# システム設計

## 共通

2023-04-20

止まった時の事も考える（メンテ）

## 要件定義

### 機能要件

操作方法、画面表示、データ種類、処理内容、出力帳簿の形式、など。

### 非機能要件

可用性、性能・拡張性、運用・保守性、移行性、セキュリティ、システム環境・エコロジー　の６つの観点

具体例

・障害発生からの３時間以内での復旧

・ボタン押下３秒後以内に結果を表示

参考：<https://www.ipa.go.jp/archive/digital/iot-en-ci/jyouryuu/hikinou/ent03-b.html>

## マイクロサービス

メリット

ある機能に障害が発生したりアップデートした場合に他の機能への影響を最小限にできる

機能を別のサービスに流用することが容易になるので開発コストの削減に繋がる

課題解決型

# 問題解決手法

## 深さ優先探索

試行錯誤を繰り返しながら、条件を満たす解に到達する方法。

場合分けを行い深さ優先で探索し、解が見つからなければ１つ前の場合分けの状態に逆戻りする。

# 重要な事まとめ

## 全体

・プロジェクト開始前に課題をはっきりとさせる事が重要。

一般的な流れとしては：目的を明確化 ⇒ 現状を把握 ⇒ 課題を洗い出す。

・組織として、「やろうと思えば外に依頼せずに出来る」状態を保つ。

・必要以上の開発を行わない

ユーザー部門の思い込みによって、過剰な要求になっていないか。

・扱うソフト、機器の種類はなるべく少なく。

・新しすぎるものは避ける

・考えやアイディアの変更履歴を残す。（なぜ変えたのか）

## ドキュメント関連

・ドキュメントを作り過ぎない

作りこみすぎもまたドキュメントの品質を落とす。（メンテが大変なので）

・必要最低限の情報のみ残す

管理に時間がかからないようにする。

## アジャイルの場合

概要

仕様が決まっていない為変更が発生しやすく、機能単位で小さくすばやく開発を繰り返していく。

小規模システム向き。

・できるだけ早くユーザーに操作してもらう。

## ウォーターフォールの場合

概要

固まっている仕様に対し、上から下へ工程を後戻りしないで開発を進める。

大規模システム向き。

・マイルストーン（遅延が許されない節目となる工程）を明確に定めるのが重要

## 注文関連

・見積を「レイヤー」ごとに分解する

・徹底的に比較検討する

## Well-Architected Framework

### 概要

クラウド上、特にAWS利用における、適切な設計を可能にするための推奨事項が記載されている。

<https://aws.amazon.com/jp/architecture/well-architected/>

「運用上の優秀性」「セキュリティ」「信頼性」「パフォーマンス効率」「コスト最適化」

### 一般的な設計原則

・必要キャパシティーの推測が不要に

・本稼働スケールでシステムをテストする

・自動化によってアーキテクチャでの実験を容易にする

・発展するアーキテクチャが可能に

・データに基づいてアーキテクチャを進化させる

・ゲームデーを利用して改善する

### 運用上の優秀性

運用をコードとして実行する

運用手順をコード化することでコードをトリガーとしてイベントが自動的に実行されます。その結果、ヒューマンミスが抑制されシステムが効率的に運用されます。

小規模かつ可逆的な変更を頻繁に行う

変更は失敗した場合に元に戻すことができるように、小規模で行います。可能であれば、顧客には影響がないようにします。

運用手順を定期的に改善する

運用手順が最適であるかどうかを常に考えます。運用手順は定期的に改善できるように計画を立て、決まった日程ですべての運用手順が効果的に運用され、チームメンバーが運用手順を十分に理解していることを確認します。

障害を予想する

障害が考えられる原因を取り除いたり軽減できるように、原因を特定するための演習を実施します。具体的には障害が起こる予測シナリオに基づいてテストを実行してその影響を検証します。

障害に対する対応手順をテストして、手順が効果的であることを確認し、チームメンバーが対応手順を十分に理解していることを確認します。障害のシミュレーションに対してチームメンバーが対応できるかどうかを確認する日を定期的に設定します。

運用上のすべての障害から学ぶ

運用上のすべてのイベントと障害から教訓を学び、改善を促進し、チーム間だけでなく組織全体で教訓を共有します。

### セキュリティ

強力なアイデンティティ基盤の実装

アイデンティティ管理を一元化して強力なアイデンティティ基盤を実装します。

トレーサビリティの実現

使用環境に対して、リアルタイムで監視、アラート、トレーサビリティを実現します。ログとメトリクスの収集をシステムに統合して、自動的に調査します。

全レイヤーでセキュリティを適用する

複数のセキュリティコントロールを使用して深層防御アプローチを適用します

セキュリティのベストプラクティスを自動化する

自動化されたソフトウェアベースのセキュリティメカニズムにより、より速く、より費用対効果の高いスケールが可能になります。

伝送中および保管中のデータの保護

データを機密性レベルに分類し、暗号化、トークン分割などによって伝送中および保管中のデータの保護を行います。

データに人の手を入れない

データに直接アクセスしたりデータを手動で処理したりする必要を減らし、機密性の高いデータを扱う際の誤処理、変更、ヒューマンエラーのリスクを軽減します。

セキュリティイベントに備える

組織の要件に合わせたインシデント管理および調査によってインシデントに備えます。インシデント対応シミュレーションを実行し、検出、調査、復旧のスピードを上げます。

### 信頼性

障害から自動的に復旧する

障害を発見するために主要指標をモニタリングし、しきい値を超えた場合にオートメーションをトリガーにします。これにより自動的に障害を通知および追跡することができ、障害を回避または修正する復旧プロセスも自動化できます。より複雑な自動化を使用すると、障害が発生する前に修正を予期できます。

復旧手順をテストする

クラウドでは、どのようにシステム障害が発生するかをテストでき、復旧の手順も検証できます。このアプローチでは、実際の障害シナリオが発生する前にテストおよび修正できる障害経路が公開されるため、リスクが軽減されます。

