開発戦士
ワンダラレンジャー

第1話　目指せユーザー開発のヒーローズ

ノーコード／ローコード時代のシステム開発 分科会

説明しよう！

マンダラレンジャーは何故生まれたのか！？

迫りくる 未来の断崖絶壁、その名も「2025年の崖」「2030年問題」！
IT人材が足りない！そんな危機に立ち向かうため、
今、最強の開発戦士「マンダラレンジャー」が誕生した！
ノーコード/ローコードツールとマンダラチャートという切り札を手に、
非IT部門を「ユーザー開発のヒーローズ」へと変身させる！

マンダラチャートとは！？

その正体は戦略を視覚化する究極の秘密兵器！
中央に最終目標を据え、周囲の8つのエリアに最終目標達成に必要な要素を、
更にその要素を満たすための具体的な行動計画を、周囲の8つのエリアにそれぞれ配置する！
まるでヒーローの必殺技を組み立てるが如く、複雑な戦略を一目で理解できる驚異のダイアグラム！

きみもユーザー開発のヒーローズの一員になろう！

はじめに

プログラミングせずにシステム開発できるノーコード/ローコードツール※ は、使いこなすことができればユーザー部門（非IT部門）が主体となって業務課題を解決できるようになる魔法のようなツールになります。

しかしながら、部門内で内製化を成熟させるためには、単にツールの使用方法を理解するだけでなく、業務分析や要件定義といったシステム開発を取り巻く多くの要素を知る必要があります。

※ ローコードツールは一部の機能実装にプログラミングが必要な場合があります。

このガイドでは、マンダラチャートを通して
主にユーザー部門の方に向けてノーコード/
ローコードツールで開発する際に抑えるべき
ポイントや学び方について解説しています。

アプリケーション開発を通じて
業務改善を推し進め、
ユーザー開発のヒーローを
一緒に目指しましょう！

【Tips】ノーコード/ローコードツールとは

プログラミング知識がなくても、アプリやウェブサイトを作成できるツール

視覚的なインターフェースで作成するため、開発期間を短縮することができる

専門知識が不要なため、学習コストを削減することができる

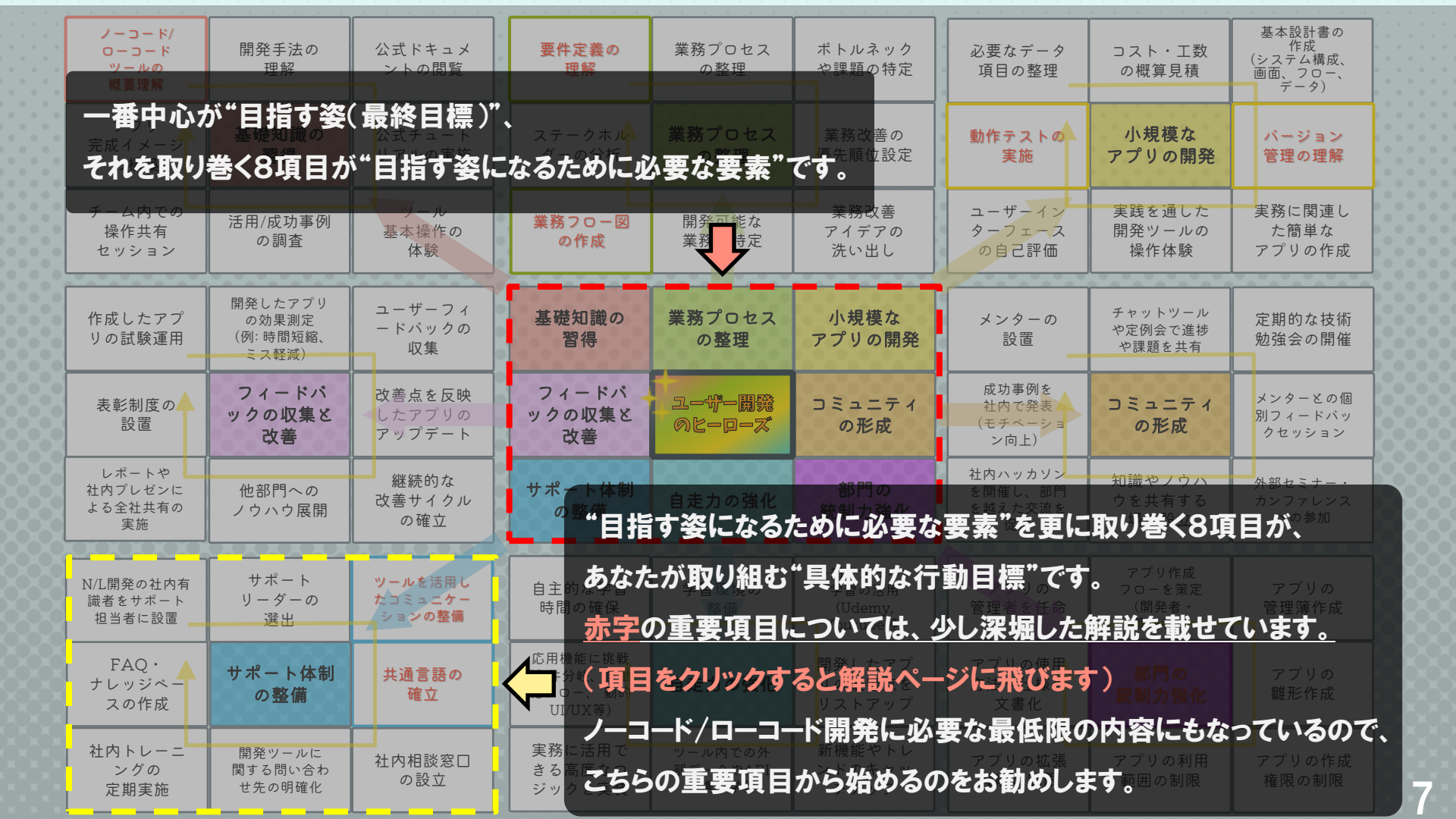
【代表的なツールの紹介】

Microsoft Power Automate

ビジネスプロセスの自動化と最適化を目的としたツール。サーバーからデータを抽出したり、アプリ間のデータのやり取りができる。また、データ転送やメール送信などのタスクを自動化することができる。
例えば、受け取ったメールのデータの必要な情報を抽出して、サーバー内に項目別に保存し、
タスクリストの作成などを自動化することができる。

付録①：Power Automateの活用例を見る


ノーコード/ローコードツールの概要理解
開発手法の理解
公式ドキュメントの閲覧
アプリ完成イメージの構築
基礎知識の習得
公式チュートリアルの実施
チーム内での操作共有セッション
活用/成功事例の調査
ツール基本操作の体験
要件定義の理解
業務プロセスの整理
ボトルネックや課題の特定
ステークホルダーの分析
業務プロセスの整理
業務改善の優先順位設定
業務フロー図の作成
開発可能な業務の特定
業務改善アイデアの洗い出し
必要なデータ項目の整理
コスト・工数の概算見積
基本設計書の作成（システム構成、画面、フロー、データ）
動作テストの実施
小規模なアプリの開発
バージョン管理の理解
ユーザーインターフェースの自己評価
実践を通した開発ツールの操作体験
実務に関連した簡単なアプリの作成
作成したアプリの試験運用
開発したアプリの効果測定（例: 時間短縮、ミス軽減）
ユーザーフィードバックの収集
表彰制度の設置
フィードバックの収集と改善
改善点を反映したアプリのアップデート
レポートや社内プレゼンによる全社共有の実施
他部門へのノウハウ展開
継続的な改善サイクルの確立
基礎知識の習得
業務プロセスの整理
小規模なアプリの開発
フィードバックの収集と改善
ユーザー開発のヒーローズ
コミュニティの形成
サポート体制の整備
自走力の強化
部門の統制力強化
メンターの設置
チャットツールや定例会で進捗や課題を共有
定期的な技術勉強会の開催
成功事例を社内で発表（モチベーション向上）
コミュニティの形成
メンターとの個別フィードバックセッション
社内ハッカソンを開催し、部門を越えた交流を促進
知識やノウハウを共有する場の設立
外部セミナー・カンファレンスへの参加
N/L開発の社内有識者をサポート担当者に設置
サポートリーダーの選出
ツールを活用したコミュニケーションの整備
FAQ・ナレッジベースの作成
サポート体制の整備
共通言語の確立
社内トレーニングの定期実施
開発ツールに関する問い合わせ先の明確化
社内相談窓口の設立
自主的な学習時間の確保
学習環境の整備
外部オンライン学習の活用（Udemy, YouTube等）
応用機能に挑戦（条件分岐、自動化フロー、動的UI/UX等）
自走力の強化
開発したアプリの改善点をリストアップ
実務に活用できる高度なロジックを実装
ツール内での外部データやAPI連携の学習
新機能やトレンドのキャッチアップ
アプリの管理者を任命
アプリ作成フローを策定（開発者・管理者）
アプリの管理簿作成
アプリの使用方法や仕様の文書化
部門の統制力強化
アプリの雛形作成
アプリの拡張機能の制限
アプリの利用範囲の制限
アプリの作成権限の制限
チャートの見方は次ページにて解説


