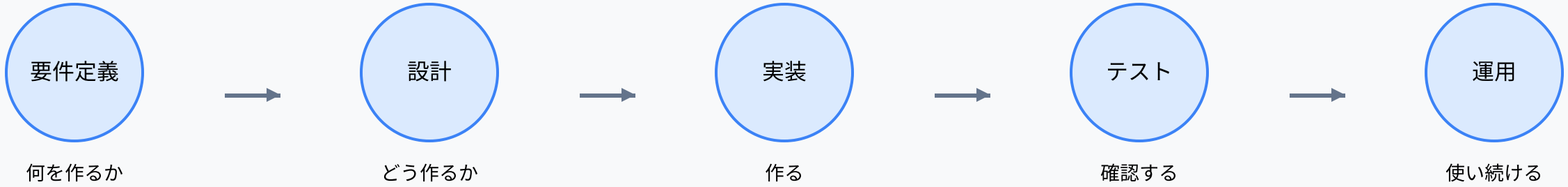
システム開発における開発とインフラの役割

工程ごとの仕事の違いをわかりやすく整理

システム開発工程の全体像



開発チームの主な役割

- 機能・画面の実装
- データベース設計と構築
- APIの設計と実装
- プログラムの品質確保

インフラチームの主な役割

- サーバー環境の構築
- ネットワーク設計
- セキュリティ対策
- システムの安定運用

各工程ごとの「開発」と「インフラ」の詳細な役割と連携については、次のスライドで詳しく解説します。

要件定義：役割の違い

開発チーム

- 機能要件の定義：画面・機能一覧、操作フローの作成
- API要件の定義：外部連携の方式や仕様を決定
- データベース要件：必要なデータ項目、関連性を整理
- 画面要件：UIの基本構想、操作性の要件を定義

インフラチーム

- サーバー要件：必要なサーバー構成、台数を検討
- クラウド環境：最適なクラウドサービス選定（AWS/GCP等）
- セキュリティ要件：認証方式、暗号化、アクセス制御を定義
- 性能要件：レスポンス時間、同時アクセス数の目標設定

具体例

要件：「ユーザーが**10万人同時利用**できるシステム」と決定

→

インフラチームの対応：高負荷に耐えられるサーバー構成、スケーラブルなクラウド環境、負荷分散の仕組みを検討

要件定義では開発とインフラが密に連携し、ビジネス要求を技術的に実現可能な要件へと落とし込みます。それぞれの視点から要件を洗い出し、システム全体の青写真を描きます。

設計：役割の違い

開発チーム（設計）

- 画面設計：画面レイアウト、画面遷移、UI/UXの詳細設計
- API仕様設計：RESTful APIのエンドポイント、パラメータ、戻り値の定義
- DB設計：ER図作成、テーブル定義、インデックス設計
- 処理フロー：ビジネスロジックの実装方針、アルゴリズム設計

インフラチーム（設計）

- サーバー構成設計：Webサーバー、APサーバー、DBサーバーの構成
- ネットワーク設計：サブネット設計、ロードバランサー、CDN構成
- セキュリティ設計：ファイアウォール、WAF、証明書、暗号化
- 冗長化・監視設計：マルチAZ構成、バックアップ方式、モニタリング

「アプリの設計図」と「システム基盤の設計図」の対比

アプリの設計図

- ✓ 画面モックアップ・ワイヤーフレーム
- ✓ クラス設計・コンポーネント図
- ✓ データモデル・ER図
- ✓ API仕様書

→ユーザーから見える「機能」の設計

システム基盤の設計図

- ✓ インフラ構成図（クラウドアーキテクチャ）
- ✓ ネットワーク構成図
- ✓ セキュリティ設計書
- ✓ 運用監視設計書

→ユーザーからは見えない「土台」の設計

設計段階では、開発とインフラが互いの制約条件を理解し合い、整合性のある全体設計を構築することが重要です。開発はユーザー体験を重視し、インフラは安定性・セキュリティを重視しますが、両者が連携することで理想的なシステムが実現します。

実装・構築（開発）

開発チームが担当する実装作業

フロントエンド開発

- 画面UI実装（HTML/CSS/JavaScript）
- React/Vue/Angular等のフレームワーク活用
- レスポンシブデザイン対応
- API連携処理の実装

バックエンド開発

- API開発（REST/GraphQL）
- ビジネスロジックの実装
- セキュリティ対策（認証・認可）
- 外部サービス連携処理

データベース構築

- DBスキーマ作成・テーブル設計
- インデックス最適化
- マイグレーションスクリプト作成
- ORM・クエリ実装

```
// フロントエンド実装例（React）
function UserList() {
  const [users, setUsers] = useState([]);
  useEffect(() => {
    fetchUsers()
      .then(data => setUsers(data));
  }, []);
}
```

```
-- データベーススキーマ例
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  email VARCHAR(100) UNIQUE,
  created_at TIMESTAMP
);
```

単体テストコード作成

- 各機能・モジュールの動作検証（Jest, JUnit等）
- 入力値検証、エラーハンドリングのテスト
- テストの自動化によるリグレッション防止

重要ポイント

この工程で開発チームは「**機能**」を実際に作り込みます。設計工程で決めた仕様に基づき、ユーザーが実際に使える画面や機能を形にしていきます。コーディング作業だけでなく、品質を確保するための単体テスト実装も重要な役割です。

実装・構築（インフラ）

インフラチームが担う作業

開発チームが機能を作り込む一方で、インフラチームは安定したシステム基盤（土台）を準備します

サーバー構築

- クラウドプラットフォーム活用（AWS、GCP、Azure）
- 仮想サーバー（EC2、GCE）のセットアップ
- データベース（RDS、CloudSQL）の構築
- ネットワーク・セキュリティグループの設定