水平方向にスケールしてワークロード全体の可用性を高める

1つの大規模なリソースを複数の小規模なリソースに置き換えることで、単一の障害がワークロード全体に与える影響を軽減します。

キャパシティーを推測することをやめる

クラウドでは、需要と使用率をモニタリングし、リソースの追加と削除を自動化することで、最適なレベルを維持できます。

オートメーションで変更を管理する

インフラストラクチャに対する変更は、オートメーションを使用して実行する必要があります。

### パフォーマンス効率

最新テクノロジーの標準化

クラウドベンダーに委託して最新のテクノロジーを標準化できるようにします。専門知識が必要な高度で複雑なテクノロジーでも、クラウドではITチームメンバーが専門的に機械学習することなく、サービスとして使用できるかたちで提供されます。そうすることで、ITチームメンバーのリソースをテクノロジーの管理ではなく、製品の開発に使うことができるようになります。

わずか数分でグローバル展開する

AWSでの作業は、最小限のコストで、遅延することなく使うことができます。AWSのサービスセンターは世界各地にあるので、世界中で快適に使用できます。

サーバーレスアーキテクチャを使用する

サーバーレスアーキテクチャを使用することで、システムに必要な容量を、システム稼働時のみのコストで利用する事ができるため、サーバー容量を意識せず必要なサービスを利用でき、コストも削減できます。

より頻繁に実験する

仮想的で自動化できるリソースを使うことで、異なるタイプの設定を使用した比較テストをより頻繁に実行できます。

システムに対する精通の程度を考慮する

クラウドサービスの使用法を理解し、常にワークロードの目標に最適なテクノロジーアプローチを使用します。

### コスト最適化

クラウド財務管理の実装

コスト効率の高い組織になるには、セキュリティまたはオペレーションの能力と同様、知識を積み上る必要があります。そのためには、クラウド財務管理・コスト最適化に投資する必要があります。

消費モデルを導入

AWSでは、必要な費用のみを支払い、複雑な予測をすることなく、ビジネス要件に応じて使用量を増減できます。

全体的な効率を測定する

ビジネス成果とその実現にかかったコストを測定し生産性を向上させコストを削減することで得られる利点を把握します。

差別化につながらない高負荷の作業に費用をかけるのをやめる

AWSは、サーバー運用の高負荷作業を行うことでオペレーティングシステムやアプリケーションの管理に伴うオペレーション上の負担も解消されます。この結果、顧客やビジネスプロジェクトに集中できるようになります。

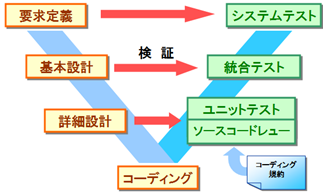
費用を分析および属性化する

システムの使用状況とコストを正確に特定し、分析することで、投資収益率(ROI)を把握できるため、リソースを最適化してコストを削減する機会が得られます。

# 図・表

## 要件定義

### V字モデル

厳密にはツールではないが、一般的に用いられるソフトウェア開発の流れの事。

また、

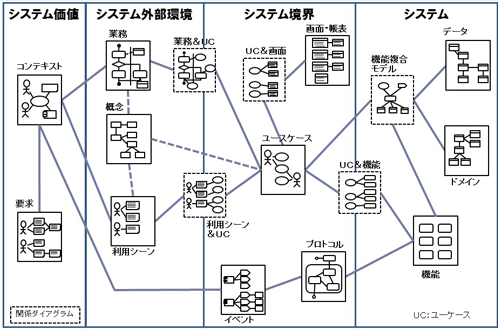
要件定義にはシステムテスト

基本設計には総合テスト

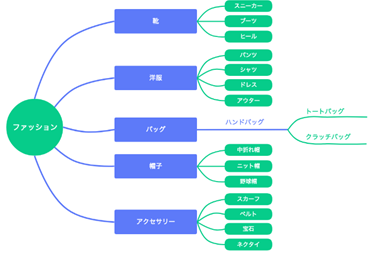
詳細設計にはユニットテスト（単体テスト）

が対応し、ソフトウェアの品質管理にも用いられる。

### 資料



### マインドマップ

参考資料

<https://note.com/abiruy/n/n8f55133a06d6>

核となる情報を中心に置き、その周りにアイディアに関するトピックを派生させる。

関連するアイディア同士を結び付けたり、情報を整理するのに役に立つ

用途

比較

### ユースケース図

Use Case Diagrams

#### 概要

システムにおいて、誰がどの機能を利用するかを表現する為の図。

補足

粒度 ユースケースの細かさ。例えば、「本を登録する」

と「本の登録情報の作成、本の登録情報の削除」では粒度が異なる。

|  |  |
| --- | --- |
|  | 備考／概要 |
| アクター | システムを利用する人、関係する外部システム |
| ユースケース | 「○○を△△する」というシンプルな書き方で、システムが持つべき機能の大枠を表現する。粒度を統一する事が重要。 |