一番中心が“目指す姿（最終目標）”、
それを取り巻く8項目が“目指す姿になるために必要な要素”です。
“目指す姿になるために必要な要素”を更に取り巻く8項目が、
あなたが取り組む“具体的な行動目標”です。
赤字の重要項目については、少し深堀した解説を載せています。
（項目をクリックすると解説ページに飛びます）
ノーコード/ローコード開発に必要な最低限の内容にもなっているので、
こちらの重要項目から始めるのをお勧めします。
ノーコード/ローコードツールの概要理解
開発手法の理解
公式ドキュメントの閲覧
要件定義の理解
業務プロセスの整理
ボトルネックや課題の特定
必要なデータ項目の整理
コスト・工数の概算見積
基本設計書の作成（システム構成、画面、フロー、データ）
完成イメージの共有
基礎知識の習得
公式チュートリアルの実施
ステークホルダーの分析
業務プロセスの整理
業務改善の優先順位設定
動作テストの実施
小規模なアプリの開発
バージョン管理の理解
チーム内での操作共有セッション
活用/成功事例の調査
ツール基本操作の体験
業務フロー図の作成
開発可能な業務の特定
業務改善アイデアの洗い出し
ユーザーインターフェースの自己評価
実践を通した開発ツールの操作体験
実務に関連した簡単なアプリの作成
作成したアプリの試験運用
開発したアプリの効果測定（例: 時間短縮、ミス軽減）
ユーザーフィードバックの収集
基礎知識の習得
業務プロセスの整理
小規模なアプリの開発
メンターの設置
チャットツールや定例会で進捗や課題を共有
定期的な技術勉強会の開催
表彰制度の設置
フィードバックの収集と改善
改善点を反映したアプリのアップデート
フィードバックの収集と改善
ユーザー開発のヒーローズ
コミュニティの形成
成功事例を社内で発表（モチベーション向上）
コミュニティの形成
メンターとの個別フィードバックセッション
レポートや社内プレゼンによる全社共有の実施
他部門へのノウハウ展開
継続的な改善サイクルの確立
サポート体制の整備
自走力の強化
部門の統制力強化
社内ハッカソンを開催し、部門を越えた交流を
知識やノウハウを共有する
外部セミナー・カンファレンスの参加
N/L開発の社内有識者をサポート担当者に設置
サポートリーダーの選出
ツールを活用したコミュニケーションの整備
自主的な時間の確保
学習用途（Udemy、
管理者を任命
アプリ作成フローを策定（開発者・
アプリの管理簿作成
FAQ・ナレッジベースの作成
サポート体制の整備
共通言語の確立
応用機能に挑戦（自分
UI/UX等）
開発したアプリ
リストアップ
アプリの使用
文書化
部門の統制力強化
アプリの雛形作成
社内トレーニングの定期実施
開発ツールに関する問い合わせ先の明確化
社内相談窓口の設立
実務に活用できる高度
ツール内での外
新機能やトレ
アプリの拡張
アプリの利用範囲の制限
アプリの作成権限の制限

ノーコード/ローコードツールの概要理解	開発手法の理解	公式ドキュメントの閲覧
アプリ完成イメージの構築	**基礎知識の習得**	公式チュートリアルの実施
チーム内での操作共有セッション	活用/成功事例の調査	ツール基本操作の体験

【基礎知識の習得】

ノーコード/ローコードとは何か？どのようにアプリケーション開発を行うのか？

利用するツールの公式情報やハンズオンなどの体験を通して、ノーコード/ローコード開発の具体的な方法（操作方法等）を習得することがこの項目の目的となります。

要件定義の理解	業務プロセスの整理	ボトルネックや課題の特定
ステークホルダーの分析	**業務プロセスの整理**	業務改善の優先順位設定
業務フロー図の作成	開発可能な業務の特定	業務改善アイデアの洗い出し

【業務プロセスの整理】

実際に開発を行う前に、どのようなアプリケーションを開発するのかを事前に定義することも重要です。

定義するために必要な情報（現状の業務内容や課題）の整理を目的とします。

業務プロセスを整理することで、無駄を省き効率化を図りながら、

問題の早期発見や複数人での開発におけるスムーズなコミュニケーションが可能となります。

必要なデータ項目の整理	コスト・工数の概算見積	基本設計書の作成（システム構成、画面、フロー、データ）
動作テストの実施	小規模なアプリの開発	バージョン管理の理解
ユーザーインターフェースの自己評価	実践を通した開発ツールの操作体験	実務に関連した簡単なアプリの作成

【小規模なアプリの開発】

ノーコード/ローコード開発では、初期段階は小規模な開発を行い、初心者でも短期間で達成可能です。

開発経験は複雑な開発への足がかりとなり、座学だけでは得られないスキルを習得できます。

設計、開発、テストの基本的な流れを通じて理論と実践を結びつけます。

作成したアプリの試験運用	開発したアプリの効果測定（例: 時間短縮、ミス軽減）	ユーザーフィードバックの収集
表彰制度の設置	**フィードバックの収集と改善**	改善点を反映したアプリのアップデート
レポートや社内プレゼンによる全社共有の実施	他部門へのノウハウ展開	継続的な改善サイクルの確立

【フィードバックの収集と改善】

ノーコード/ローコード開発ツールの導入初期には、

期待通りの効果が得られなかったり、使用中に多くの課題が見つかります。

プロトタイプ（試作品）を作り、ツールの操作性や業務適合性を現場で実践し、

フィードバックする工程がアプリケーションの品質向上に繋がります。

【サポート体制の整備】

アプリケーションの開発・運用・管理を円滑にするためには、部門間のサポート体制の整備が不可欠です。

あらかじめ体制を整備しておくことで、システムの認識齟齬による手戻りや、

開発運用における相談や問い合わせ時に発生しやすいタイムロスを防ぐことができます。

N/L開発の社内有識者をサポート担当者に設置	サポートリーダーの選出	ツールを活用したコミュニケーションの整備
FAQ・ナレッジベースの作成	**サポート体制の整備**	共通言語の確立
社内トレーニングの定期実施	開発ツールに関する問い合わせ先の明確化	社内相談窓口の設立

メンターの設置	チャットツールや定例会で進捗や課題を共有	定期的な技術勉強会の開催
成功事例を社内で発表（モチベーション向上）	**コミュニティの形成**	メンターとの個別フィードバックセッション
社内ハッカソンを開催し、部門を越えた交流を促進	知識やノウハウを共有する場の設立	外部セミナー・カンファレンスへの参加

