
DESIGN
ENGINEERING
55分でわかる！
システム開発の
超基本ワード集
START!

知っておくと差がつく！システム開発の基本用語

DESIGN ENGINEERING

	用語	意味	具体例
1	**ソフトウェアエンジニアリング**	ソフトウェアを「効率よく・安全に・質の高い状態」で開発するための考え方や技術のこと。	例えるなら、「家を建てるときの設計・工事・検査」すべてを丁寧に進める方法です。設計図を書く、プログラムを作る、テストして不具合を直す…という一連の作業を、より良く進めるための知識のことです。
2	**コンピューターサイエンス**	コンピューターがどう動くかを、仕組みレベルで学ぶ学問。	たとえば「パソコンがどうやってデータを記憶するの？」「AIはどうやって動くの？」といった、理論的な内容を扱います。プログラミングだけでなく、アルゴリズム（計算手順）やセキュリティ、ネットワークなども含まれます。
3	**ウォーターフォール**	ソフトウェア開発を「上から下へと流れる滝のように」段階的に進める方法。	最初に仕様を決めて→設計して→作って→テストして→納品する…という順番で、戻らずに進む方法です。手順がはっきりしていて、ルール通りに進めやすい反面、途中で変更しにくいという特徴があります。
4	**スパイラル**	小さな単位で「試作→評価→改善」を何度も繰り返して、徐々に完成度を高める開発方法。 リスク管理に優れた堅実な開発モデル。慎重に段階的に進めたいときに有効。	一度に完璧なものを作ろうとせず、まずはシンプルなものを作って試し、課題を改善しながら進めるスタイル。ウォーターフォールとアジャイルの中間的な位置づけです。
5	**アジャイル**	変化にすばやく対応できるように、小さな単位で作っていく柔軟な開発方法。 変化に強く、ユーザーのフィードバックをすぐ反映できる軽快な開発スタイル。	数週間ごとに「設計→開発→テスト→見直し」を繰り返し、都度改善していくスタイルです。ユーザーの意見を取り入れやすく、スピーディに価値を届けることができます。最近のIT開発現場で特に多く使われています。
6	**プロジェクトマネジメント（PM）**	目標を達成するために、決められた期間・予算・品質の中でプロジェクト（仕事のまとまり）を管理していく技術や考え方のことです。	プロジェクト全体の進行役。スケジュールを組んだり、チームメンバーを調整したり、トラブル対応をしたりします。リーダーシップとコミュニケーション力が求められる仕事です。
7	**プロセスマップ**	プロセスマップとは、プロジェクトの進行を「いつ・何を・どう進めるか」を整理した「全体の流れ」を示した図のようなものです。 「時間の流れ」のこと。	5つのプロセス群（プロセスグループ） ① 立ち上げ（Initiating） プロジェクトを始めるための準備。目的や責任者、予算などをざっくり決める。 ② 計画（Planning） スケジュール、予算、役割分担、リスクなどを細かく計画する。 ③ 実行（Executing） 実際にプロジェクトを動かし、成果物を作っていく。 ④ 監視・コントロール（Monitoring & Controlling） 計画と進捗のズレを見つけて修正する。品質や予算の管理も行う。 ⑤ 終結（Closing） 成果物を納品し、プロジェクトを正式に終える。振り返りも行う。
8	**知識エリア（Knowledge Area）**	プロジェクトを成功させるには、いろんな観点からバランスよく管理する必要があります。それを「知識エリア（Knowledge Area）」として9つに分けて考えるのがPMBOK（ピンボック）というガイドラインです。 「管理すべき項目」のこと。	① スコープマネジメント どこまでやるか（やること／やらないこと）をはっきりさせる。 ② スケジュールマネジメント いつまでに何をやるかを決めて、進捗を管理する。 ③ コストマネジメント お金（予算）の使い方を管理する。 ④ 品質マネジメント 作るものの品質（クオリティ）を保つための管理。 ⑤ 資源マネジメント 人やモノ（資源）をうまく配置し、無理なく進める。 ⑥ コミュニケーションマネジメント 関係者とちゃんと情報共有をし、連携を取る。 ⑦ リスクマネジメント トラブルになりそうなことを予測し、対応策を考える。 ⑧ 調達マネジメント 外注や発注など、他社との契約やモノの購入を管理する。 ⑨ ステークホルダーマネジメント お客さんや上司など関係者との期待をうまく調整する。
9	**QCD（Quality, Cost, Delivery）**	「品質・コスト・納期」という、プロジェクトで大事な3つの軸。	Quality（品質）：ちゃんと動く・バグがない Cost（コスト）：予算内でできる Delivery（納期）：期限までに完成させる この3つのバランスを取るのが、PMの重要な仕事です。