#### 作成の仕方

１　アクターを全て洗い出す

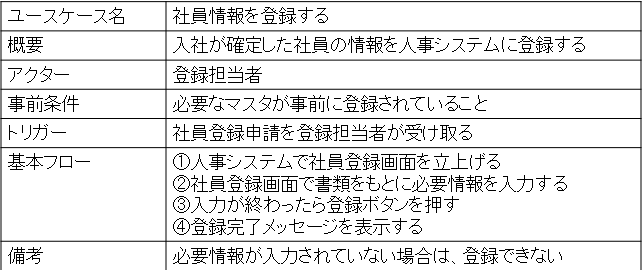
２　アクター別にユースケースを洗い出す

３　ユースケース図の構成、ユースケースの粒度を決める

４　ユースケース図作成

（５　ユースケース記述を作成） 任意

#### ユースケース記述

ユースケース図はシンプルさが長所でもあるが、シンプルすぎて誤解を生む可能性もある。

簡易的にユースケースごとに「ユースケース記述」を作成すると良い。

#### 注意点／備考

・使用する用語を統一し、表記揺れが起こらないようにする。

・人のイラストは必要では無いので、アクター、ユースケース列を持つテーブルを作成しても良い。

・大カテゴリ、中カテゴリ、小カテゴリで別々のユースケースを作成するのが良い

#### 用途

・エンドユーザー向けの説明資料

#### 詳細情報

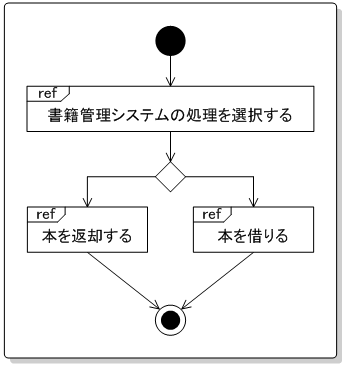
## 基本設計

### ER図

### 相互作用概要図

Interraction Overview Diagram

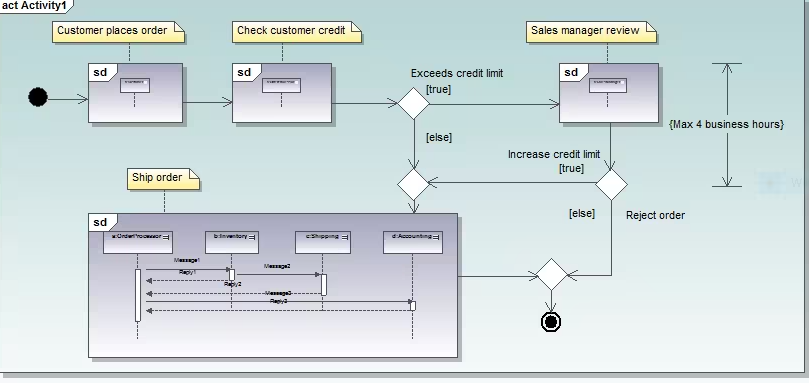
概要：ユースケース図やシーケンス図を構成要素としてより大枠の処理の流れをあらわす



詳細

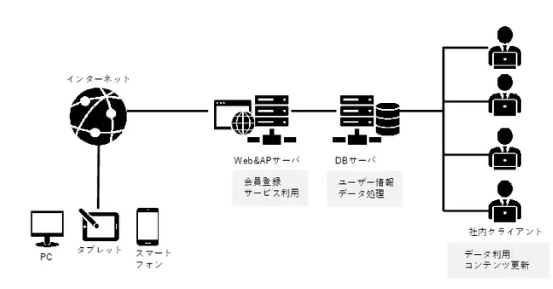
相互作用図（ユースケース図やシーケンス図など）が、より広域のシステム構成から見たとき、それぞれがどのように連携しているのかを表現する図