コンテナ環境

- Dockerコンテナの構築・設定
- Kubernetesによるオーケストレーション
- コンテナレジストリ管理
- マイクロサービス基盤の整備

IaCによる自動化

- Infrastructure as Codeの実践
- Terraform/CloudFormationによるリソース定義
- AnsibleやChefによる構成管理
- 環境構築の再現性確保

開発と同じタイミングで「土台」を整備

アプリケーション開発と並行して基盤整備を進めることで、テストフェーズでの連携がスムーズになります。近年はDevOpsの浸透により、開発チームとインフラチームの協業がより重要になっています。

テストにおける役割

開発チーム

- 機能テスト：各機能が要件通りに動作するかを検証
  例：ログイン処理、データ登録、検索処理など
- 結合テスト：複数の機能を連携させて処理が正常に動くか検証
  例：ユーザー登録から決済までの一連の流れ
- システムテスト：システム全体として要件を満たしているか検証
  例：エンドツーエンドのユーザーシナリオ検証

「正しく動くか」の検証

インフラチーム

- 負荷テスト：高負荷時のシステム挙動を検証
  例：同時アクセス数増加時のレスポンス測定
- 障害復旧テスト：障害発生時の回復手順を検証
  例：サーバーダウン時の自動フェイルオーバー
- セキュリティテスト：脆弱性や不正アクセスへの対応を検証
  例：ペネトレーションテスト、脆弱性スキャン

「安定して動くか」の検証

開発とインフラの両輪によるテスト

機能性：ユーザーが求める機能を正しく提供できるか

＋

＝

非機能性：安定稼働、セキュリティ、パフォーマンスが確保できるか

両面からの品質保証により、ユーザーに「価値あるシステム体験」を提供

テスト連携のポイント

- 各テストの結果を互いに共有し、問題解決に協力
- 負荷テストで見つかった問題がアプリケーションのチューニングにフィードバック
- セキュリティテストの結果が機能やコードの修正に反映

リリース・運用（開発視点）

デプロイ
コードの本番環境への反映

バグ修正
障害・不具合の特定と解消

機能改善
新機能追加・既存機能の強化

改善サイクル
継続的な開発と改善の循環

デプロイ業務

- 本番環境へのコード反映・適用作業
- デプロイスクリプト・自動化ツールの整備
- ダウンタイムを最小化するリリース調整
- デプロイ後の動作確認とロールバック準備

バグ修正

- ユーザーからの不具合報告の調査・対応
- エラーログの分析と根本原因の特定
- ホットフィックス対応と緊急デプロイ
- 修正内容のテストと検証の実施

機能改善

- ユーザーフィードバックに基づく機能拡張
- パフォーマンス最適化・レスポンス改善
- UI/UXの改善・ユーザビリティの向上
- ビジネス要求に応じた新機能の開発

サービス改善サイクル

- アクセス解析・ユーザー行動の分析
- A/Bテストによる機能検証と最適化
- 定期的なスプリントによる継続的開発
- KPI達成に向けたプロダクト進化の推進

💡 開発チームは、システムのリリース後も**「サービス品質の維持向上」**を担う重要な役割を担っています。ユーザーに価値を提供し続けるために、継続的な改善サイクルを回し、常に進化するシステムを実現していきます。

リリース・運用（インフラ視点）

インフラチームが担う運用業務

システムのリリース後、インフラチームは安定的な稼働環境の維持と安全性の確保を担当します。継続的なモニタリングと迅速な対応が重要です。

システム監視

- CPU/メモリ使用率の常時監視
- ディスク容量の監視と警告設定
- ネットワークトラフィック分析
- アプリケーションログの監視
- アラート通知の設定と対応

ツール例： Prometheus、Grafana、Datadog

障害対応

- インシデント管理と対応フロー整備
- 障害の切り分けと原因特定
- 復旧対応と影響範囲の最小化
- 再発防止策の実施
- 障害報告と改善提案

重要点： 障害対応手順書の整備とフェイルオーバー対策

バックアップ・セキュリティ

- 定期的なデータバックアップの実施
- リストア手順の整備と検証
- セキュリティパッチの適用計画
- OS・ミドルウェアの脆弱性対策
- 定期的なセキュリティ監査

ポイント： パッチ適用による影響評価とテスト

安定稼働を守る役割

予防的対応：監視体制の強化と自動復旧の仕組みにより、障害発生リスクを低減

事後対応：障害発生時の迅速な検知・対応によりダウンタイムを最小化

運用フェーズでのインフラチームの役割は「縁の下の力持ち」。ユーザーには見えにくいが、システムの信頼性を支える重要な役割を担っています。

まとめ

工程ごとの明確な役割分担

開発 = 機能

- ユーザーが使う機能の実現
- 画面・UI/UXの構築
- データ処理ロジックの実装
- サービス品質の向上

インフラ = 土台

- システムの安定稼働基盤
- セキュリティの確保
- パフォーマンスの最適化
- スケーラビリティの実現

両者の連携でシステム価値が最大化

開発とインフラの密な協力により、「正しく動く」かつ「安定して動く」高品質なシステムが実現します。一方だけでは成立しない、車の両輪の関係です。

変化するトレンド：DevOpsの台頭

最近のDevOpsやSREなどのモダンな開発手法では、開発とインフラの境界が曖昧になりつつあります。これにより、より迅速なデリバリーと高い運用効率が実現できます。

継続的インテグレーション + 継続的デリバリー + インフラ自動