10 **PMBOK（ピンボック）**	世界中で使われている「プロジェクトマネジメントの教科書」。	「どうやって計画を立てる？」「リスクが起きたらどう対応する？」など、プロジェクトを成功させるための体系的な知識がまとめられています。資格（PMP）もあり、グローバル企業では重視されています。
11 **IT戦略**	ITを「会社の強み」にするための計画を立てること。	たとえば「業務を効率化するシステムを導入する」「データを活用して売上を伸ばす」といった、経営の目標に沿ってITを活かす方法を考えます。経営者とエンジニアをつなぐ役割でもあります。
12 **コンサルティングサービス**	企業のIT課題を見つけて、解決策を提案する専門家。	「システムが古くて業務が非効率」「データ管理がバラバラ」などの悩みに対して、現状分析から改善提案までを行うのがITコンサルの仕事です。技術とビジネスの両方の視点が必要です。
13 **IT関連の資格**	就職・転職活動でのスキルの裏付けになるほか、業務の中で必要とされる専門性の証明	未経験・職種選び段階：①ITパスポート→ITの基礎理解 就職・転職活動前：①基本情報技術者②CCNA③LPIC→実務的なスキルを見える化しやすい資格 就職後〜キャリア形成：①応用情報②AWS③PMP→上流工程や専門分野へのキャリアアップに有利
14 **業務の見える化**	会社やチームで行っている作業を、誰が見てもわかるように図や表で整理すること。	たとえば、レストランで「注文 → 調理 → 配膳 → 会計」という流れがあるとします。 この流れを紙に書き出すことで、「どこで時間がかかってる？」「無駄な作業はない？」といった問題点を見つけられます。 つまり、“作業を見えるようにする”ことで、改善ができるのです。
15 **フローチャート**	作業や判断の流れを図形と矢印で表現した図のこと。	「もし○○なら、こうする。違うならこうする」という処理の流れを図で表します。 たとえば、学校の先生が「遅刻してきた生徒がいたら→理由を聞く→理由によっては注意する」といった流れを、四角や矢印で表します。 迷わず行動するための「地図」のような役割です。
16 **産能大方式**	作業を細かく分けて時間や動きを分析することで、効率化を目指す方法。	料理の手順を「材料を出す → 野菜を切る → フライパンを温める → 炒める」など細かく分けて、 「この順番で合ってる？時間がかかりすぎてない？」と見直すイメージです。 無駄な手間を減らして、よりスムーズに作業ができるようになります。
17 **レーン・フロー・ダイヤグラム（LFD）**	「誰が、どの順番で、どんな作業をするか」を横並びで見えるようにする図。	プールの「泳ぐレーン」のように、部署や人ごとに横並びの線を作って、どの作業をいつするかを並べていきます。 たとえば： 営業部：お客様と契約 → 経理部に連絡 経理部：請求書作成 → 請求 このように誰がどの工程を担当するのかが一目でわかります。
18 **アクティビティ図（UMLの図の一種）**	作業や処理の「動きの流れ」を示す図。コンピューターの処理や業務プロセスを表現します。	例えば「ログイン処理」なら → ユーザー名とパスワード入力 → データベースで確認 → 成功したらマイページへ、失敗したらエラー表示 というような流れを、図にします。プログラムの動きを見える化するためによく使います。
19 **BPMN（ビジネスプロセスモデリング表記法）**	業務の流れ（プロセス）を、世界標準の記号ルールを使ってわかりやすく図にする方法。	「○＝スタート」「■＝作業」「◆＝判断」など、あらかじめ決められた記号を使って、業務の流れを図にします。 これを使えば、別の会社の人が見ても内容が理解しやすく、共通言語になります。
20 **IDEF（Integration Definition）**	業務やシステムの「構造」や「流れ」を分析・整理するための図式のルール。	業務を「どんな情報を使って」「どんな処理をして」「どんな結果が出るか」に分けて、整理する方法です。 たとえば「受注処理」なら： 入力：注文内容 処理：商品確認、在庫確認 出力：注文確定メール このように、全体像を「部品ごと」に整理できます。
21 **DMM（データマネジメントマップ）**	会社の中で「どの部署が、どんなデータを使っているか」を整理する図。	例えば、 営業部 → 顧客情報・見積もりデータ 商品部 → 商品マスタ・在庫情報 経理部 → 売上・請求情報 このように、どの部門がどの情報を管理しているかを一覧にします。 DX（デジタル化）や情報管理をする上でとても重要です。
22 **ROI（リターン・オン・インベストメント）**	投資（お金をかけた）ことに対して、どれだけ利益が返ってきたかを表す指標。	100万円かけて、120万円の利益が出たら → 20万円の得（＝ROI20%） 逆に80万円しか回収できなかったら → 20万円の損（＝ROI−20%） つまり、「この投資、割に合ってる？」を数字で判断する考え方です。 システム開発やIT導入の効果を測るときに使われます。
23 **アジャイル開発**	短い期間ごとに小さな成果を積み重ね、柔軟に進める開発方法。 プロセスやツールよりも個人と相互作用 包括的なドキュメントよりも動くソフトウェア 契約交渉よりも顧客との協調 計画の遂行よりも変化への対応	「全部完成してから納品」ではなく、「少しずつ作って、都度お客さんに確認しながら進める」スタイル。 簡単にいうと、マニュアルより話し合い、計画より柔軟な対応、完璧な設計よりまず動かすこと、契約よりチームワークを重視。
24 **イテレーション**	アジャイル開発で使われる「1サイクル」の単位。	「2週間ごとに小さな成果を出す」など、繰り返し改善する。