【コミュニティの形成】

非IT部門のノーコード/ローコード開発を部門内で成熟させるためには、

開発担当者が特定の人物に偏ることは避けるべきです。

適切なコミュニティを形成することで、開発担当者の孤立や挫折を防ぎ、

互いの心理的な支えやアイデアの共有を通して

開発担当者のモチベーションの維持・向上につながります。

【自走力の強化】

自走力とは、ノーコード/ローコード開発および改善サイクルを自部門で完結させる能力を指します。

自走力を強化することによりIT部門への負担・依存度を下げることができますが、そのためには

個人レベルでの主体的な学習習慣の構築、それをサポートする組織全体での文化・仕組み作りが求められます。

自主的な学習時間の確保	学習環境の整備	外部オンライン学習の活用（Udemy, Youtube等）
応用機能に挑戦（条件分岐、自動化フロー、 動的UI/UX等）	**自走力の強化**	開発したアプリの改善点をリストアップ
実務に活用できる高度なロジックを実装	ツール内での外部データやAPI連携の学習	新機能やトレンドのキャッチアップ

【組織の統制力強化】

非IT部門開発における代表的な懸念事項である野良システム（管理が行き届いていないシステム）の乱立や、管理者不在となったシステムのしわ寄せがIT部門の負担となっては本末転倒です。

開発および管理におけるルールを制定し、可能な限り部門内で解決できる体制作りを目指します。

アプリの管理者を任命	アプリ作成フローを策定（開発者・管理者）	アプリの管理簿作成
アプリの使用方法や仕様の文書化	**部門の統制力強化**	アプリの雛形作成
アプリの拡張機能の制限	アプリの利用範囲の制限	アプリの作成権限の制限

ノーコード/ローコードツールの概要理解

そもそもノーコード/ローコードツールとは何か？
“その特徴と、何ができるか“を学ぶ。

この項目を学習/実践することで…

- プログラミング未経験でも実現可能なことが理解できる。
- アプリケーションの開発イメージが湧き、
  後続学習のモチベーションアップに繋がる。
- ノーコード/ローコードツールの特徴を掴むことで、
  ツールの利点を活かした設計が可能となる。

ノーコード/ローコードツールの概要理解

この項目を学習/実践しないと…

- 👎 ノーコード/ローコードツールで“何が実現できるのか”が分からない…
- 👎 アプリケーションの開発イメージが湧かない…
- 👎 モチベーションが上がらず学習が頓挫してしまう…

ノーコード/ローコードツールの概要理解 までの道のり

① 開発手法の理解

ノーコード/ローコード/プロコード開発の
それぞれの特徴を知る

② ツールの基本操作を体験

公式サイトやドキュメントを見ながら、
実際に開発ツールを触ってみる

③ 成功事例を学ぶ

開発したいアプリケーションの機能と
類似した事例を調査

④ 今後の展望を行う

身近な業務改善から、革新的なものまで！
ツールを活用して何ができそうかをイメージする

【Tips】システム開発にはどのような手法がある？

	ノーコード開発	ローコード開発	プロコード開発（スクラッチ開発）
開発方法	マウス操作を主とした、機能や処理ロジックの設定変更による開発	ノーコード開発をベースに、一部のカスタマイズにはプログラミングを要する	プログラミングによる開発
プログラミングスキル・知識	必要なし	ほぼ必要無し	必要
開発期間	短		長
カスタマイズ性	低		高
習得難度・教育コスト	低		高

要件定義

ノーコード/ローコードツールを用いたツール開発において、
“明確な目標設定と開発範囲の定義“を行う。

この項目を学習/実践することで…

- 手戻りを防止し、開発工数を低減できる。
- 無計画な仕様変更を防ぎ、品質が確保できる。
- タスクが明確になり、進捗管理が容易になる。
- チーム内の認識を合わせることで、チームワークが向上する。

要件定義

この項目を学習/実践しないと…

- 当初の想定とは異なるアプリケーションが出来上がってしまい、手戻りが発生する…
- 行き当たりばったりな仕様追加をしてしまい、機能が複雑化してしまう…
- 残タスクの把握が難しく、進捗管理ができない…

要件定義 が上手くなるためには何を学べばいい？

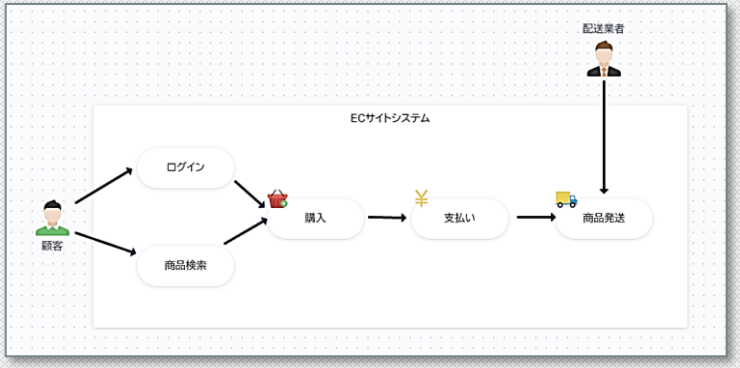
要件定義 が上手くなるためには何を学べばいい？

【Tips】UMLとは？

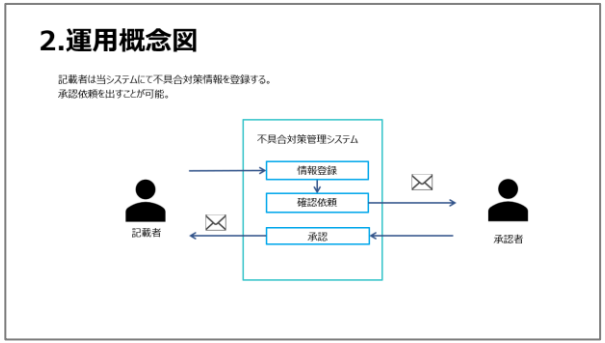
システムの設計や構造を**視覚的に表現するため**の標準化された図法。

- **ユースケース図：**
  システムとユーザーとの関係性を表す。
- **クラス図：**
  システムに存在すべき構成要素を図式化したもの。
- **シーケンス図：**
  クラスやオブジェクト間のやり取りを時間軸に沿って図式化したもの。

<ユースケース図のイメージ>



要件定義 って具体的にどういうもの？



<定義書に記載する主な項目>

目次 改訂履歴	書類に含まれる情報や最新情報を示す
背景・目的	なぜこのシステムが必要なのか、 どのような目的で利用するのかを示す
運用イメージ 概念図	システムを誰がどのように運用するのかを図で示す
主要機能要件	システム内の主要な機能について図や表を用いて示す
画面遷移図	システム画面がどのように遷移していくのか図で示す

付録②：要件定義書のサンプルを見る

業務フロー図

ある業務において“誰が、いつ、何を、どのように”するのかを図式化したもの。
業務の全体像を客観的に把握することができ、
後続の開発フェーズにおいて有用な指標となる。

この項目を学習/実践することで…

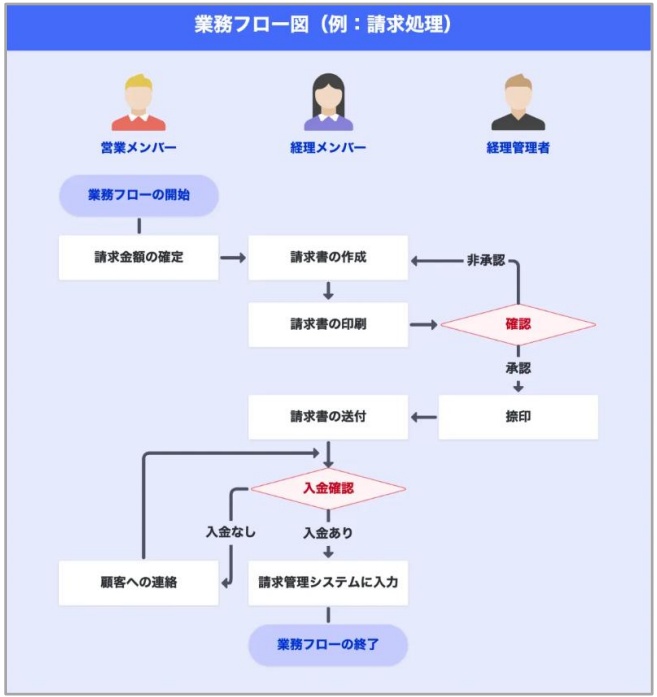
- 関係者間での共通認識が形成しやすくなる。
- 業務の問題点や改善点の早期発見に繋がる。
- アプリケーション（システム）に落とし込む範囲が特定できる。
- 開発が必要な機能やテストケースの洗い出しがしやすくなる。

