

# ドキュメント生産性向上のために

第 1 版

株式会社 ビー・エス・エス

# 目次

---

1. 背景
2. 課題
3. 具体的なドキュメント作成環境
4. 記述サンプル
5. 設計書に必要なもの
6. システム開発におけるドキュメントのトラブル
7. 最低限必要なドキュメント
8. ケースによって必要なドキュメント
9. 要件定義書
10. 基本設計書
11. 内部設計書
12. キュメントの種類

# 1. 背景

---

開発業務においては、各種ドキュメントに基づき、作業を進める。

たとえば、プログラム実装では、詳細設計書やテーブル定義書などが必要となる。

また、開発業務では進捗が悪くなることは日常茶飯事であり、ドキュメントの修正を後回しにして、実装を優先させてしまう。

この状態（即ち、ドキュメントとソースファイルの乖離）が、次工程までに解消されない場合、生産性の低下を招いてしまう。

そして、往々にして、解消されない場合が多い。

この原因は、ドキュメントを修正した後に実装するという手順を遵守しない事にあるのは明白である。

しかしながら、多くの開発現場では、実装作業を急ぐあまり、ドキュメントの修正を後回しにしてしまっているのも、現実である。

その結果、ドキュメントは信用できず、

- ソースファイルを解析し仕様を確認する
- 解析に漏れが生じる
- 潜在的バグを仕様と誤解する
- 各工程、各担当者が、上記作業を重複して作業する

などにより、生産性の低下や、品質の低下を招いている。

## 2. 課題

---

開発業務における生産性向上と品質向上のために、ドキュメントとソースファイルの乖離を防ぐこと。

### 2.1. ドキュメントの利用方法

ドキュメントとしての利用方法は、

- 納品物
- 決裁、承認の対象
- 仕様書
- 開発工程の前提資料

などである。

一般的には、いずれも作成後に変更はできない、あるいは変更しないものという位置づけと考えられている。（とくに、ウォーターフォールの場合、厳格に取り扱われる）

しかし、仕様書や開発工程の前提資料などは、その内容に変更は生じるものであり、その変更は関係者に周知されるべきものである。つまり、速やかにドキュメントに反映されるべきである。

そして、その変更は、履歴として残すことも重要である。

実は、ドキュメントの利用方法として、

- 作成の後、変更できてはならないドキュメント（以下、Complete 型ドキュメント）
- 使用していく中で、変更され、完成されるドキュメント（以下、Progress 型ドキュメント）

の二種類に分類できる。

### 2.2. ドキュメントに必要な要素

- 表紙
- 改訂履歴
- 目次
- 見出し、本文
- ヘッダー、フッター
- 画像
- 表

### 2.3. ドキュメントのファイル形式

(1) ドキュメントは、なぜ Excel や Word なのか

多くの場合、ドキュメントは Excel や Word で作成されている。その理由として、

- 他に適切な編集用アプリケーションがないため
- 納品物として、他者（主に顧客）でも閲覧可能な形式である必要性
- 非開発者によって編集する場合もある
- Excel は、管理用途での集計などのため、開発業務に浸透している
- ワープロソフトとしては Word 以外にも選択肢はあるが、Excel とセットで導入されている場合がほとんどであり、余分なコストがかからない
- ドキュメントに必要な要素を扱うことができる

などが挙げられる。

つまり、Excel や Word によってドキュメントが作成されているのは、

- 多くの人、組織などで、閲覧、編集が可能である

からであり、そのファイル形式によるものではない。

## **(2) Complete 型ドキュメントの場合**

当然ながら、Excel や Word のように、編集可能なファイル形式を避けるべきである。Excel や Word の場合、パスワードを付けるなどで、編集不能としている。

本来なら PDF を採用すべきではないだろうか。

なお、PDF として作成するためには、以下のように、いくつかの手段がある。

1. 直接 PDF ファイルを編集するのであれば、専用アプリケーションを使用する
2. PDF 形式として保存することが可能なアプリケーションを使用する
3. PDF 形式に変換可能なファイル形式で編集したのち、変換する

## **(3) Progress 型ドキュメントの場合**

変更しやすいことが求められる。もちろん、Excel や Word であっても支障はない。

しかし、主に変更するのは、設計者や開発者である。開発者はテキストエディターでのファイル編集作業は不可欠であり、また中心的でもある。その作業の途中で他のアプリケーションを使用することで、生産性が低下してしまう。

ファイル変更履歴管理は、ソースファイルの場合、Git や Subversion などを活用している。

ドキュメントについても、同様の管理方法とすることが好ましく、ソースファイルと同様に管理している場合多くなっている。

ただ、Excel や Word のファイルは、これら変更履歴を管理するツールが苦手としているファイル形式である。たとえば、変更前後の差分を確認することは難しく、ディスク容量も消費する傾向にある。

したがって、テキスト形式を採用すべきではないだろうか。

### 3. 具体的なドキュメント作成環境

#### 3. 1. 今回の環境におけるドキュメント像

	Progress 型ドキュメント	Complete 型ドキュメント
最終ファイル形式	テキスト (Markdown)	PDF
作成中ファイル形式	テキスト (Markdown)	←
変更履歴管理	要	←