25	**XP（エクストリームプログラミング）**	チーム開発でミスなく素早く作るための実践的な開発スタイル。	ペアプログラミングやテストを重視。
26	**スクラム**	役割分担と毎日の進捗確認で、チームが自律的に動く開発スタイル。	ラグビーの「スクラム」のように、全員で1つの目標に向かって進む。
27	**リーン開発**	ムダを減らして、必要なものを必要なときに作る考え方。トヨタ生産方式がベース。	「冷蔵庫の中に余計な食材を入れすぎず、使う分だけ買う」ような開発。
28	**ファンクションポイント（FP）法**	機能の数で工数を見積もる方法。	レストランで「メニューの品数」「注文フォーム」「レシート印刷」といった機能の数を数えて、それぞれに点数をつけ、合計でかかる工数を予想するようなもの。
29	**LOC（Line of Code）**	コードの行数をもとに工数を推定する方法	「この記事を5000文字書いたから、これくらいの時間がかかる」と目安を立てるのと同じ。
30	**類推法**	過去の似たプロジェクトと比較して見積もる方法。	「前に作ったカフェ向けサイトも5日だったから、今回も同じくらい」など過去経験を参考にする。
31	**積算法**	各作業にかかる時間を個別に見積もり、それを合計して全体の工数を出す方法	「設計に10時間、開発に20時間、テストに5時間…と積み上げていく」ような形。
32	**COCOMO**	コード量やプロジェクトの性質を数値化して、自動的に工数を計算するモデル	エクセルなどで、入力項目を入れれば「このくらいの工数が必要」と自動表示される見積ツールのようなもの。
33	**一括請負契約**	成果物（完成したシステム）を納品するまでベンダーが責任を持つ契約形式	ケーキ屋さんに「結婚ケーキを全部お任せで作ってください」と依頼するような形。内容・納期・価格が決まっており、花嫁さんが出来栄えを待つ。
34	**RFP（リクエスト・フォー・プロポーザル）**	依頼内容（要望・条件など）をまとめた提案依頼書	「こんな旅行に行きたいんです。予算はこのくらい。プランを提案してください、という旅行会社への依頼書」のようなもの。
35	**要件定義**	何をつくるのか、なぜそれが必要なのかをハッキリさせる工程	目的：そのシステム・サービスで何を実現したいか 業務要件：現在の業務フロー、課題、どう改善したいか 機能要件：必要な機能（例：検索機能、予約機能など） 非機能要件：セキュリティ、使いやすさ、速度、拡張性など 利用者：誰が使うのか（顧客、社員など） 制約条件：予算、納期、使用技術など
36	**業務流れ図**	業務や作業プロセスの手順や関係性を図式化したもの。全体の流れを把握したいならこれ。	社内の「備品購入申請」の業務フロー↓ 「社員 → 上長 → 総務部 → 業者 → 経理」
37	**LFD（レーン・フロー・ダイヤグラム）**	業務や作業の流れを図で表したもの。詳細な判断・処理の流れを描きたいならこれ。 矢印で順番を示し、四角は作業、ひし形は判断を表します。業務改善したいときに有効。	「カフェの注文」↓ お客さんが入店 →メニューを見て注文 →スタッフに注文を伝える →注文内容を確認（間違い？はい→再確認、いいえ→次へ） →キッチンで準備 →商品を渡す →会計 →お客さん退店（終了）