業務フロー図

この項目を学習/実践しないと…

- どの業務範囲をアプリケーションとして実装したいのか分からない…
- 実装したい機能の抜け漏れが発生してしまう…
- アプリケーションの開発計画を部門内に展開する際、説明が難しい…

業務フロー図 って具体的にどういうもの？



<フローで用いる基本的な記号>

開始・終了 （端子）		業務フローの開始・終了を表す
プロセス （処理）		工程・処理を表す
判断 （条件分岐）		Yes/No条件等で フローが変化する場合に使用
矢印		処理や条件分岐の間をつなぐ記号

参考：
業務フロー図の書き方を0から丁寧に解説 記号や便利ツールも紹介【2024年11月24日】
https://product.strap.app/magazine/post/knowhow_workflow

バージョン管理

アプリケーションの変更履歴を適切に管理することで、
バグの発生時や誤った変更があった場合に迅速に対応できる。
“誰が、いつ、どのような変更をしたのかを明確に記録する”ことで、
問題の原因を特定しやすくする。

この項目を学習／実践することで…

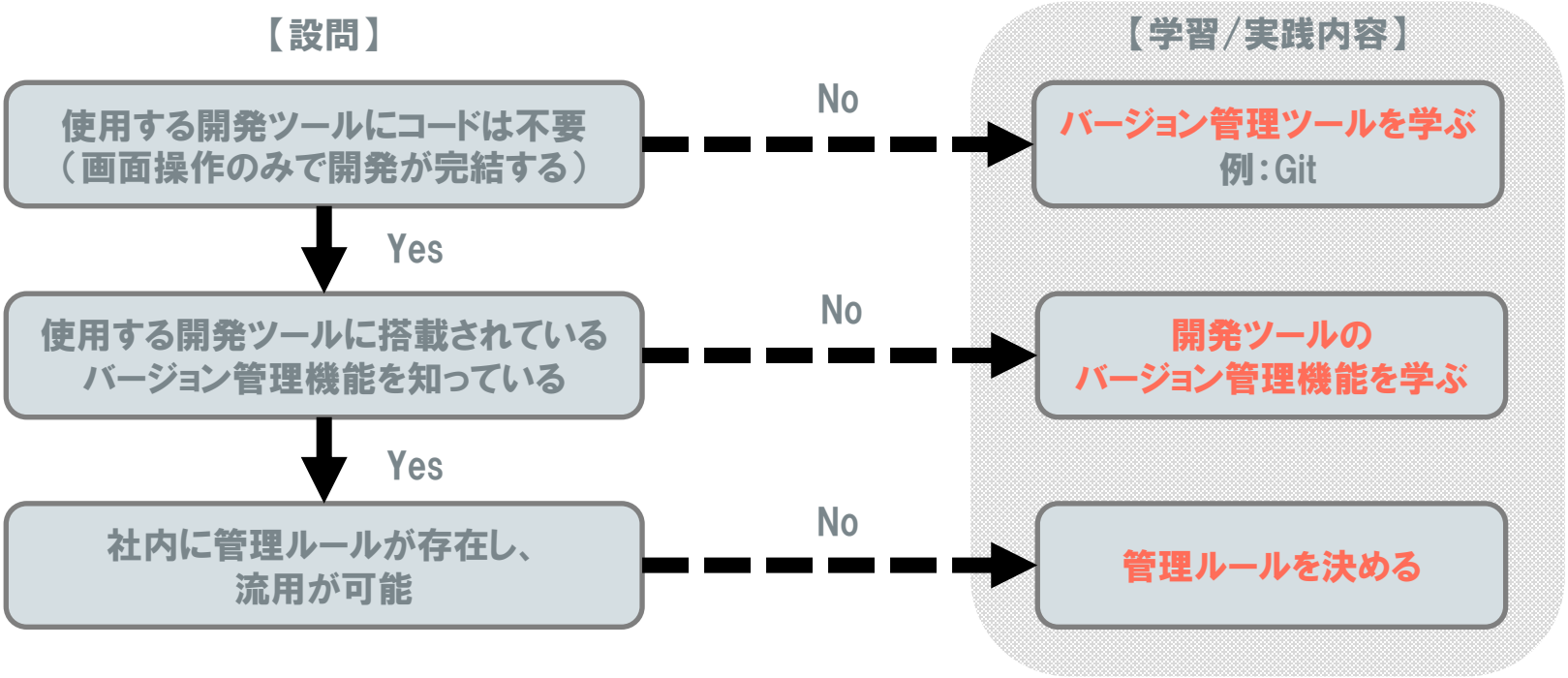
- 問題発生時の調査・手戻り作業工数を低減できる。
- 変更履歴から仕様変更の背景や経緯を追跡でき、
  アプリケーションの保守性が向上する。

バージョン管理

この項目を学習／実践しないと…

- 仕様変更が発生した際、修正箇所の特定が難しい…
- 複数人での開発の場合、誰がどこを変更したのか分からない…
- アプリケーションの最新リリース版を紛失してしまい、運用が破綻する…

バージョン管理 って何を学習/実践すればいい？



動作テスト

開発中／開発後に“アプリケーションに対して行うテスト”。
アプリケーションが正しく動作するか、
動作停止が発生しないかをテストケースを作成し確認する。

この項目を学習／実践することで…

- アプリケーション利用時の不具合防止
- 不具合による改修作業工数の低減
- アプリケーションの利用破綻の防止
  （業務に支障が出るような致命的なエラーを防ぐ）

動作テスト

この項目を学習/実践しないと…

- アプリケーションの何をテストすれば良いのか分からない…
- エラーによってシステムが停止してしまい業務に支障が出てしまう…
- 想定外の更新処理によって業務データが破壊されてしまう…

動作テスト って具体的に何をすればいいの？

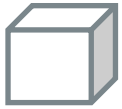
ノーコード/ローコードツールでの開発においては、最小限の要素として下記2点を確認する。

開発したアプリケーションが、

① 通常の動作に対して意図した通りの処理をしているか？

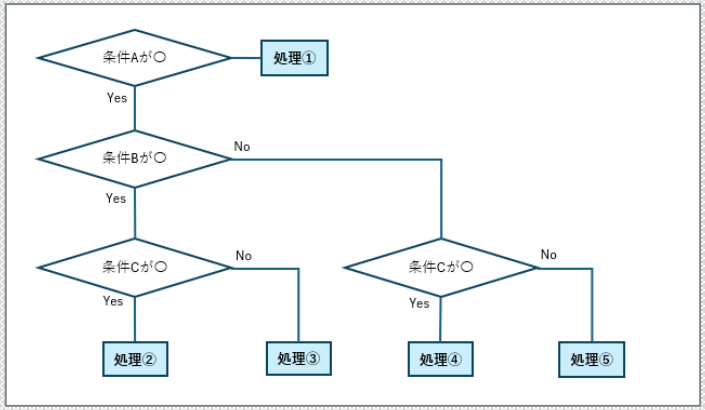
② 例外的な動作に対して重大なエラーを発生させないか？

① 通常の動作に対して意図した通りの処理をしているか？



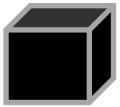
ホワイトボックステストと呼ばれる、アプリケーションの作り手側目線のテストを行う。
設計時に作成した業務フロー図や処理パターンを基に、**網羅的にテストケースを作成する。**

