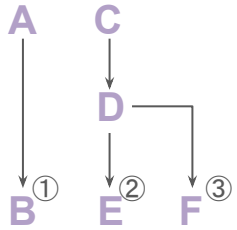
- ドキュメントとして作成します。
- テスト項目が「実施済み」なのか「未実施」なのか、状態が分かるようにしましょう。
- テスト設計が完了したら、要件定義と相違が無いかのレビューを行いましょう。

3.3.1 テスト設計(2)

ノーコード

ローコード

パターンの網羅

入力値の網羅	文字を入力する欄では、入力可能な文字の種類(ローマ字／日本語／数字／日付など)が決まっています。 入力可能な種類以外の値が入ってきた場合の動作パターンを網羅する必要があります。 【例】Eメールの入力欄に氏名のような値が入力された場合、 エラーとしてメッセージを表示する	○ E-mail taro_nocoloco@example.com × E-mail ノコロコ太郎
条件分岐の網羅	Aという動作をする→ Bという結果になる Cという動作をした時、Dという条件に合致→ Eという結果になる など処理が分岐した場合のすべてのパターンを網羅する必要があります。	
境界値の網羅	境界値を意識したパターンを作成します。 【例】Aが1～20個まで登録できるとした場合、 0個 → 登録不可 ／ 1個(最小値)→ 登録可 20個(最大値)→ 登録可 ／ 21個 → 登録不可	

3.3.2 テストの実行

ノーコード

ローコード

概要

選定

開発

運用

正常系・異常系

	観点	サンプル条件	サンプル結果
正常系	想定された**値**が入ってきた時、 想定した結果になること	項目「メールアドレス」に 「hoge@example.com」を入力する	正常に登録されること
	想定された**操作**がされた時、 想定した処理が実行されること	存在するアカウント情報 「ID」と「パスワード」を入力して、 ログインボタンを押下する	正常にログインできること
異常系	想定されて**いない**値が入ってきた時、 想定した結果になること	項目「メールアドレス」に 「山田太郎」を入力する	登録されず、 エラーメッセージが表示されること
	想定されて**いない**操作がされた時、 想定した処理が実行されること	存在しないアカウント情報 「ID」と「パスワード」を入力して、 ログインボタンを押下する	ログインはできず、 エラーメッセージが表示されること

3.3.2 テストデータ準備

ノーコード
ローコード

テストに使用するデータは主に2種類あります。

1 **テスト設計に基づいたデータ**

テストパターンを網羅するため、テスト設計に基づいたデータを準備する必要があります。

2 **本番に近しいデータ**

予期しない文字数や文字の種類が使用されている場合があるため、本番に近しいデータによるテストも行うことが望ましいです。
その際には個人情報などが含まれている場合があるため、取り扱いには十分注意しましょう。

3.4 リリース（1）

ノーコード
ローコード

リリースとは、開発したシステムを検証環境から本番環境へ移行することです。
本番環境へ移行する際は、既存システムへ影響を与える可能性があるため、細心の注意が必要です。

リリース判定
新システムを現場で稼動させてよいかを判定するもので、現在の準備状況を総合的に判断します。
システムだけではなく業務の状況も踏まえて、リリースの判断を行いましょう。

【判定基準例】

テスト完了	テストが完了し、テスト時に発生した不具合が解消されていることを確認します。
システムパフォーマンス	構築したシステムが実運用に耐えうる仕様になっているかチェックします。 例えば、レスポンスタイムの確認は、実業務をスムーズに回すために重要なポイントとなります。
リリース機能の周知完了	関係者に対し、新たにリリースするシステムや機能についての周知を行っていることを確認します。 利用者向けのマニュアルを作成／配布したり、現場で利用するユーザー向けの研修や説明会を実施するのも良いでしょう。実施する場合は、それらが完了していることを判定基準にします。
バージョン確認完了	検証環境と本番環境でツールのバージョンが異なる場合、バージョン違いによる影響有無を確認します。 影響がある場合は、対処方法を明確にする必要があります。
各種申請の確認完了	本番環境へ変更を行うにあたり、申請が必要な場合があります。必要となる申請内容／申請先／申請方法を把握しておきましょう。リリース作業前に必要な申請は完了しているようにしてください。