38	**スイムレーン図**	担当部署ごとに分けて図示する業務フロー図。チーム共有や説明資料として使いたいときに有効。	リレーのように「営業」「設計」「承認」の順で業務をつなげて見えるようにした地図。
39	**アクティビティ図（UML）**	処理の流れに条件分岐や並行処理を含めて表現する図	ゲームのフローで「選択肢A→イベント1、B→イベント2」と分岐を図式化したようなイメージ。
40	**ユースケース図／アクター**	誰（アクター）が、何（ユースケース）をシステムで行うかを示す図	自動販売機を例に、「利用者」が「商品選択・購入」する操作流れを整理する図。
41	**機能構成図（DMM）**	システム全体を「○○機能」「△△画面」といった構成要素に分け整理する図	テレビの「チャンネル切替」「録画」「音量調整」など、どんな機能で構成されているかを見える化。
42	**機能情報関連図（DFD）**	システム内で「どのデータが、どこに流れるか」を図で表現する	「商品注文」データが「注文入力画面 → 注文DB → 売上管理画面」へ流れる構造を整理。
43	**プロセス中心設計（プロセス指向設計）**	システムの処理の流れ（プロセス）を中心に設計する方法です。 「何をするか」「どんな手順で進むか」を重点的に考えます。	銀行のATMシステムで、「お金を引き出す」処理の順番（カード挿入 → 暗証番号入力 → 金額指定 → 残高確認 → お金払い出し）を設計するイメージ。
44	**データ中心設計**	システムで扱うデータの構造や関係性を中心に設計する方法です。 「どんな情報を保存し、どう繋がっているか」を重視します。	顧客管理システムで、「顧客情報」「注文情報」「商品情報」などのデータがどう関連しているか（例えば顧客が複数の注文を持つ）を設計するイメージ。
45	**オブジェクト指向設計**	「データ」と「処理（メソッド）」を一つのまとまり＝オブジェクトとして設計する方法です。 実世界のモノや概念をプログラムの中で表現します。	車のシステム設計で、「車」というオブジェクトがあり、車は「色」や「スピード」といったデータを持ち、「走る」「止まる」といった動作（処理）を持つイメージ。
46	**外部設計／内部設計／詳細設計**	外部設計＝見た目・操作の仕様 内部設計＝中の処理構造やデータ構造 詳細設計＝テーブル設計・関数設計など	家づくりで「間取り図を決める」「壁や柱など構造を決める」「電気配線や家具配置まで細かく決める」に対応。
47	**ER図（エンティティ・リレーション図）／クラス図／シーケンス図／オブジェクト図／ステートチャート図**	システム設計の表記法↓ ER図＝データベースの設計図 クラス図＝プログラムの構成を表す図 シーケンス図＝オブジェクト間のメッセージや処理順序 オブジェクト図＝実行時の具体的なオブジェクトの状態 ステートチャート図＝状態遷移（例：待機→処理中→完了）を図で表す	家を建てるとき「どの部屋にどんな家具がある」「家具がどう動くか（ドア開閉）」「電気が入る・消える」などを整理するような細かな設計図。