【例】
3つの条件A,B,Cを満たすかどうか（〇か×か）で処理が変わるシステムの場合、
下表の**8通り**のテストを行い正しい処理がなされるかを確認する。



No.	条件A	条件B	条件C
1	〇	〇	〇
2	〇	〇	×
3	〇	×	〇
4	〇	×	×
5	×	〇	〇
6	×	〇	×
7	×	×	〇
8	×	×	×

② 例外的な動作に対して重大なエラーを発生させないか？



ブラックボックステストと呼ばれる、アプリケーションの使用ユーザー目線でのテストを行う。
ユーザーとして可能な動作であれば、通常行うことのない動作等でもシステムが正常に動くかを確認する。

【例】
- ブラウザの戻るボタン押下時の処理
- 数値を入力するエリアに日本語などの文字列を入力する
- 入力必須項目を空欄のまま実行する
- ユーザー登録等を行う場合、同一文字列のユーザーを作成する

⇩

○ エラー表示等を行い、ユーザーに再操作等を求めていればテスト合格 （正常動作）

× 操作ができない状態が続く等、システムが停止してしまう場合はテスト不合格 （異常動作）

共通言語の確立

共通言語とは、複数の人が”同じ認識を持っている”用語を指す。
プロジェクト内で共通言語を確立することでメンバー間の認識齟齬を防ぎ、
円滑かつ品質の高いコミュニケーションが可能となる。

この項目を学習/実践することで…

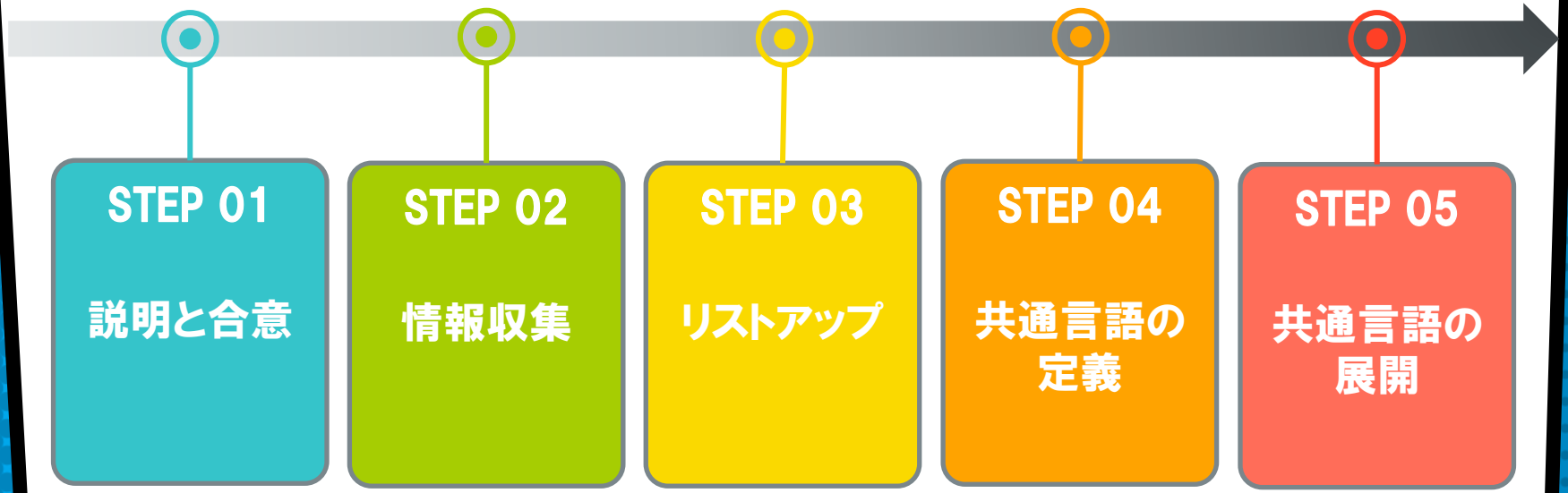
- 誤解による手戻りや遅延を防止できる。
- 円滑なコミュニケーションにより会議時間を短縮できる。
- 用語の統一により要件定義の品質が向上する。

共通言語の確立

この項目を学習/実践しないと…

- 用語に対する認識齟齬が発生してしまう…
- 用語の認識確認が頻繁に発生し、設計がなかなか進まない…
- 求めている機能と実装した機能が一致せず、手戻りが発生してしまう…

共通言語の確立 に必要な5つのステップとは？



共通言語の確立 に必要な5つのステップとは？

共通言語確立の必要性をプロジェクトメンバーに説明し、合意を得る。

関連部門の用語集や文章の収集を行う。

収集したものから、開発対象の業務フロー内に登場する専門用語をリストアップする。

リストアップした用語の意味を再定義し、共通言語として確立する。

共通言語が定義された用語集やガイドラインを作成し、プロジェクトメンバーへ展開する。

ツールを活用したコミュニケーション

“コミュニケーションツール”を活用することで、
チーム全体でスムーズな情報共有が実現できる。
計画の進捗状況の可視化と透明性を向上し、
より効率的なチームコラボレーションを目的とする。

この項目を学習/実践することで…

- 情報伝達や進捗管理がリアルタイムで可能になる
- 場所や時間に縛られない柔軟な情報共有が実現できる
- 過去の会話や決定事項を簡単に検索/参照できる
- ファイルやドキュメントの最新版を一元管理できる