必要アプリケーション	用途
テキストエディター (VS Code)	編集
Markdown Preview Enhanced (VS Code 拡張機能)	PDF 変換、変換結果確認
markdownlint (VS Code 拡張機能)	Markdown 文法チェック
Paste Image (VS Code 拡張機能)	画像ファイル張り付け
PlantUML (VS Code 拡張機能)	作図
vscode-pdf (VS Code 拡張機能)	PDF ファイル表示
Java 実行環境	作図
Graphviz	作図
Google Chrome	PDF 変換

## 4. 記述サンプル

この PDF ファイル自体が、上記の環境で作成されたサンプルでもある。

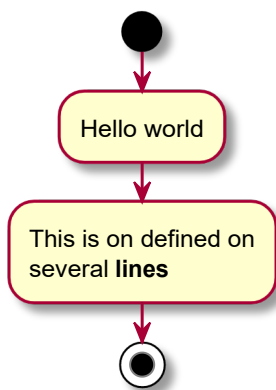
### 4. 1. 画像（UML 図）

#### (1) ソース

```
1 | @startuml
2 | start
3 | :Hello world;
4 | :This is on defined on
5 | several **lines**;
```

```
6 | stop
7 | @enduml
```

#### (2) 出力



### 4. 2. 画像（イメージ）

#### (1) 手順

1. 画像をクリップボードに入れる
2. 画像を張り付ける位置にカーソルを置く
3. [Ctrl]+[Alt]+V

#### (2) ソース

上記の手順を実施した結果、

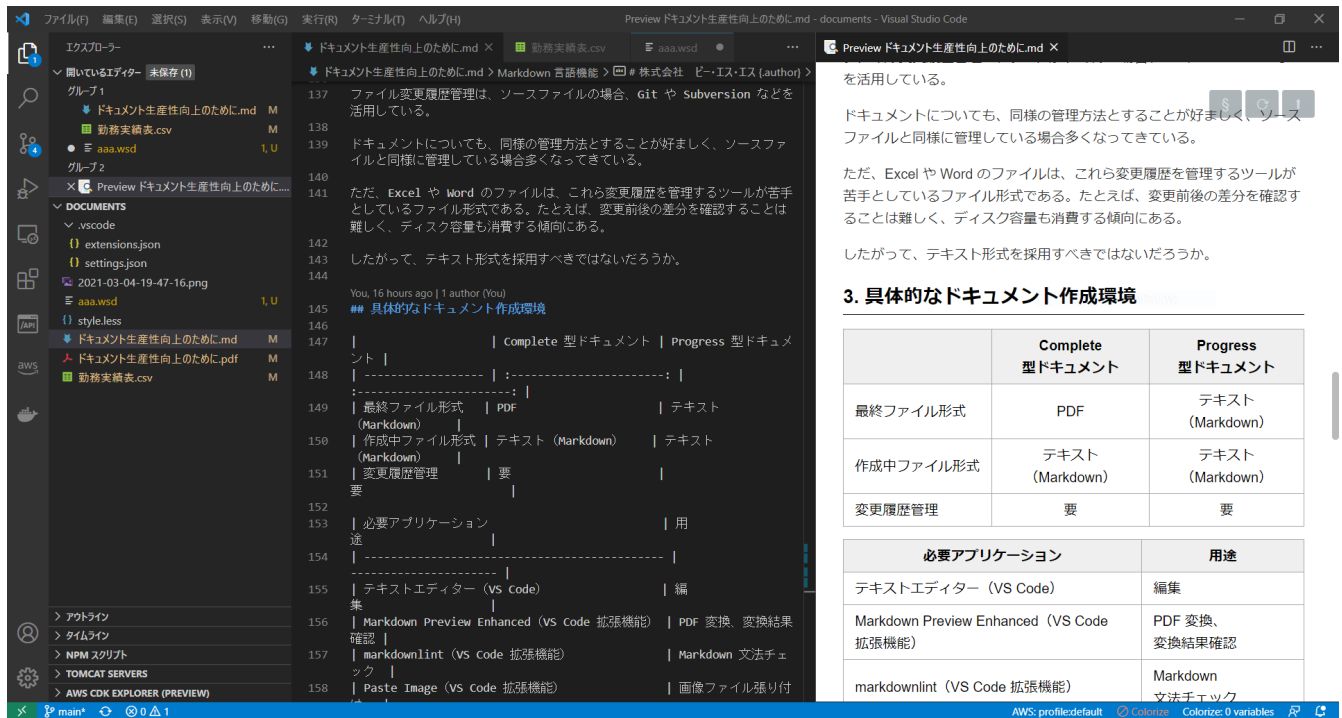
```
1 | 
```

のような記述が追加されると同時に、  
( )内のファイル名で、クリップボードの内容が保存される。

[]の中には、注釈（alternate text）を記入する。



## (3) 出力



Preview ドキュメント生産性向上のために.md

を適用している。

ドキュメントについても、同様の管理方法とすることが好ましく、ソースファイルと同様に管理している場合多くなっている。

ただ、Excel や Word のファイルは、これら変更履歴を管理するツールが苦手としているファイル形式である。たとえば、変更前後の差分を確認することは難しく、ディスク容量も消費する傾向にある。