3.4 リリース(2)

ノーコード

ローコード

リリースを行う際は、手順書やチェックリストを作成し、作業を確実に実施できるようにしましょう。
手順書にはトラブル発生時の対応も記載しておくことが望ましいです。
手順書やチェックリストに記載する一般的な内容は以下になります。

リリース対象(抜け漏れがないか要確認)

- リリース対象一覧
- リリースするためのタスク
- リリース作業手順

リリース体制

- リリース当日の体制と役割分担
  「誰が」「何の作業を」実施するかを記載しましょう。
  作業は1人で行わず、立会者と共に2人以上の体制で行うことが望ましいです。

関係者の連絡先

- リリース作業実施部門の管理者
- リリースするシステムを利用する(している)業務担当者
- トラブル発生時の連絡先

3.4 リリース(3)

トラブル発生時の対応

- リリース作業の中止／延期／続行の判断基準
- リリース作業の中止／延期／続行の判断を行う責任者
- トラブル対応要員
- ツールのサポート連絡先

万が一、トラブルが発生してしまった場合でも、慌てずに対処しましょう。

例えば・・・処理でエラーが発生してしまった場合
エラーメッセージやログファイルを調査し、根本原因を特定します。

例えば・・・導入した機能の反応が遅い、環境の負荷が高い場合
リソースの追加／一部機能の停止による負荷削減を検討します。

詳しくは『4.1.3 障害発生時の対応』を参照してください。

運用／展開

4.1 管理方法

この章では、ノーコード／ローコード開発を安全に進めるために、また、ガバナンスを機能させながら内製の効果をより得ることができるように、管理が必要となる事項について記載していきます。

1. 開発時の推奨事項
2. 保守体制
3. 障害発生時の対応
4. ライセンス管理
5. セキュリティ
6. バージョンアップ
7. バックアップ

4.1.1 開発時の推奨事項

ノーコード
ローコード

開発を安全に進めていくための取組みとして、以下を推奨します。

開発者の育成

マネジメントや開発・保守などに必要なスキルを獲得できるよう人材育成を行いましょう。
開発担当者の異動や退職のリスクがあるため、開発担当者は複数人居ることが望ましいです。
スキルの取得や継続には、勉強会の開催、ツールの提供元によるハンズオン研修への参加などの方法があります。
その他、サポートサイトや情報収集先を共有するなど、情報を得るための手段を記録しておくことも有効です。

開発標準／ひな型の作成

開発プロセスの標準化や、設計書／テスト仕様書などのドキュメントの標準化、開発ルールの策定など（開発標準）を定めておくと、システム全体の統一感が生まれ、利用しやすさや運用のしやすさを実現できます。
また、作成可能なツールである場合は、ひな型を作成しておくことで、より統一性を保つことができます。

仕様／操作方法のドキュメント化

『2 選定』で整理した業務内容や、ツール作成時の仕様、ツールの操作方法はドキュメント化しておきましょう。
ツールの操作方法は、作る側／使う側の両方のドキュメントを作成するといいでしょう。
ツールを利用する上で必要となる最低限の情報を記載します。動画で操作手順を残すのも一つの手です。
ドキュメントとして残しておくことで、開発担当者が入れ替わった際の引継ぎがスムーズに行えます。

4.1.2 保守体制

ノーコード
ローコード

利用者からの問合せ対応窓口を設け、問合せや要望など、対応履歴を適切に管理するようにしましょう。
問合せ窓口は一本にまとめ、問合せ先を利用者に周知してください。
問合せ窓口は、「誰が」「何を」するのか、役割を明確にした体制を整えてください。