ツールを活用したコミュニケーション

この項目を学習／実践しないと…

- 他部門との連携に時間が掛かる…
- 適切な議事録が無く、言った言わない論争が発生してしまう…
- 開発メンバーの進捗度が把握しづらい…

ツールを活用したコミュニケーション の代表例①

オンライン会議

在宅でも出張先でも。
移動時間ゼロで会議スタート！

代表的なツールのチュートリアル

Teams
https://support.microsoft.com/ja-jp/teams

Slack
https://slack.com/intl/ja-jp/help

ツールを活用したコミュニケーション の代表例②

タスク管理ツール

「あれ、どうなってるっけ？」はもう過去の話。
タスクの見える化で、期限管理もバッチリ！

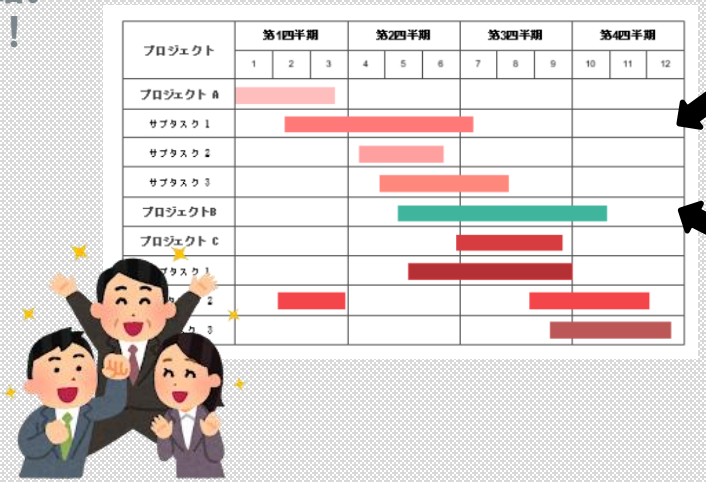
代表的なツールのチュートリアル

Trello
https://trello.com/ja

Asana
https://asana.com/ja

Project Canvas
https://www.rumix.co.jp/pc/

Wrike
https://www.wrike.com/ja/



ツールを活用したコミュニケーション の代表例③

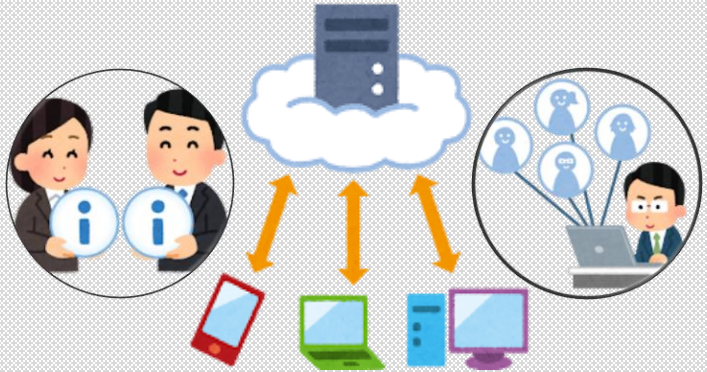
共有サーバで情報一元管理

もう添付ファイルの送信ラッシュとはサヨナラ！
最新版はいつでもここを見れば OK！

代表的なツールのチュートリアル

OneDrive
https://support.microsoft.com/ja-jp/onedrive

Box
https://www.box.com/ja-jp/customers



部門の統制力強化

“アプリケーション開発に携わる全てのメンバーが同じルールに従うこと”で、
アプリケーションの開発、使用、管理において一貫性が保たれ、
操作ミスや混乱を防ぐことができる。

この項目を学習／実践することで…

- 野良アプリ（組織が把握できていないアプリ）を防ぐ
- アプリケーションの品質を一定に保つことができる
- 明確なルールや手順書があることで新人教育が捗る
- ルールや手順が明確であることで、
  誰が何を行ったかを追跡しやすくなり、透明性が高まる

部門の統制力強化

この項目を学習/実践しないと…

- 組織が把握できていないアプリケーションが乱立してしまう…
- 開発フローが統一されておらず、後継者が育たない…
- アプリケーションのデザインや品質がバラバラ…

部門の統制力強化 をするためには何を決める必要がある？

アプリケーションの管理者を任命

開発・運用・セキュリティに関する、**部門内での最終判断を行う統括責任者**を定める。

リスク管理の一元化および統制力の強化を図る。

アプリケーションの作成フローを策定

企画、設計、開発、テスト、リリースにおける具体的な手順と、開発者と管理者の役割・責任を明確に定義する。

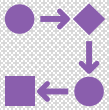
開発プロセスを標準化することで、属人化の防止および品質を一定に保つことを目的とする。