例



情報元；<https://www.altova.com/ja/umodel/interaction-overview-diagrams>

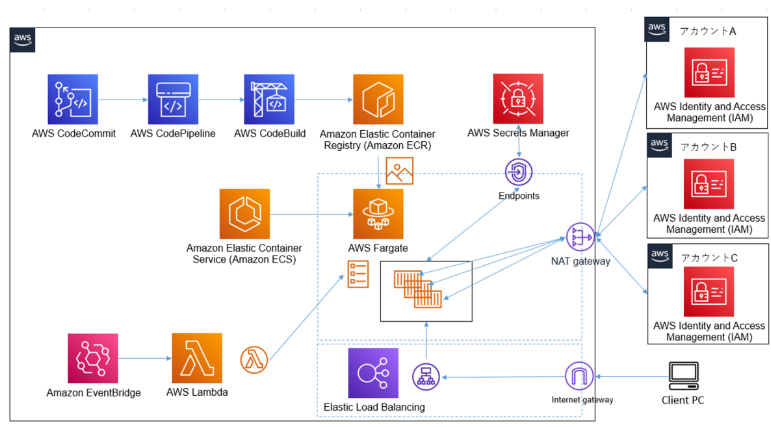
### アーキテクチャ図

別名：全体構成図、architecture diagram

システム全体で用いる機器やクラウドサービスなどの構成を記したもの。

書き方にルールはあまりないらしく、自由に書いてよい、との事。

補足：特にクラウドサービスを利用したアプリケーションで作成される傾向が強い。

例

### アクティビティ図

別名：Activity Diagram

#### 概要

システム実行時における一連の処理の流れや状態遷移をあらわす。

フローチャートを横にしたもので、**対象範囲の広いフローチャート**とも言える。

#### 比較

シーケンス図 複数の「オブジェクト」間の**ロジックの流れを時系列で**表現

フローチャート 処理タイミングの記述。

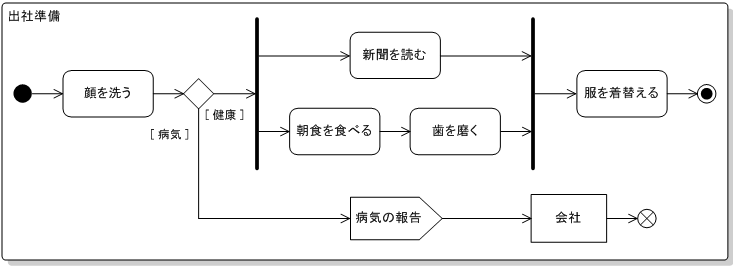
#### 作成の仕方

#### 詳細

パーティションによって利用者などを分割し、それぞれのプロセスを可視化する。

アクション、決定、分岐、ループなどシステムがどのような処理を行うかを表す。

#### 具体例



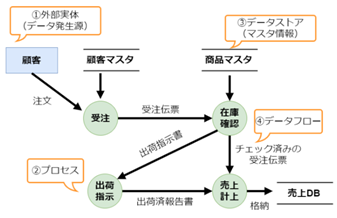
資料

<https://www.itsenka.com/contents/development/uml/activity.html>

### データフロー図

別名：DFD；Data Flow Diagram

#### 概要

システムの１機能の構成に必要な**入出力データと処理の流れを視覚的に表現**したもの。

処理手順と処理モジュール（または関数）を同時に考える時に便利。

#### 作成の仕方

１　データの発生場所（外部実態）を把握

２　データの保存場所（データストア）を把握する

３　処理を列挙

４　線でつないでDFDに落とし込む

メモ

アクティビティ図より細かく、シーケンス図よりは大雑把なレイヤーの仕様を考える時に分かりやすそう。

#### 比較

シーケンス図 処理タイミングの記述

#### 詳細

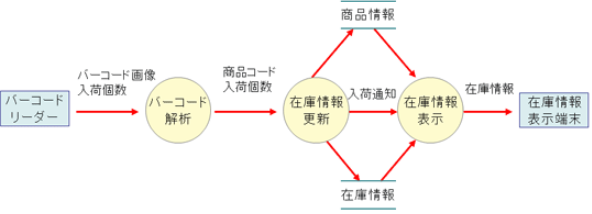
要素

|  |  |  |
| --- | --- | --- |
| 要素 | 写真 | 説明 |
| 処理（機能）  Process |  | 受け取ったデータを加工する |
| 外部実態  （入出力）  External Entity |  | 入力されるデータの内容や種類を表す |
| データストア |  | データの保管場所 |
| データフロー |  | データの移動先 |

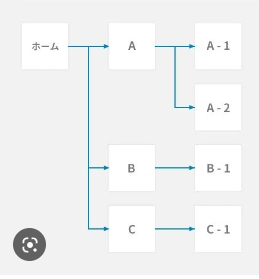
#### 注意点

・一枚に記載できない場合はレベル分けして記載。全体像を記載したものから始まり、レベルを下げていく

#### 具体例



### 画面遷移図



ツール

Overflow

<https://www.tam-tam.co.jp/tipsnote/tool/post17911.html>

guiflow（UI Flow）

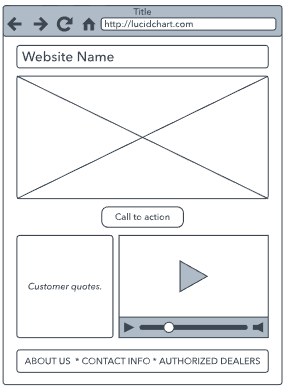
<https://github.com/hirokidaichi/guiflow/releases/tag/v_0.1.1>

他：draw.io、Prott、figma

参考：<https://www.lucidchart.com/pages/ja/examples/screen-transition-diagram>

### ワイヤーフレーム

#### 概要

アプリの画面（GUI）部分の設計図。特にWebページ。

ページのレイアウトやコンテンツの配置を定めた設計図

目的

・デザインは良くても、必要なコンテンツが足りていない　という状況を防ぐ。

・コーディングに入ってからのレイアウト変更　などを避ける。

・レイアウトや機能を制作者や利用者で共有する

参考：<https://www.lucidchart.com/pages/ja/wireframe>

#### 作成の仕方

１　掲載したい要素（必要情報）を洗い出す

２　要素ごとに優先度を決める

３　大枠のレイアウトを決める（マルチカラム、シングル、フルスクリーン、タイルなど）

（４　手書きでラフ画を作る）　任意

５　ツールで聖書

#### 詳細

・エンジニアよりも、ディレクターにあたる人が作成した方が効率良い（らしい）

・エクセル、パワーポイントでも作成する事はできる

#### 注意点

#### 具体例

## 詳細設計

### 決定表

別名：Decision Table

想定されるすべての条件の組み合わせと、対応する動作の関係を表した表。

用途

・社内のルールを記述する

・システム設計の初期段階でやや複雑な条件をまとめる

・ソフトウェアテストの際、条件とそのあるべき結果をまとめる

### シーケンス図

別名：

#### 概要

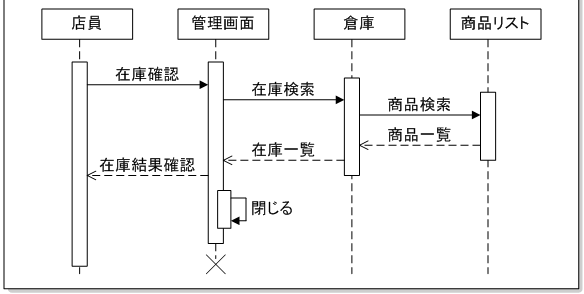
あるシステム内のある機能の開始から終了までの**複数の「オブジェクト」間のロジックの流れ**を時系列に沿って視覚的に表現したもの。

「オブジェクト」

２つ以上の装置 ：

２人以上の人間 ：［上司、部下］、 ［店員、顧客］

複合 ：［人、Webブラウザ］



各オブジェクト間の相互作用を時間の経過に沿って表す。

メッセージの送受信、タイムライン、合意事項などを表す為に用いる

#### 比較

シーケンス図 処理タイミングの記述

データフロー図 データと処理の流れを同時に確認

## プログラム設計

### クラス図

別名：

概要：クラスの定義、関連付けなど

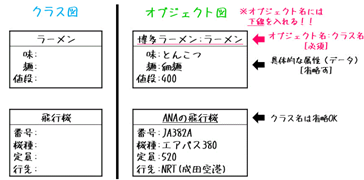
オブジェクト指向プログラミングにおけるクラスの構造と関係性を表現する。

継承関係、リレーションなども記述できる。

ER図と似ているが、ER図にはメソッドの記述がない。

参考：https://www.edrawsoft.com/jp/uml-class-diagram-relationships.html

### オブジェクト図



システム実行時のインスタンス（オブジェクト）の状態や振る舞いを可視化する。

用途

比較

クラス図との違い

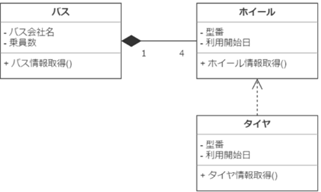
|  |  |  |
| --- | --- | --- |
|  | クラス図 | オブジェクト図 |
| 表現方法 | 抽象的 | 具体的 |
| 表現するもの | 静的な構造 | 動的な構造 |
| 目的 | 設計、実装の指針を提供 | 要件の明確化 |