したがって、テキスト形式を採用すべきではないだろうか。

### 3. 具体的なドキュメント作成環境

	Complete 型ドキュメント	Progress 型ドキュメント
最終ファイル形式	PDF	テキスト (Markdown)
作成中ファイル形式	テキスト (Markdown)	テキスト (Markdown)
変更履歴管理	要	要

必要アプリケーション	用途
テキストエディター (VS Code)	編集
Markdown Preview Enhanced (VS Code 拡張機能)	PDF 変換、 変換結果確認
markdownlint (VS Code 拡張機能)	Markdown 文法チェック

## 4. 3. Markdown として表を直接記述

### (1) ソース

```

1 | 見出し 1 (左寄せ) | 見出し 2 (中央寄せ) | 見出し 3 (右寄せ) |
2 | :---|:---:|---:|
3 | データ 1 | データ 2 | データ 3 |
4 | データ 1 | データ 2 | データ 3 |
5 | データ 1 | データ 2 | データ 3 |
6 | データ 1 | データ 2 | データ 3 |

```

### (2) 出力

見出し 1 (左寄せ)	見出し 2 (中央寄せ)	見出し 3 (右寄せ)
データ 1	データ 2	データ 3
データ 1	データ 2	データ 3
データ 1	データ 2	データ 3
データ 1	データ 2	データ 3

## 4. 4. csv ファイルを表として表示

### (1) ソース

```

1 | @import "勤務実績表.csv"

```

## 5. 設計書に必要なもの

---

全体像が把握できるもの

インフラ構成図 / システム間連携図 / ER 図

業務ごとの関連モジュールが把握できるもの

業務ごとに分けられた ER 図 / 業務ごとに分けられたクラス図\*1

具体的にどこを見ればいいのかを教えてくれるドキュメント

要件 / ユーザストーリーが把握できるもの

ユーザストーリーが書かれたチケット / 機能特性に応じたドキュメント

System of Record / System of Engagement などがヒントになりそう

## 6. システム開発におけるドキュメントのトラブル

---

最新版がどれか分からない、バージョン管理がされていない  
同じ名前や同じような内容のドキュメントが複数存在する  
適切に更新されず、最新版になっていない

## 7. 最低限必要なドキュメント

---

### 7.1. 仕様

画面設計書（デザインと機能説明含む）  
機能/画面一覧（画面名称・URL構造含む）  
メールやプッシュ通知の一覧と詳細  
フロー図

### 7.2. システム

ER図  
シーケンス図  
テーブル定義  
APIドキュメント（swagger）

### 7.3. インフラ

デプロイ手順・構成図  
非機能要件  
サーバー構築手順・構成図

## 8. ケースによって必要なドキュメント

---

体制図

単体テスト/結合テスト仕様書・結果

バージョン管理方法

ローカル環境構築手順

## 9. 要件定義書

---

要件定義の結果を文書にしたモノを要件定義書と呼ぶ。

参考 : <http://www.atmarkit.co.jp/ait/articles/0911/19/news109.html>

最低限明記しておいた方が良さそうな項目は以下の5つ。

### 9. 1. システム概要

何をするシステムなのか

どうしてそのシステムが必要なのか

そのシステムの目標は何なのか

システムの構成図

システム概念図

システムの業務フロー

ユースケース図

### 9. 2. 機能要件

システムの機能一覧

各機能の詳細

### 9. 3. 入出力要件

入力データ一覧

各入力データの詳細(データ項目・ファイル形式など)

出力データ一覧

各出力データの詳細(入力データと同様)

### 9. 4. 非機能要件

セキュリティ要求

品質・性能要求

### 9. 5. その他

概略スケジュール

ステークホルダー相関図 (関係図)

## 10. 基本設計書

---

### 10. 1. システム概要

シナリオ

ビジネス・ロジック

### 10. 2. システムの構成

システム構成図

業務フロー・アクティビティ図

ハードウェア・ソフトウェア構成図

ネットワーク構成図

### 10. 3. 機能一覧表

### 10. 4. データベース仕様

ER図

テーブル定義書

### 10. 5. UI設計

画面遷移図

画面レイアウト設計図

### 10. 6. その他

開発体制

開発スケジュール

プロジェクト管理ツール

## 11. 内部設計書

---

### 11. 1. 機能分割説明

クラス図

### 11. 2. データフロー

DFD図

### 11. 3. モジュール詳細

モジュール名

役割

引数・戻り値

処理概要

備考



## 12. キュメントの種類

---

ドキュメントは大きく分けて

- 開発するために必要なドキュメント
- 利用者向けのドキュメント
- 契約上必要なドキュメント
- 運用保守のためのドキュメント

に分類できます。