要件定義

要件定義とは？

- システム開発における最初のステップ
- クライアントのニーズを明確にし、開発チームと共有するプロセス
- 「**何を作るか**」を決める重要な工程

要件定義の重要性

- 開発の土台（ここが曖昧だと後の工程で大きな手戻りが発生）
- 開発チームと顧客の認識を統一
- コスト・納期を管理しやすくなる

要件定義の流れ

４つの基本プロセス

- 要求の抽出
- 要求の分析
- 要求の仕様化
- 要求の確認

要求の抽出

- **ステークホルダー**，文書，知識などから問題・要求を見つける．
- 典型的な手順
  - 目的の明確化：ビジネスの目標や解決すべき問題を抽出
  - 構造・業務・制約条件：対象領域（ドメイン）の分析
  - ステークホルダーの定義：対象ドメインのステークホルダーを抽出
  - 要求の抽出と明確化：ステークホルダーの要求・問題を網羅的に抽出
- 手法：資料収集，アンケート，インタビュー，観察，業務分析，ブレインストーミングなど
- 注意すること
  - ドメインの設定とステークホルダーの設定が重要！
  - 要求を網羅すること（隠れた要求：非機能要求）

要求の種類

1. ビジネス要求（Business Requirements）

- 定義：組織や顧客がビジネス目標を達成するために必要とする要求
- 例：業務効率を20%向上させたい、新しい市場で顧客対応を自動化したい

2. ステークホルダー要求（Stakeholder Requirements）

- 定義：ビジネス要求を満たすために、ステークホルダーがシステムに求める内容
- 例：営業部はタブレット端末から顧客情報を即座に検索したい

3. システム要求（System Requirements）

- 定義：ステークホルダー要求を満たすために、システムに課される具体的な要求
- 備考：ここから機能要求と非機能要求に細分化される

システム要求の分類

3.1 機能要求（Functional Requirements）

- 定義：システムが「何をするか」に関する要求（業務機能、入力・出力など）
- 例：ユーザがログインできる、在庫を検索できる、レポートを出力する

3.2 非機能要求（Non-Functional Requirements）

- 定義：システムの「性質」に関する要求（品質特性）
  - システムの品質・運用面の制約
  - パフォーマンス: 「同時アクセス1,000人でも快適に動作」
  - セキュリティ: 「パスワードはハッシュ化して保存」
  - 運用性: 「クラウド環境で24時間365日稼働」
- 例：応答時間は2秒以内、99.99%の可用性、暗号化通信を使う

要求の分析

- 抽出した要求を整理する
- 要求の**優先順位**を付ける

観点	説明
必要性	本当に必要な要求か？（「あったら便利」レベルか、「必須」か）
類似性	同じ要求が複数出ていないか？統合できないか？
一貫性	他の要求と矛盾していないか？
完全性	抜け漏れがないか？目的に対するカバレッジは？
実現可能性	技術的・コスト的・時間的に実現できるか？

要求の仕様化

要求分析結果をドキュメント化（仕様化）する

手法	説明
要求図	SysMLなどを用いて要求の階層構造や依存関係を視覚的に示す
ユーザストーリー	アジャイル開発向けの形式で、ユーザの視点から要求を記述する
ユースケース図	アクターとシステム間のインタラクションを明示する
ユースケース記述	ステップ・前提条件・例外処理などを文章で詳細に記述する

観点	説明
正確性	曖昧語（例：「早く」「簡単に」など）を排除しているか？
一貫性	要求間で矛盾がないか？仕様がドメインルールと整合しているか？
最小性	重複記述がないか？必要最低限か？

要求の確認

- 仕様をもとにユーザ・ステークホルダーとの合意を行う

手法	目的・内容
レビュー	関係者との仕様文書の確認を行い、誤解や抜け漏れを防ぐ
プロトタイピング	動作イメージを示して、ユーザの誤解を防ぐ
モデル化	UMLやDFDなどで動作イメージを可視化し、理解度を高める

注意すること

- 要求内容がユーザやステークホルダーが考える内容とあっているか？
- 要求の優先順位がユーザやステークホルダーが思っている優先順位とあっているか？
- 要求の取捨選択：要求の範囲とコストの折り合い
- 合意形成：承認者・責任者が誰かを明確にし記録を残す
- トレーサビリティ：要求が後続工程（設計・テスト）と対応していることを確認

典型的な失敗事例

関係者の認識のズレ

- 事例: 業務担当者と開発者の間で「顧客情報管理機能」の定義が異なっていた
- 業務担当者は「営業履歴や問い合わせ内容まで管理したい」と考えていた
- 開発者は「名前・住所・電話番号」程度の顧客情報を想定
- 結果: 納品後に「使い物にならない」と判断され、再開発コストが発生

非機能要件の見落とし

- 事例: ユーザー数やアクセス負荷に関する非機能要件を定義せず、実装もされない
- 結果: システムは少人数では問題なく動作．本番運用ではアクセス集中によりサーバーダウンが頻発

ユーザーの声を聞かずに要件を決定

- 事例: 現場の業務を知らない管理職が主導して要件をまとめた
- 結果: 現場の作業フローと合わず、実際には全く使われないシステムとなった

要件の変更管理が不十分

- 事例: 要件定義後に顧客から新たな要望が次々に出され、それを都度対応
- 結果: **スコープ**が膨張（スコープクリープ）し、納期もコストも大幅にオーバー

※スコープ：プロジェクトで実施する作業の範囲とその境界を定めたもの

仕様化

ユーザーストーリー

- アジャイル開発で使われる軽量な要求記述の形式
- ユーザーの視点で「何をしたいか」「なぜそれが必要か」をシンプルに表現する
- 詳細な仕様ではなく、**共通理解のきっかけ（会話の出発点）** を作るためのもの

書き方（テンプレート）

「○○として、△△ができるように、□□をしたい」

- ○○：役割（誰が）
- △△：目的（何のために）
- □□：行動（何をしたいか）

例：「一般ユーザーとして、ログイン状態を保持できるように、自動ログイン機能を使いたい」
目的：毎回ログインせずにサービスをすぐ使えるようにしたい

エレベーターピッチ（背景説明）

- **なぜこの機能が必要なのか？** を1〜2文で補足

例：「ユーザーがログインした状態を維持することで、毎回のログイン操作を省略でき、利便性が高まる。」

ユースケース

- システムの機能をユーザーの視点で捉える
- ユーザーがシステムを使って達成したい目標を明確にする
- ユースケース図：システムの機能とユーザーの関係を視覚的に表現
- ユースケース記述：各ユースケースの詳細な説明（前提条件、基本フロー、代替フローなど）

項目	ユーザーストーリー	エレベーターピッチ	ユースケース
目的	ユーザーのニーズを簡潔に表現	機能の価値を短時間で伝える	システムとユーザーのやりとりを明確化
形式	「○○として、△△ができるように、□□をしたい」	「この機能によって○○な課題が解決され、△△な価値がある」	ユースケース図（UML）＋ユースケース記述（テキスト）
内容	ユーザーの目的と行動	なぜその機能が重要か・役立つか	アクターとシステムの具体的なインタラクション
主な利用場面	アジャイル開発の計画や優先順位付け	経営層や他者への機能のプレゼン、合意形成	要件分析や設計時の機能の整理・確認

まとめ

- 要件定義はシステム開発の出発点かつ成功の鍵
- 要件は 機能要件 と 非機能要件 に分類して整理する
- ユーザー視点を大切にし、ユーザーストーリーやユースケースを活用する
- 要件の 抽出 → 分析 → 仕様化 → 確認 を通じて、ステークホルダーと合意形成を行う