### フローチャート

低レイヤー（マイコンなどの１デバイス）の処理をまとめるにはいまだに有効。

|  |  |  |
| --- | --- | --- |
|  | 高レイヤー | 低レイヤー |
| フローの流れ | 横（時系列） | 縦（処理ごと） |
|  |  |  |

### パッケージ図

Package Diagram

用途

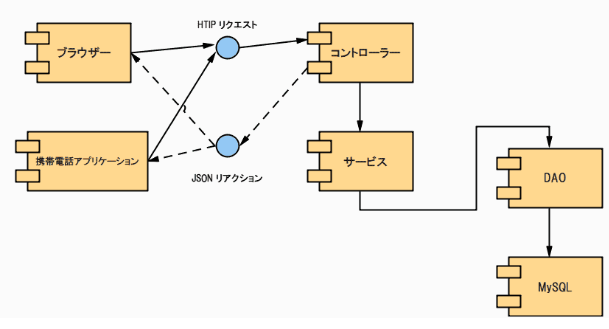
比較

### 複合構造図

Composite Structure Diagram

概要：複数クラスを内包するクラスやコンポーネントの内部構造をあらわす

### コンポーネント図

Component Diagram

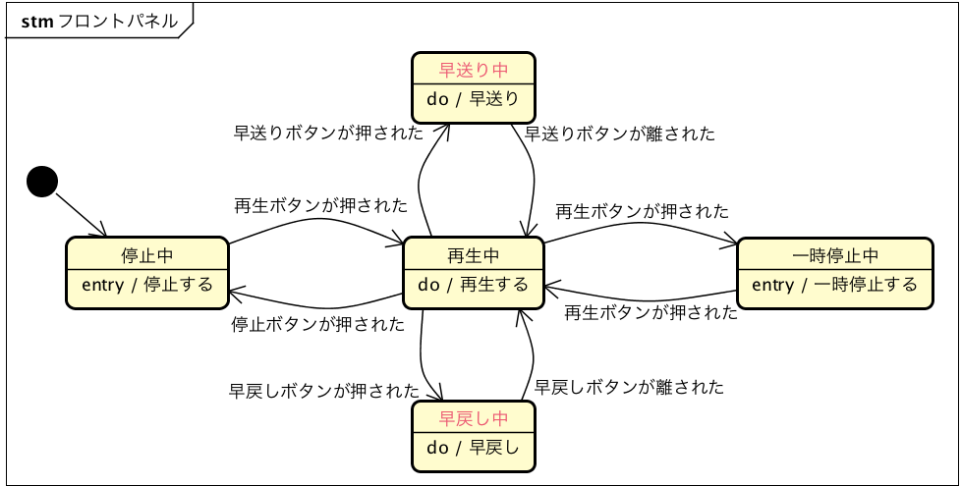
概要：システムを構成する複数のクラスを1つのコンポーネントとみなし、その内部構造と相互関係を表現する

## 設備設計

### 状態マシン図

#### 概要

State Machine Diagram、ステートマシン図

ボタン押下などの「イベント」によって起こるオブジェクトの状態遷移をあらわす。

状態の階層化、複数の状態を表現できる。

#### 用途

- 機械やマイコンの機能を確認する

- テスト項目の漏れや抜けを防ぐ

- １機能や１装置の全体的なイメージを合わせる

- １機能や１装置のできる事、できない事を明らかにする

- フラグが乱立して複雑化した状態を整理する

#### 作成の仕方

１　四角を作成し、状態名を書いていく

２　遷移を矢印で記入する

３　矢印のそばにイベントを記入する

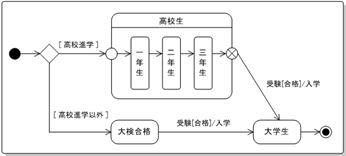
#### 注意点

あまりにも複雑すぎる条件を記述するのは向かない

#### 詳細情報

比較

決定表 状態マシン図よりもさらにシンプル

補足１：状態遷移図　State Transition Diagram

発生するイベント群による状態の遷移と、状態や状態遷移に付随するアクションによって動作を表す。状態マシン図は、状態遷移図に対して状態の階層化やいくつかの疑似状態を追加して拡張したものです。