外部事業者にシステム開発を委託した場合

保守費を十分に確保する必要があります。
サポート契約の内容や、問合せ先をしっかりと確認し、保守担当者間で共有してください。

自社で開発した場合

必要な時には外部事業者からの支援を受けられる体制があると良いでしょう。

4.1.3 障害発生時の対応（1）

ノーコード
ローコード

障害とは

ツール自体やツールを導入している環境、通信環境などに問題が生じ、正常に動作しなくなる状態のことです。
また、その原因となった問題や不具合のことを指します。

【障害発生時に起こることの例】

- データの消失／破壊
- 処理能力の低下
- 外部との通信ができなくなる
- 一部の機能の損失
- システムの完全停止

【障害発生原因の例】

- 操作ミス
- ツールの設定ミス
- プログラミングの誤り（バグ）
- 想定を超える負荷がかかった
- 機器の故障／破損
- 外部からの攻撃
- コンピューターウィルス

SaaS型では、ツールの提供元で発生した障害が原因の場合があります。
自身では対処のしようがなく、情報を待つしかないこともあります。

4.1.3 障害発生時の対応（2）

ノーコード
ローコード

障害が発生した時

以下のことを並行して速やかに進める必要があります。

1 **関係者への周知／連絡**

早急に影響を受ける部門や利用者に連絡をしましょう。

2 **代替手段への切り替え**

業務を止めないことが最優先となります。

運用でカバーできる場合（※）は、プログラムや設定の修正は行わず、一旦運用方法によるカバーで対応します。

※特定の操作でエラーが発生し、その操作が回避できる場合など

3 **原因究明／復旧作業**

業務への影響度に応じて対策を検討し、暫定的または恒久的な復旧を行います。

すぐに原因が分かり、修正に時間が掛からない場合は、プログラムや設定の修正で対応します。

すぐに復旧することが難しい場合は、暫定的な復旧を行います。暫定的な復旧を行った場合は、プログラム修正などを行った後、利用時間外や利用者の少ない時間帯に改めてリリースを行い、恒久的な復旧を行います。

4.1.3 障害発生時の対応（3）

ノーコード
ローコード

利用できる状態への復旧

以下のことを確認しましょう。

1 **検証環境で当該のエラーが起こらないことを確認**

2 **プログラム修正／設定変更による他の不具合が発生していないかを確認**

プログラムや設定の変更を行う際は、変更を行う機能のバージョンに気を付けましょう。
また、複数人で対応に当たる場合は、同じ機能を複数人で触らないように注意が必要です。

3 **障害発生時と復旧時でデータの整合性を確認**

再操作やデータの修正が必要になる場合もあるので、よく確認しましょう。

サービスが使用可能な状態であることや、応答が良好であることを定期的に監視するツールもあります。このようなサービスを使うことで障害を検知することができるため、ツールの使い方によっては監視ツールの導入を検討してもいいかもしれません。

4.1.4 ライセンス管理

ノーコード
ローコード

余剰ライセンスをできる限り削減し、必要最低限のライセンスで運営することが望ましいです。
そのために、ライセンスの利用状況を一元管理できる仕組みが必要となります。定期的に情報を更新し、状況を把握できるようにしましょう。
開発用ライセンスと、機能を利用するためのライセンスが分かれている場合は、それぞれの情報を管理しましょう。

台帳で管理する内容

1 **ライセンスの種類**

ライセンスの種類により利用可能な機能が異なる場合があります。
どのライセンスを契約しているのか管理する必要があります。

2 **ライセンスの数量**

ライセンス数によって利用料が異なる場合が多いです。
契約しているライセンス数と、それに対する使用済みライセンス数を常に把握できている状態にしましょう。
異動や退職により、ツールの利用が不要となった人については、アカウントの停止や削除、ツールのアンインストールなどによりライセンスを返却してもらいましょう。