48	**アルゴリズム**	ある問題を解くための手順やルールのまとまりのこと	ケーキを焼く手順↓ 材料を用意する→材料を混ぜる→型に流し込む→オーブンで焼く→冷ます この手順を間違えずに行うことで、ケーキが完成する。
49	**開発環境**	プログラムを作るための「作業場所」や「使う道具・ソフト」のこと。	パソコンにプログラムを書くためのソフト（例：Visual Studio Code）を入れる。 テスト用のデータベースやサーバーを準備する。 開発者同士が同じ環境で作業できるように設定する。
50	**構成管理**	プログラムやドキュメントなどのファイルを最新の状態に保ち、誰が何を変更したか記録する管理方法。	複数人が同じプログラムを編集しても、変更が競合しないように管理。 「前のバージョンに戻したい」ときに簡単に戻せる。 変更履歴をみて、誰がどんな修正をしたかわかる。
51	**MVCモデル**	「表示（View）・処理（Controller）・データ（Model）」を分離して設計するモデル	レストランで「料理を作る人」「注文を聞く人」「会計をする人」を担当別に分けるイメージ。
52	**テストファースト（テスト駆動開発）**	コードを書く前に「どう動くか（テスト）」を明確に決め、合格するようにコードを書くやり方。 メリット：バグを早く見つけられる。無駄なコードを書かない。改善しても安心して動作保証できる。長期的に開発効率が上がる。	「問題集の答えを先に知って、それから解答するような進め方」。
53	**ペアプログラミング**	2人1組で交互にコードを書きながら品質を高める開発手法。 メリット：ミスが減る。スキルを素早く共有できる。問題解決が早い。チームのコミュニケーションが良くなる。モチベーションが保ちやすい。	2人で一緒に道を作ると、安全かつ効率的に進められる状態。
54	**スマホアプリ開発**	スマホで使うメッセージ送受信や写真共有のアプリを作る開発。	LINEやInstagramのスマホアプリ。 iOS（iPhone）：Swift、Objective-C Android：Kotlin、Java クロスプラットフォーム　　：Flutter（Dart）、React Native（JavaScript/TypeScript）、Xamarin（C#）
55	**Webシステム開発**	ブラウザ上で動くオンラインショップやホテル予約サービスなどの仕組みを作る開発。	AmazonのECサイトや予約サイト。 フロントエンド：HTML、CSS、JavaScript、TypeScript バックエンド：PHP、Python、Ruby、Java、Go、Node.js（JavaScript）
56	**業務システム開発**	社内で社員の出勤時間や売上データを管理するためのシステム開発。	会社の勤怠管理システムや売上管理システム。 大規模業務アプリ：Java、C#（.NET）、Python データ処理・自動化： Python、Shell、VBScript、Perl データベース連携：SQL（PostgreSQL、MySQL、Oracle等）
57	**Webオーサリングツール**	専門的なコードを書かなくても、ドラッグ＆ドロップでWebサイトを作れるソフトやサービス。	Wix（ウィックス）、ペライチなど。 ボタンや画像を画面に置いていくだけでホームページが完成する。
58	**リビジョン**	プログラムやドキュメントの変更履歴やバージョン番号のこと。どのバージョンかを識別するために使う。	「このプログラムはv1.0、修正したらv1.1」というように番号がつく。 不具合があったときに、前のバージョン（リビジョン）に戻せる。
59	**単体／結合／システム／導入／機能テスト**	単体＝プログラムの1つ1つの部品が正しく動くか確認。 結合＝部品同士をつなげて、連携がうまくいくかチェック。 システム＝システム全体として仕様どおり動くか総合チェック。 導入＝お客様が実際に使ってみて、問題がないか確認。 機能＝動くことの確認	車を作るとき、部品ごとにチェックし（単体テスト）、部品を組み合わせて走るか試し（結合テスト）、全体の性能をチェックし（システムテスト）、最終的にお客さんが試乗して納得するか確認する流れ。
60	**ホワイトボックス／ブラックボックス／トップダウン／ボトムアップテスト**	ホワイトボックス＝プログラムの中身（コード）を見ながらテスト ブラックボックス＝外から見た動作だけでテスト トップダウン＝全体から詳細へテストする ボトムアップ＝詳細から全体へつなげてテストする	鍵穴から中を見てドアを確認（ホワイト）、ドアを押して開くか試す（ブラック）、上から順に中を見る／下（床）から順に見るようなテスト順序。
61	**静的解析ツール／性能・メモリ監視ツール／テスト・バグ管理ツール**	静的解析＝コードの質を自動チェック 性能・メモリ監視＝実行時の速度やメモリを監視 テスト・バグ管理＝不具合や修正履歴を記録・管理	文書チェックソフト（誤字検出）／車のスピードメーター／故障記録ノート。
62	**ファイヤーウォール／クラスタリング／ミラーリング／ハウジング／ホスティング**	ファイヤーウォール＝外部不正アクセス防止 クラスタ／ミラー＝機械が壊れても別の機械で動く構成 ハウジング／ホスティング＝サーバーを設置・保管する場所	家の鍵と防犯扉、スペア鍵と予備電源、レンタル倉庫と土地を借りる違い。
63	**上位モジュール・下位モジュール**	システムやプログラムを「部品（モジュール）」に分けて考えます。 上位モジュール：全体をまとめる「親」の役割を持つ部分。 下位モジュール：細かい処理を担当する「子」の役割を持つ部分。	家を建てるとき、 上位モジュールは「リビング」「キッチン」「寝室」という大きな部屋の設計図。 下位モジュールは「リビングの窓の開閉装置」「キッチンの蛇口の配管」などの細かいパーツ。