アプリケーションの管理簿作成

全社で開発されている**アプリケーションの基本情報（名称/目的/開発部門/開発日/管理者など）を一元管理する**データベースまたは台帳を作成する。

アプリケーションの棚卸しと活用状況を可視化し、管理を容易にする。

アプリケーションの雛形作成



標準化されたテンプレートを作成し、**デザイン/機能/セキュリティ要件を統一する**ことで、アプリケーションの品質と一貫性を確保する。

新規開発者の学習コストや諸々の開発工数の削減ができる。

部門の統制力強化 をするためには何を決める必要がある？

アプリケーションの作成権限の制限

開発ツールの権限設定機能や汎用的なアクセス制限を活用し、**適切なスキルと理解を持つメンバーのみにアプリ開発を許可する**。

セキュリティリスクの低減を図ると共に、不適切な開発を防止する。

アプリケーションの利用範囲の制限

アプリケーションが利用可能な部門、ユーザー、データアクセス権限を明確に定義し、不正利用を防止する。

データプライバシーの保護や、コンプライアンス遵守を目的とする。

アプリケーションの拡張機能の制限

サードパーティ製プラグインや高度な機能の使用を制限し、セキュリティリスクと機能の複雑性の増大を防止する。

システムの安定性、管理の容易性を確保する。

アプリケーションの使用方法/仕様の文書化

アプリケーションの目的/機能/操作手順/注意事項等を記載したマニュアルを作成し、ユーザーの適切な利用をサポートする。

ユーザーの利用支援だけでなく、文書化することで属人的な運用も防止できる。

【Power Automateの活用例】

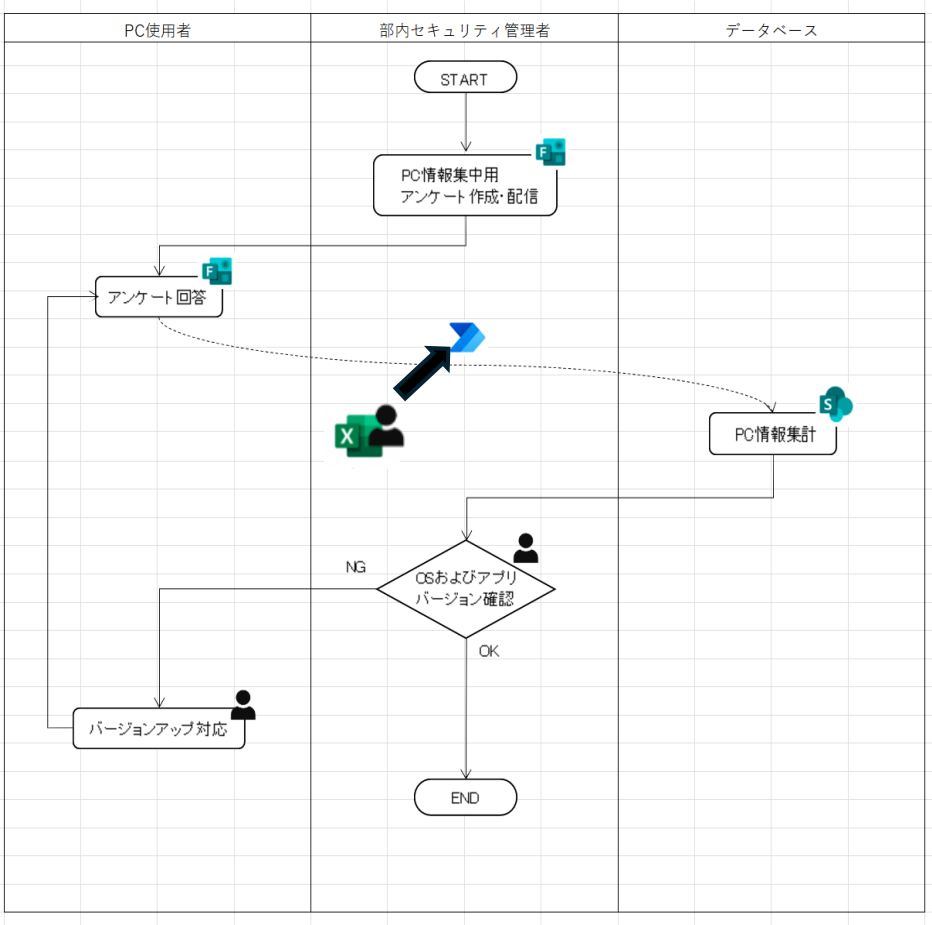
■システム概要
データ集計～結果公開を自動化

■システム化対象の業務課題
セキュリティ脆弱性を改善するため、
部内全員のOSおよびセキュリティソフトの状態を
定期的に調査(アンケート)している。

部員全員のアンケート結果の収集にExcelを
使用しており、集計と部員公開は手作業で行っていた。

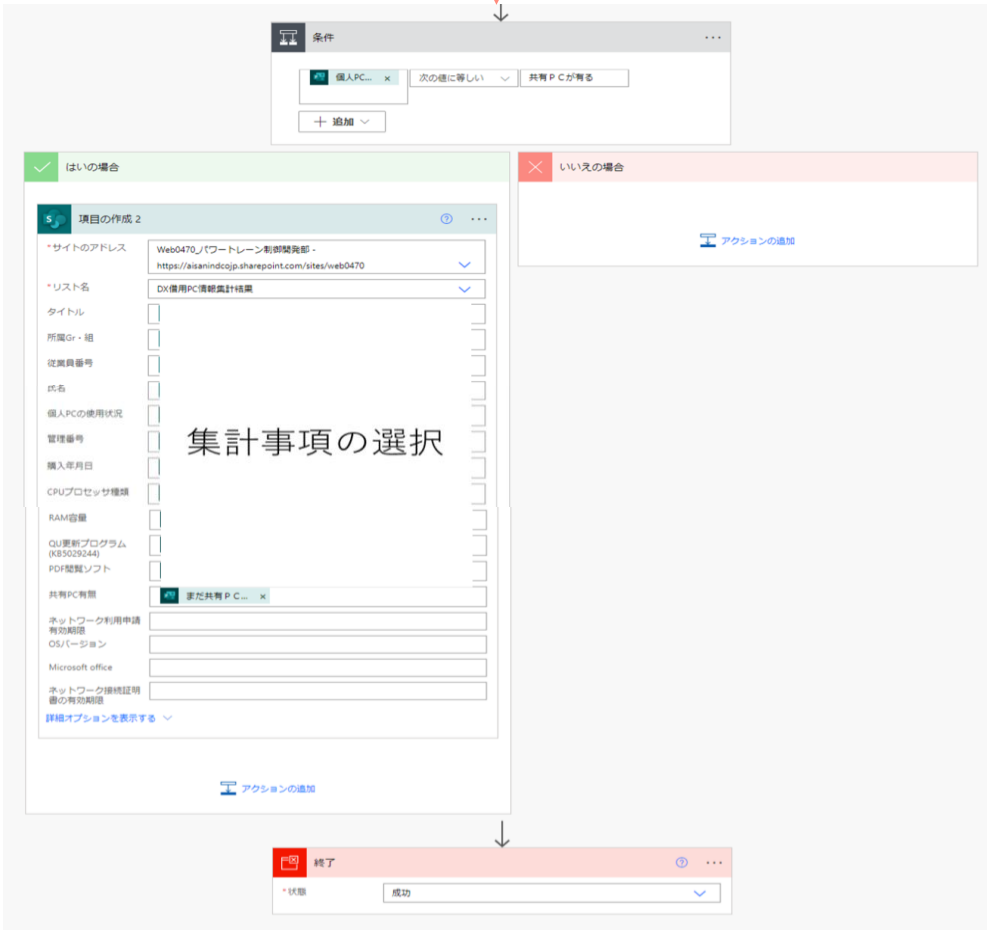
■開発者コメント
PowerAutomateでFormsとSharePointを
連携させることで、データ整理作業および
部員公開を自動化できた。

さらに機能を追加することで、OSと
セキュリティソフトの状態が最新になっていない
部員に対してアラートを出すこともできる。



【Power Automate処理フロー】

付録① :Power Automate活用例(2/2)



要件定義書

不具合対策管理システム

マンダラレンジャー株式会社
IT開発戦士部
本郷 隼人
作成日：20YY/MM/dd　　第１版

改訂履歴

版数	作成日	改定箇所	内容	作成者	承認
1	2025/MM/DD	-	新規作成		

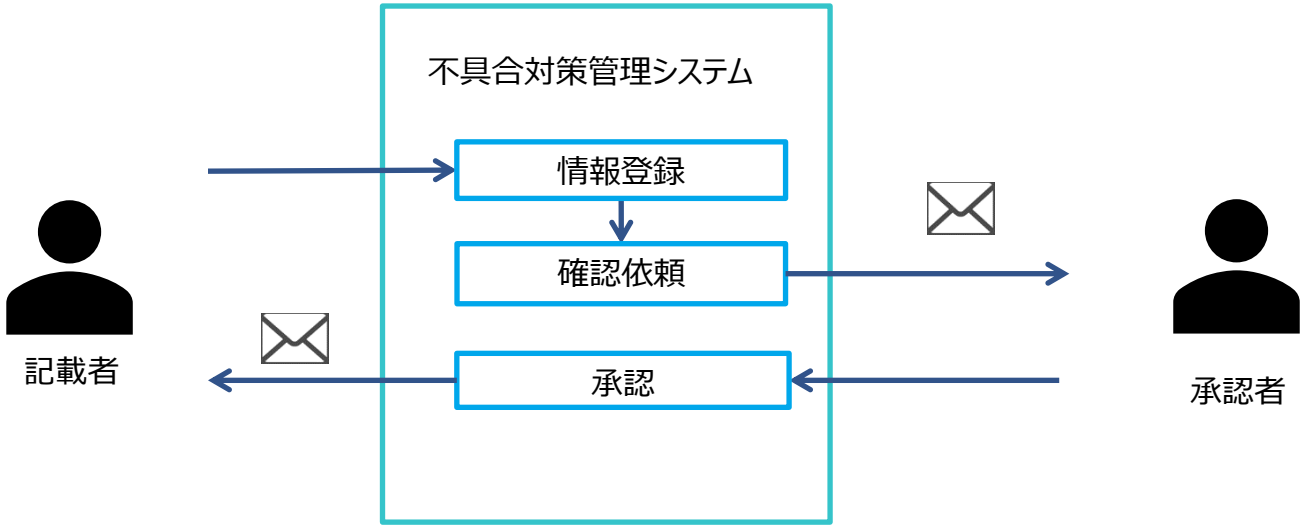
目次

1.背景目的

- 製品の不具合発生時に顧客へ対策報告を行っているが、社内へ共有が行われていない。
- 社内共有のフォーマットが決められていないため内容にバラつきがある
- 手書き書面での管理の為、内容の確認/分析に時間がかかる
- 顧客毎や案件毎など様々な項目で不具合内容の検索が出来る
- 不具合の内容を項目毎に分析する事ができる
- 不具合対策の内容を申請し、内容を確認して承認する流れをアプリで完結できる