3 **導入済み端末**

導入端末に変更があった場合は、台帳を更新し、常に最新の情報を管理できるようにしましょう。

4.1.5 セキュリティ

ノーコード
ローコード

開発用アカウントの管理

開発アカウントの取り扱いには十分気を付けましょう。
管理者権限は何でもできてしまうため、管理者権限を与えるアカウントは最小限に留めるようにします。
管理者に変更があった場合は、権限の設定変更やアカウントの無効化／停止をし、利用できない状態にしましょう。

不正利用／認証情報の流出、ライセンス違反などのリスクがあるため、
共通アカウントの使いまわしは行わないようにしてください。

パスワードの管理

設定するパスワードには、以下のようなルールを設けるといいでしょう。

- 使いまわしパスワードの使用禁止
- 「○○文字数以上」といった最低文字数のルール
- 「英数字記号○種類以上」といった文字種のルール

ツールにより設定可能なパスワードのルールが異なります。設定可能な条件の中でルールを検討しましょう。

4.1.6 バージョンアップ（1）

ノーコード
ローコード

アップデート

利用するツールは定期的にアップデートされていくものがあります。

アップデートの中には以下のものが主に含まれます。

- 機能の追加／削除
- 見た目／操作性の変更
- バグの修正

1

インストール型

アップデートを適用するか否かの検討が必要となります。

アップデートの適用は必須ではありませんが、次の観点から総合的に適用の判断を行いましょう。

- アップデートの内容
- 現行バージョンの保守期限
- ライセンス更新の要／不要　など

ただし、バグの修正は脆弱性を改善するためのものであることが多いです。

セキュリティの観点からも積極的にアップデートをすることが望ましいです。

4.1.6 バージョンアップ（2）

ノーコード
ローコード

アップデート適用の注意点

- 既存機能が動かなくなる場合があります。
  アップデートを適用する前に、必ず動作検証を行いましょう。
  検証環境にアップデートを適用し、テストを実施するという方法が一般的です。

- 利用者へアップデート適用に関する告知を行いましょう。
  アップデート適用時に、ツールの利用を停止する場合や、操作性に変更がある場合は、事前に案内しておきましょう。
  →『3.4 リリース』と基本的な手順は同じです。

2

SaaS型

アップデートは自動的に行われ、特別な対応が必要ないことが一般的です。
自動的にアップデートが行われるため、ある日突然、見た目や操作性が変わることがあります。

サービス提供の終了／アップデートの停止

ノーコード／ローコードツールの中には、サービス提供が終了してしまうもの、アップデートが停止されてしまうものもあります。
その場合、利用中のツールから別のツールへの乗り換えを検討する必要があります。

4.1.7 バックアップ

ノーコード
ローコード

バックアップの取得が可能なツールの場合は、定期的にバックアップを取得することを推奨します。
利用するツールがバックアップが可能か、バックアップが可能な利用プランなのかを確認しておきましょう。

ツールにバックアップ機能がない場合は、設定ファイルを定期的にコピーして保管しておくなどの対応が可能な場合もあります。

データベースを利用している場合は、データベースのバックアップ取得や、コピーなども行いましょう。

定期的にバックアップを取得することで、障害発生時の復旧がスムーズになります。
障害発生時の復旧手順を明確にしておくといいでしょう。

4.2 システム改善のポイント

ツールの利用／検証／改善のサイクルを回すことで、よりツールの導入効果を高めていきましょう。
また、導入の成果を社内に積極的に発信し、他部門でのツール利用を促進していきましょう。

1 **効果検証**

効果検証を行うことで、ツールの利用が有用なものであったか判断します。
その情報を元に今後の開発を、より課題解決につながるものにしていく必要があります。

2 **システムの改善**

効果検証の結果、ツール利用に課題のあった箇所については改善を行いましょう。
効果のあった点については、他の業務改善のために利用ができないか検討するといいでしょう。

3 **利用促進**

ツールの利活用を促進するため、社内でコミュニティや意見交換の場を設けるといいでしょう。

4.2.1 効果検証

ノーコード
ローコード

ツールを導入したことによる効果がどの程度あるか、ツールを利用する上での課題は何かを検証します。
ツール利用者へのヒアリングやアンケートの収集などを行うといいでしょう。
以下に検証項目の例を記載します。