64 **システムテスト**	システム全体が設計通りに動くかをチェックする最終テストのこと。	新しく建てた家を、家具を入れて生活してみて、水道や電気が全部ちゃんと動くか確かめる作業。
65 **負荷テスト**	システムにたくさんの利用者や処理を一気にかけて、耐えられるかを調べるテスト。	レストランの厨房に、一度に大量の注文が入ったときに料理がちゃんと間に合うか試すこと。
66 **パフォーマンステスト**	システムの処理速度や動作の効率を測定し、快適に使えるか確認するテスト。	車のエンジンがスムーズに動いているか、スピードや燃費が良いかをチェックするようなイメージ。
67 **障害テスト**	システムに故障やエラーが起きたとき、正しく対応できるかを確かめるテスト。	火災報知器が鳴ったときに、きちんと避難アナウンスが流れるか確認する作業。
68 **セキュリティテスト**	不正アクセスやデータ漏洩を防げるかを確認するテスト。	家の鍵や防犯カメラがちゃんと機能して泥棒を防げるか確かめること。
69 **構成テスト**	システムの構成要素（機器・ソフト）が正しく連携して動くか確認するテスト。	テレビ・レコーダー・スピーカーをつなげて、ちゃんと映像や音が出るかチェックする作業。
70 **受入れテスト**	お客様がシステムを実際に使って、仕様どおり動いているか確認する最終テスト。	新築の家を施主さんが内覧し、気になるところがないかチェックして「合格！」と判断する場面。
71 **マニュアル**	システムの使い方やトラブル時の対応方法を書いた説明書。	新しい家電製品の取扱説明書や、料理レシピのように、誰でも使えるように案内するもの。
72 **システム開発のV字モデル**	システム開発の流れを「V字」の形に整理した考え方。 左側で設計や計画をしっかり進め、右側でその設計に対するテストを段階的に行うイメージ。	家づくりで「設計図を描く → 土台を作る → 壁を立てる」左側工程に対し、右側で「基礎検査 → 内装検査 → 最終検査」を丁寧に進める流れ。
73 **SLA（Service Level Agreement：サービスレベル合意）**	SLAとは、「サービス提供者（IT会社など）」と「利用者（顧客）」の間で取り決める サービスの品質・内容・範囲・対応スピードなどを定めた文書（契約）」のこと。 お客様との「約束ごと」（品質・対応時間など）。	クラウドサーバーを契約している場合↓ 【稼働率】「99.9%の稼働率保証」＝月間43分以上サーバーが止まったら違約 【障害対応】「障害発生から1時間以内に復旧対応開始」 【違反時】「達成できなかった場合は月額費用の10%を返金」など
74 **ITIL（Information Technology Infrastructure Library：情報技術インフラストラクチャ・ライブラリ）**	ITサービスマネジメントの「ベストプラクティス（成功事例や標準的な方法）」をまとめたガイドライン集のこと。 その約束を守るための「運用の手引き」	たとえばIT部門でメールシステムの運用をしている場合↓ 【設計】「メールの遅延をなくす設計・SLAを定める」 【運用】「障害が起きたら○時間以内に対処する」 【改善】「月に1回、トラブル分析会をして対策を立てる」
75 **システム障害**	コンピュータやネットワークなどのITシステムに何らかのトラブルが発生して、正常に動かなくなる状態のこと。	スマホのアプリが急に落ちて開かなくなるのと同じ。企業ではこうしたことが大きな損失になるため、原因調査や早期復旧が重要になります。