補足２：状態遷移表　というものも存在する。状態遷移図作成より先に作成すると良いらしい。



N Not Allowed

I Ignored

### コミュニケーション図

Communication Diagram

オブジェクト間の関連とそこで行われるメッセージのやりとりをあらわす

用途

比較

### 配置図

Deployment Diagram

概要：システムを構成する物理的な構造をあらわす

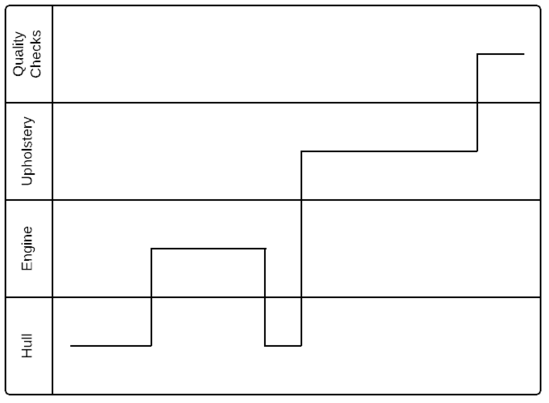
用途

比較

## 機器設計

### タイミング図

用途

電子機器、電気機器の信号のタイミングを表す

比較

## その他代表的なもの

### 雛形

#### 概要

目的

#### 比較

#### 作成の仕方

ページ区切り

#### 詳細

要素

#### 注意点

#### 具体例

### 仕様記述言語

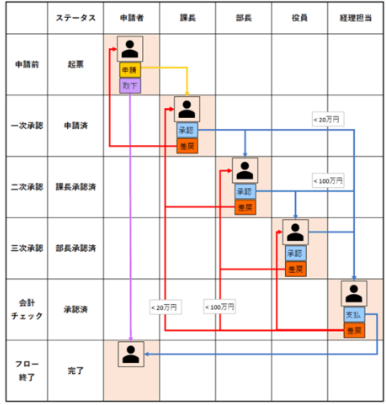
SDL；Specification and Description Language

フローチャートとあまり違いが無いように見える。

あまり使われていない印象。

### Work Breakdown Structure diagram

### これは何と呼べばよい？



## UML

### 概要

オブジェクト指向のシステム開発のツール

メジャーどころは

|  |  |
| --- | --- |
|  | 概要 |
| ユースケース図 | 利用者、利用環境からどのような機能が必要かを記す。 |
| データフロー図 |  |
| ER図 | システムで用いるデータとデータの関係を記す。 |
| アクティビティ図 | システムの１機能の開始から終了までの流れを記す。 |
| シーケンス図 |  |

・立ち位置は微妙である

・自身の頭の整理のために書く

### ツール

PlantUML <https://plantuml.com/ja/>

参考資料

<https://trslog.com/post-1685/>

<https://qiita.com/mumucochimu/items/e97a03217588f4101849>

13種類のUMLを紹介

# 作図ツール

## cacoo

<https://cacoo.com/ja/home>

## draw.io

<http://draw.io/>

# プロジェクト全般

## 打ち合わせ

### 雰囲気作りが大事

-　始まる前に良い雰囲気を作る

わざわざ「良い部屋」を取って会議をする　など

-　アイスブレイク

ボードゲームとか？

私的なできごとを一人一つ話させる

議題に関連する、近日中の話題

-　ゲーム化する

同じ事をした仲間、という一体感。

10秒で１アイディアだすゲーム

-　会議の結論は参加者全員が作る

・

### いい意見　より　たくさんの意見

-　カードに書いてもらう。参加者は書くだけでも満足する。50文字くらいをめどに（Twitter？）

-　聞く事　が大事。言いたくなるのを我慢する。

・全員の発言回数をできるだけ均等に

強制的に「参加」させる

会議の最初は「考える時間」に紙に書かせて提出　など

### 資料はできるだけ実物を用意

-　参加人数より少ない数の資料（共有させる）

### 時間厳守

「なんでもない話」をしてもらうときに敢えて制限時間を設けてみる

## ソフトウェア開発

### プログラム開発手法

ペア・プログラミング

２人のプログラマが協力して、１つのプログラムをコーディングする。

リファクタリング

外部仕様を変更すること無く、内部構造を変更する。

テスト駆動開発

テストケースを先に設定してからプログラムをコーディング

プロトタイピング

試作品を早期に作成し、利用者からのフィードバックを得ながら開発を進める。

### テスト技法

命令網羅

すべての命令を少なくとも一回は実行する。

条件網羅

全ての分岐**条件**を網羅するようにテストを作成する。

すべての分岐方向を網羅するとは限らない。

判定条件網羅

全ての分岐**方向**を網羅するようにテストを作成する。

複数条件網羅

判定条件の真・偽をあらゆる組み合わせで網羅するテスト。

## CI/CD

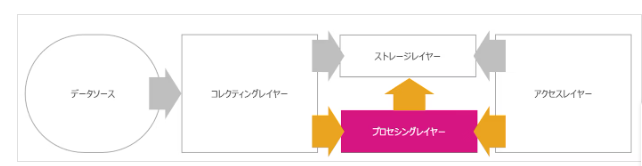
Continuous Integration／Continuous Delivery

継続的インテグレーション

ソフトウェア開発における、変更→ビルド→テスト→コンテナのクラスタへのデプロイまでの流れの事と、それを管理するツール群の事。。

# システム系　一般知識

## データ基盤の基本構造



### プロセシングレイヤー

## Wake On Lan

### ボード側（BIOS/UEFE）

ASUSのボードの場合

Advanced Modeにて：詳細 > APM > PCIEによる電源ON

### OS側

#### Windows

ネットワーク接続　を開く

デバイスマネージャ or ウィンドウキー＋[R]キー > ファイル名を指定して実行：devmgmt.msc

> ネットワークアダプター > ｛対象デバイス ｝> プロパティ

ここで以下の2つを設定

> 詳細設定 >「Wake On Magic Packet」

> 電源の管理

このデバイスで、コンピューターのスタンバイ状態を解除できるようにする

Magic Packetでのみ、コンピューターのスタンバイ状態を解除できるようにする

にチェック

#### Ubuntu

sudo apt install ethtool

ifconfig # まずはデバイス名を調べる

ethtool wlp2s0 # ethtool ｛デバイス名｝

sudo ethtool -s enp0s25 wol g # 有効化する

但しこのままだと再起動時にoffされてしまうらしいので、systemdのサービスを作成するなどして自動的にonにする必要がある。

### マジックパケットの送信

#### Debian系Linux

sudo apt install etherwake

sudo etherwake XX:XX:XX:XX:XX:XX # 起動させるマシンのMACアドレス

sudo apt install wakeonlan

wakeonlan XX:XX:XX:XX:XX:XX # 起動させるマシンのMACアドレス

## Web三層構造

### 概要

Webアプリの特にハードウェア構成をプレゼンテーション層、アプリケーション層、データ層の３層、またはそのようなシステム構成の事。

|  |  |
| --- | --- |
| メリット | デメリット |
| ・負荷分散  ・改修がしやすい  ・セキュリティの確保  ・障害の被害を限定的にする事ができる | ・ハードウェア費用  ・通信的なオーバーヘッドが発生する |