1 **解決したい業務課題への評価**

『2 選定』で整理した業務課題が解決されているかを検証します。
ヒアリングやアンケートの結果が良かった点は業務課題が解決されたところ、
悪かった点は改善の余地があるところと判断してもよいでしょう。

2 **ツールの活用状況**

ツールの利用頻度、有効活用ができているかなどを確認します。
有効活用ができている場合は、より一層の活用に向けた企画検討を推進しましょう。
有効活用ができていない場合は、当初の利用目的の達成に向けて、ツールの利用方法を見直しましょう。

3 **ツールの使い勝手**

利用者がより使いやすいものにするために、ツールの使い勝手についても確認しましょう。
より使い勝手のよいシステムにするために、利用者の意見を取り入れ、改善を重ねるといいでしょう。

4.2.2 システムの改善

ノーコード

ローコード

効果検証を行った結果、効果のなかった点や課題となった点については改善を行いましょう。
稼動中のシステムへの改善を行う際に注意する点について記載します。

既存機能への影響確認

既存機能への影響有無を確認する必要があります。
追加／修正した機能のみではなく、他の既存機能が正常に動作することを確認します。

作成した機能のバージョン管理

バージョン管理が可能なツールの場合は、作成した機能のバージョン管理を行いましょう。
複数人で修正を行う場合や、検証環境と本番環境でバージョンが異なる場合など、古いバージョンの機能を修正することで不具合が発生する場合があります。既に修正済みであったはずの内容が喪失してしまうことで、過去の不具合が再発したり、予期せぬ不具合が発生したりします。
必ず、更新日付を確認し、最新の日付のものに対して、修正を行うようにしましょう。

4.2.3 利用促進

ノーコード
ローコード

ツールの利活用を促進するため、社内でコミュニティや意見交換の場を設けるといいでしょう。
業務改善を実現した事例の共有／発信を積極的に行いましょう。

他部門への展開時に気を付けること

契約の見直し

利用者数：　追加利用者分のライセンスが足りない場合
　　追加で契約する必要があります。
性能契約：　利用者増加に伴い負荷が増加する場合
　　対応できる性能に変更する必要があります。

利用を許可する範囲の検討

すべてのメンバーがすべての機能を利用できるようにすることは、
セキュリティリスクにつながります。
システムの利用範囲に対して、制限をかける必要があります。

問合せ先

サポートへの問合せ回数に制限がある場合があります。
一部の利用者が使い切ることがないよう監視／制限をする
必要があります。
サポートへの問合せ窓口が一本となるよう体制を整えましょう。

アクセス権限

部外秘の情報など、他部門に見せられないデータがある場合は、
アクセス権を設定しましょう。
組織／ユーザー単位で、閲覧／編集可能な範囲を設定する必要が
あります。

おわりに

「ノーコード／ローコード開発スタートガイド」はいかがだったでしょうか？

「ノーコード／ローコードを利用したいけど何から始めていいのか分からない」という方にとって、少しでも導入の想像ができていれば幸いです。注目されている技術ではありますが、世の中への浸透はまだまだと言え、黎明期と呼んでも差し支えない時期かと思います。とはいえ人材不足の一途をたどるのも待ったなしの状態で、まさに今、社内でもシステム構築が可能なノーコード／ローコードを試してみる絶好のタイミングかと思います。

本スタートガイドではノーコード／ローコードとは何か、だけではなくシステム開発をするために必要なノウハウをコンパクトにまとめられていると思いますので、ぜひお役立てください。

また、本スタートガイドの作成にあたり、アンケートにお答えいただいた皆様にこの場を借りてお礼を申し上げます。
ご協力ありがとうございました。

チーム・ノコロコ

TEAM NOCOLOCO

ノーコード／ローコード開発 スタートガイド 1.3版

チーム・ノコロコ