76 **ホスティング（Hosting）**	業者が用意したサーバーを借りて、自分のWebサイトやメールなどのサービスを運用する方法。	Xserverのレンタルサーバーを借りて、会社のホームページやメールサーバーを運用
77 **ハウジング（Housing）**	自分で用意したサーバー機器を、専門のデータセンターに設置して運用する方法。	自社で購入したDELLやHPのサーバーをさくらインターネットの石狩データセンターに設置して、全国の支店向けに基幹業務システムを提供
78 **システムリプレイス**	古くなったシステムを新しいシステムに置き換えることです。「リプレイス＝入れ替え」と考えるとわかりやすいです。	古くなったガラケーを最新のスマホに替えるようなもの。使い勝手も性能も大幅に向上します。
79 **レガシーシステム**	古くから使われていて、今の技術では扱いにくいけれど、業務に深く関わっていて簡単にはやめられないシステムのことです。	昔のVHSビデオデッキ。まだ動くけど、部品も少ないし修理も難しい。だけど録画した大事な映像が入ってるから捨てられない…そんな状態。
80 **スクラッチ開発**	既存のシステムやテンプレートを使わずに、ゼロからオリジナルでシステムを作ることです。	会社専用の勤怠管理システムを導入したい。ただ、「うちの会社だけのルール」があるから、既製のソフトじゃ合わない。だから、ログイン・打刻・有休管理もすべて自社用に開発した。
81 **COBOL**	1960年代に生まれた、昔からあるプログラミング言語のひとつ。今でも銀行や保険業界の基幹システムなどで使われています。	「レトロだけど信頼性のある」言語。トヨタのクラシックカーのように、古くても動き続ける力がある。しかし開発者の高齢化などで、後継者不足が課題です。
82 **C/Sシステム**	「クライアント / サーバーシステム」の略。パソコン（クライアント）がサーバーと通信しながら動作するタイプのシステムです。	お店の注文をキッチンに伝えて料理を出すスタイルと似ていて、「注文する人＝クライアント」「作る人＝サーバー」の関係性です。
83 **ITトレンド**	IT業界で今注目されている技術や考え方、サービスのことです。「トレンド＝流行」なので、変化も早いです。	生成AI（例：ChatGPTなど） ノーコード／ローコード開発（プログラミングしなくてもアプリが作れる技術） クラウドサービス（Google Drive、AWSなど）

LINE公式アカウントにて
最新情報配信中！

変化を楽しみ、自分らしく未来へ。

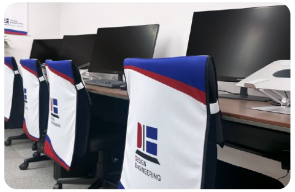
デザインエンジニアリングは、
挑戦するエンジニアの一歩を応援する会社です。

“好き”や“ワクワク”をそのままキャリアに変え、
自分の可能性を信じて前へ進む人には、無限のチャンスが広がっています。

失敗も学びに変え、仲間と共に笑い、共に成長しながら、
毎日が少しずつ楽しくなる未来へ。
未経験でも大丈夫。あなたの最初の一歩を、心からお待ちしています！

DESIGN ENGINEERING

イベント・セミナー開催中！



無料で自由に使える
学習＆カフェスペース開放中！

デザインエンジニアリング 検索

URL：https://design-engineering.jp/

カジュアル面談・エントリーは
こちらから！

LINE ID：@749gaovb