### プレゼンテーション層

GUI部分。Webブラウザ。

### アプリケーション層

アプリケーションサーバー（APサーバー）

### データ層

データベース（DBサーバー）

## SCADA

Supervisory Control And Data Acquisition

設備，インフラのデータ収集システムで，コンピューターとリモートI/Oユニットを組み合わせ、データ収集を行うシステム。その為，広義にはロガーなどもSCADAに含まれる。

・近年では統合システムを開発するツールの事を指すらしい。

・通信はEhternetやRS485などのシリアル通信を用いる事が多い。

具体的な用途

ビル監視 防犯カメラの映像，空調管理。

インフラ 道路，街灯，防犯目的でのセンサーやモニターの利用。

半導体製造 生産ラインの各工程の状況確認。工場の監督役。

具体的な製品

FA-Panel6 ￥330,000/10クライアント

zenon 開発ライセンス ￥150,000～

JoyWatcherSuite 開発版 ￥428,000 本体版￥128,000

（情報元）<https://www.scada-magazine.com/scadasoft-catalog/>

※海外製ソフトが多いらしい

三菱のSCADA <https://www.mitsubishielectric.co.jp/fa/products/hmi/scada/pmerit/genesis64/function/index.html>

用語（SCADA）

DSC；Distributed Control System

分散制御システム。マイコンやPLCなどシステムごとにコンピューターを使用する事。

MES；Manufacturing Execution System

「ヒト・モノ・時間」の生産資源を管理する，製造工程の見える化，作業者支援

# システムの種類

エキスパートシステム

ある特定の分野に特化した知識を基にルールベースの推論を行う事によって、専門家と同じレベルの問題解決を行う。

ウォークスルー型の会計ソリューション

ICタグ、カメラによる画像解析、電子決済システムによる無人店舗などの導入

# 教育

・分からない事は時間をかけすぎずに調べる。

・調べても分からない事は人に聞く。その際、ここまで調べた、と伝える。

・分からない事は何でも聞く。そのままにしない。

・理解した事を人に広める、教える。

# 保守関連

## 死活監視

簡易的な死活監視

・PINGを5回実行して、100%パケットがロスした時にメールを飛ばす

Zabbix

mackerel

DataDog

errbit

checkmk

## データ修正

データの修正はスクリプト言語で行うのが良い

理由

・修正対象のデータ抽出の条件を残す事ができる。

・具体的にどのような修正を行ったかを残せる

⇒ プログラムのユニットテストのような感覚

注意点

修正対象のデータをcsv, jsonなどで投入する場合は**テキストファイルでその抽出条件を残しておく**。

## ログ運用

ログ設定

管理すべきイベントなどのリストアップ

ログ転送

二次利用する為にデータを転送する

ログ保管

目的に応じた保管期間

セキュリティ

ログ利用

分析などを行う

## 可用性関連

### ディザスタリカバリ

#### RPO；Recovery Point Objective

別名：目標復旧時点

どの時点までのデータを復旧させるか。データベースの更新頻度が高い場合は、0秒（停止直前）のデータ復旧が求められる。

ディザスタリカバリの指標として使用される

#### RTO；Recovery Time Objective

別名：目標復旧時間

どのくらいの時間で（いつまでに）復旧させるか。サービスやシステムを停止していられる時間。

# 戦略系

## 3C分析

Customer、Competitor、Companyの分析の事。

## ITポートフォリオ

IT資産をさまざまな面から評価を行い、IT資源の最適配分を行うもの。

ベンチマーキング

ベストプラクティスを探し出し、自社のとの比較からその違いを分析する。

## SWOT分析

自社の外部環境と内部環境をStrength（強み）、Weakness（弱み）、Opportunity（機会）、Threat（脅威）の4つの要素で要因分析

## オズボーンのチェックリスト

アイデアがでない時に無理矢理アイディアをひねり出す、アイディア抽出の手法

## KJ法

ブレーンストーミングで出た多数のアイディアをまとめるときによく使われる。

アイディアを書き出す

アイデアをグルーピング

関係性を言語化

# 各種サービス

## ハウジングサービス

別名：コロケーションサービス

物理サーバーを設置するためのデーターセンターを提供するサービス

# 資料

## 経営系

### サービス窓口

SLA；Service Level Agreement

提供するサービスの内容、範囲を文書化したもの。

SPOC；Single Point Of Contacet

サービスの問い合わせ窓口を一本化したもの

SMS；Service Management System

ITサービス提供者のサービスマネジメントの活動を指揮し、管理する仕組み。

### そのほか

通信系へ

ブロックチェーン

ネットワーク上で発生した取引データをブロック単位で記録。複数のコンピュータが同期しながらブロックを生成、保持する。

仮想通過マイニング

仮想通過のコンピュータ処理作業に参加し、報酬として仮想通貨を得る事。

クラウドソーシング

アカウントアグリゲーション

## データ関連

### RAID

RAIDには「レベル」もある。採用の多いレベルをまとめる。

～20.06.30まで勘違いしていたが，どちらかと言うとアクセス速度改善に焦点が当たっている技術

まとめ（編集中）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ストライプ | ミラー | パリティ | 備考 |
| RAID0 | 2 | 0 | なし |  |
| RAID1 | 0 | 2 | なし |  |
| RAID10 | 2 | 2 | なし |  |
| RAID2 | 2 | 0 | 1:バイト単位 | 誤り訂正はハミングコード |
| RAID3 | 2 | 0 | 1:バイト単位；専用ディスク |  |
| RAID4 | 2 | 0 | 1:専用ディスク；ブロック単位 |  |
| RAID5 | 2 | 0 | ブロック単位；各ディスク |  |
| RAID6 | 4 |  |  |  |