2.運用概念図

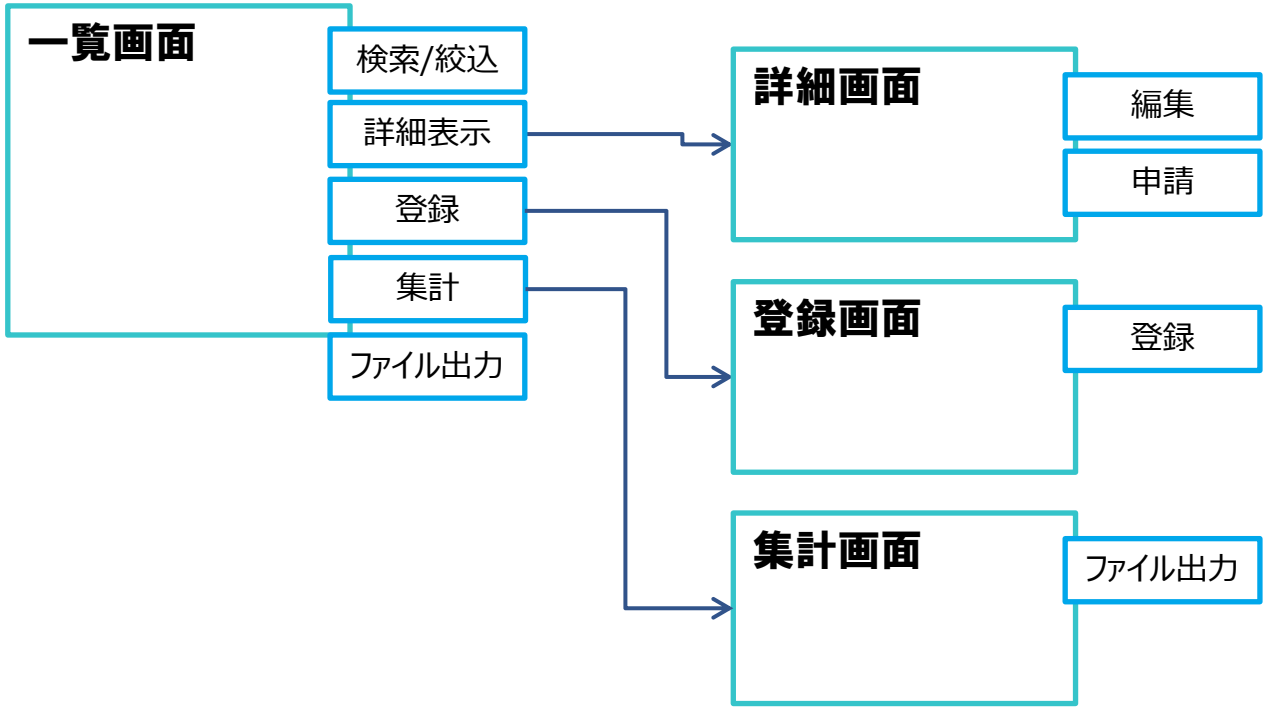
記載者は当システムにて不具合対策情報を登録する。
承認依頼を出すことが可能。



3.主要機能一覧

実装機能	
一覧画面	一覧表示
	詳細閲覧
	検索/絞込
	ファイル出力
	登録
詳細画面	申請
	更新
登録画面	登録
集計画面	ファイル出力

4.画面遷移図

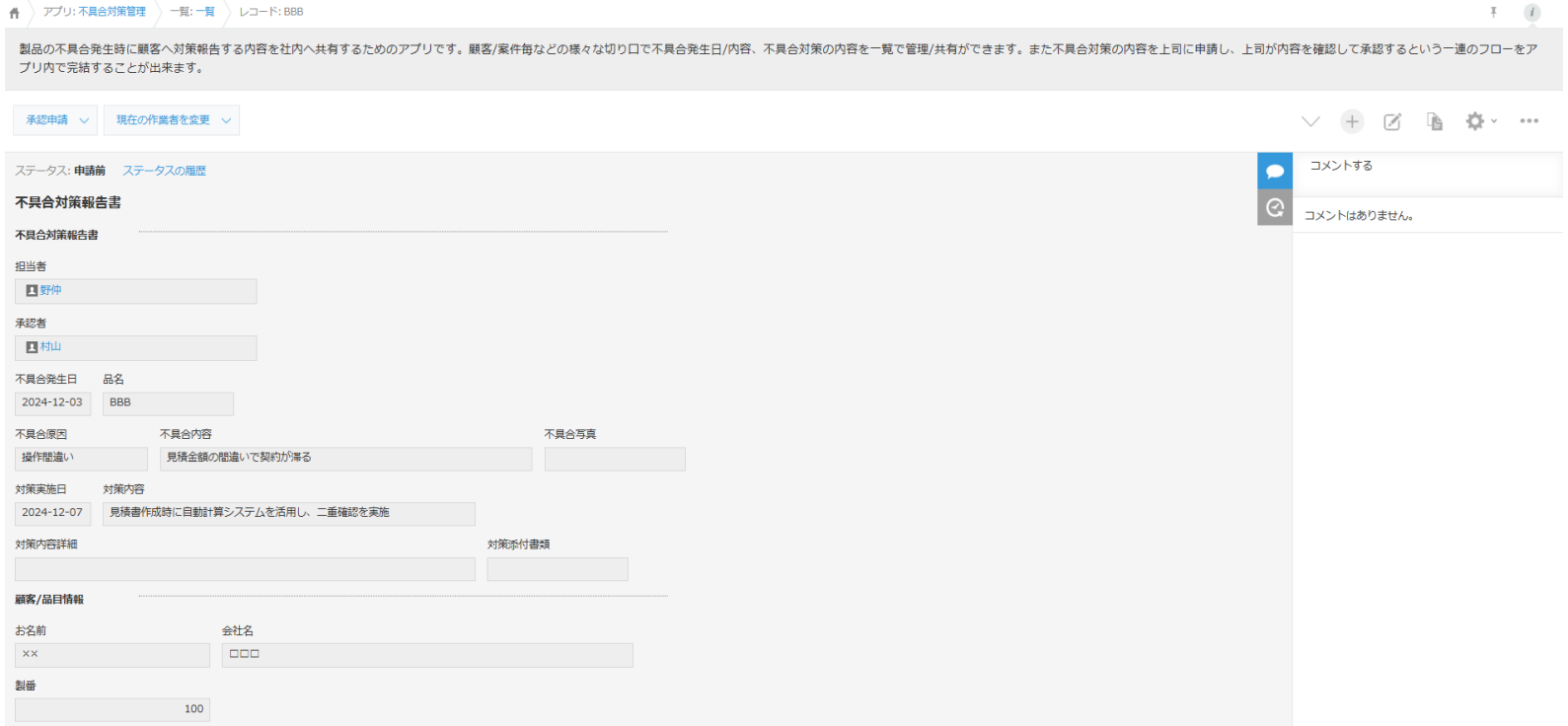


5.画面イメージ

一覧画面　　登録されている不具合情報を一覧で表示する画面
検索条件に合致した情報を絞り込み表示する事が可能

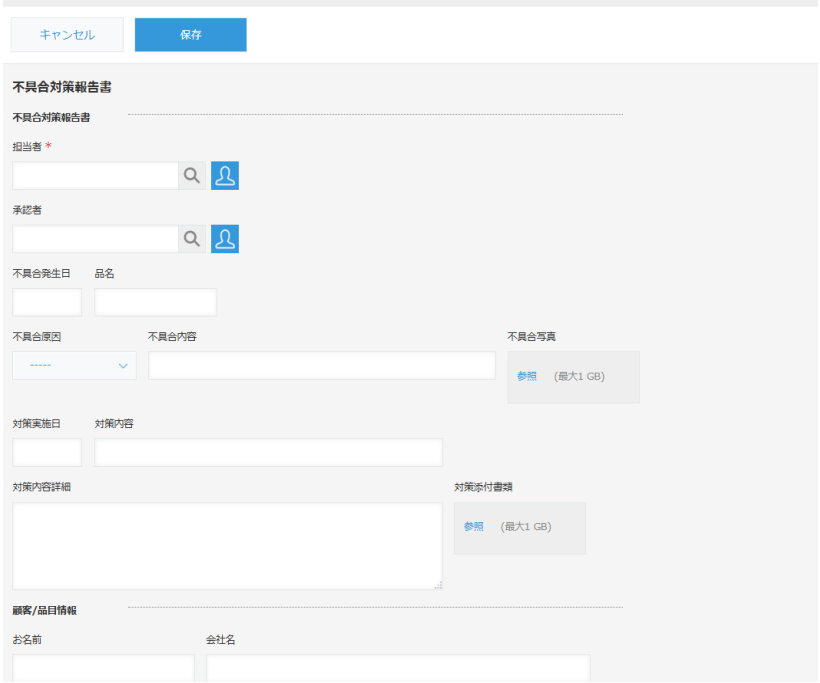
5.画面イメージ

詳細画面　　登録されている不具合情報の詳細を確認する画面
承認依頼を流す事が可能



5.画面イメージ

登録画面　　不具合対策報告を登録する画面



5.画面イメージ

集計画面　　登録された情報を集計し表示する画面
グラフの種類や集計方法が選択可能

付録③ ：マンダラチャート テンプレート

■概要
・3×3を9つ組み合せた図（合計81マス）
・中心から派生した目標や計画を整理するF/W

■書き方
・中心に大目標を記載
・周りに中目標を記載
・中目標を新しい中心として8つの小目標を記載
・左上から時計回りに優先順位順に記載

■使い方
・実行しやすい物から実施
・達成した小目標にチェック
・周りの小項目全て達成した場合中目標も達成とする

■注意点
・具体的な目標とする
・実現可能な目標とする
・重複は極力避ける
・定期的に見直しをする
・継続できる目標とする