RAID0： ストライピング ディスク２台以上

１つのファイルを複数のディスクに分散して記録。アクセス速度向上。**バックアップ能力はない**。

RAID1： ミラーリング ディスク２台

２台のディスクに全く同じファイルを書き込む。バックアップ機能のみ。

**アクセス速度には変化なし**。

RAID1+0 ディスク４台

文字通り1と0の組み合わせ。

※RAID10という表記も見受けられる

RAID5： 独立R/Wアクセス（パリティディスク分散） ディスク３台以上

ストライピングだが**データ（ブロック単位）とパリティをセットで別々のディスクに**書き込む。

RAID4で発生していた「ライトペナルティ」による負荷が抑えられている。**広く採用されている。**

RAID6： 独立R/Wアクセス（パリティディスク分散(2台) ディスク４台以上

ストライピング＋**２つのディスクにパリティデータ**を書き込み。安全面は高いが，効率と費用はやや高め。

以下はあまり使われないもの

RAID2： ストライピング（ハミングコード）

ストライピング＋パリティ専用ディスク。処理単位はブロックやバイト。ECC（Error Correction Code ；ハミングコード）と呼ばれる誤り訂正コードを用いる。データの**信頼性**確保。費用面の問題から，**ほとんど採用されていない**。

RAID3： ストライピング（パリティコード）

ストライピング＋パリティ専用ディスク。処理単位は**バイト単位**。こちらはパリティを使用。高いデータ転送速度。

（パリティには元データそのものも含まれ，障害時にはこれを利用してデータを復旧する）**あまり採用されていない。**

RAID4： 独立R/Wアクセス（パリティディスク固定）

ストライピング＋パリティ専用ディスク。（≒RAID3）処理単位は**ブロック単位**。

パリティディスクにアクセスが集中する為，アクセス速度は遅くなる。

## 見積もり手法

### キロステップ法

### ファンクションポイント法

1980年代頃には既に使われ始め、2020年代でも未だ現役の見積手法

### COCOMO

Constructive cost model

開発規模が分かっていることを前提として、工数と工期を見積もる。

### 類似法

過去に経験した類似のソフトウェアについてのデータを基にして規模と工数を見積もる。

### 標準値法

単位作業項目に適用する作業量の基準値を決めておき、作業項目を単位作業項目まで分解し、基準値を適用して算出した作業量の概算で全体の作業量を見積もる。

### 作業分解構成図

Work Breakdown Structure

## 信頼性設計

フールプルーフ（fool proof）

人間が操作を誤ったとしてもシステムの信頼性を保つ設計。

フェールセーフ（fail safe）

システムに障害が発生した時に被害を最小限に食い止めて安全を保つ設計。

フェールソフト（fail soft）

システムに障害が発生した時に故障した部分を切り離すなどして障害の影響を最低限運用を続ける為の設計。

フェールアボイダンス（fail avoidance）

そもそもシステムが停止しないように務める設計。信頼性の高いパーツのみ使用する、等。

フェールトラレンス（fail torrance）

同一の機能を持つ部品を２つ用意するなどして、障害が起きたときでも運用を継続できるようにする為の設計。

## その他資料

### Google社：ソフトウェア開発チームのパフォーマンスを示す 4 つの指標

<https://cloud.google.com/blog/ja/products/gcp/using-the-four-keys-to-measure-your-devops-performance>

# 用語（全体）

BABOK；Business Analysis Body Of Knowledge

PMBOK；Project Management Body of Knowledge

ERP；Enterprise Resources Planning

企業経営の基本となる資源要素（ヒト・モノ・カネ・情報）を適切に分配し有効活用する計画＝考え方

CSR調達

社会的責任の観点から調達先の選定条件を設定したり、調達先を選定したりすること

CI/CD；Continuous Integration／Continuous Delivery

継続的インティグレーション／継続的デリバリ

CSF；Critical Success Factor

重要成功要因

業務評価のサイクルにおいて、計画策定時に設定する、その企業や部署が目標達成のために、 重要と考えられる要素

DNC；Direct Numerical Control

コンピューターや入出力装置を用いて直接NC工作機械を制御すること。

KPI；Key Performance Indicator

重要業績評価指標

経営計画に沿った実績に導かれているかどうかなど、組織の業績管理評価を判断するための指標

LT；Lightning Talk

3～5分程度の短い時間で発表するプレゼンテーション。

MTTR；Mean Time To Repair

故障によりシステムが停止してから、修理を完了して稼働を再開するまでの時間の平均値。

RPO；Recovery Point Objective

インシデント発生時に、過去のどの次点までのデータを復旧させるかの目標値。

RFP；Request For Proposal

提案依頼書

RDRA

リレーションシップ駆動要件分析

システムを、価値、外部環境、境界、システムの４レイヤーに分けそのエリアに配置されたアイコンを線で結んで関係を示す図

RFI；Request For Information

情報提供依頼書

SPOF；Single Point of Failure

その単一箇所が働かないと、システム全体が障害となるような箇所

Gartner, Inc.

IT分野を中心とした調査・助言を行う企業。

HMI；Human Machine Interface

別名MMI；Man Machine Interface

人間と機械が情報をやり取りするための手段や、そのための装置やソフトウェアなどの総称。コンピュータにおけるHMIは特にユーザーインターフェース（UI：User Interface）と呼ばれることが多い。

monolithic

モノリシック；ソフトウェア設計の際、部分的に分割をせず、全て同一のモジュールとして作り上げること

ペルソナ

マーケティングにおいて、自社の製品・サービスを利用する典型的なユーザー像の事。

可用性

別名：Availability

システムが継続して稼働できる能力