# 119.●システム導入プロセス

✨119.●システム導入プロセス

総務部門の社員など、非エンジニアが**システム導入のプロセス**で知っておくと役立つ技術知識を、専門的すぎない言葉で整理して説明します。

———

****■システム導入のプロセス**（非エンジニア向け）

システム導入は「業務をどう改善したいか」という目的から始まり、最終的に「安定して使える状態」に持っていくまでの流れを段階的に進めます。

代表的には **7つのステップ** があります。

———

**① 要件定義（何のために導入するかを決める）**

**• 目的を明確にする**

• 例：「勤怠管理を自動化したい」「請求書作成の手間を減らしたい」

**• 必要な機能を整理する**

• 必須機能（なくては困るもの）と、できれば欲しい機能を分ける

**• 利用者や運用体制を考える**

• 誰が使うのか（総務、全社員、経理…）

• 管理は誰がするのか（社内か、ベンダーか）

**★ 非エンジニアに重要なのは「業務の現状と課題を正しく伝えること」** です。

———

**② ベンダー選定（誰にお願いするか）**

**• 複数の業者に相談・見積を取る**（RFP＝提案依頼書を作ることもある）

**• 比較ポイント**

• コスト（初期費用、ランニング費用）

• サポート体制（問い合わせや障害対応）

• 実績や信頼性

• **契約形態**（買い切り、サブスクリプション、クラウドサービス など）も確認

★ **総務部門はコスト感とサポート体制をよくチェックすることが大切** です。

———

③ **設計（どう動かすかを決める）**

• **システムの具体的な使い方や画面イメージ**をベンダーとすり合わせる

• **既存の業務フローとの整合性**を確認する

• 例：給与システムと勤怠システムのデータ連携が必要か？

★ **業務の流れに合っているかを現場目線で確認する役割**が求められます。

———

④ **開発・設定（作る／設定する）**

• パッケージソフトやクラウドなら **設定作業**

• オーダーメイドなら **開発作業**

• 進捗確認の打ち合わせに参加することもある

★ **「専門用語が多すぎないか」「操作が複雑でないか」をチェックすること**が非エンジニアの重要な視点です。

———

⑤ **テスト（ちゃんと動くか確認）**

• **動作確認**：仕様通りに動いているか

• **業務テスト**：実際の業務シナリオで試す

• **ユーザーテスト**：使いやすいか、入力ミスしにくいか

★ **実際に利用部門が触ってみることが非常に大切**です。

（現場での「使いにくい」はシステム失敗の大きな原因になります）

———

⑥ **導入・展開（現場で使い始める）**

**• 社員への説明・マニュアル整備**

**• 教育・トレーニング**（簡単な説明会や操作動画）

**• 導入後のサポート体制**（問い合わせ窓口の確認）

**★ 「全員が迷わず使えるか」を意識すること**が重要です。

———

⑦ **運用・改善（安定稼働＆見直し）**

**• トラブル対応・問合せ対応**

**• 定期的な見直し**（業務の変化や法改正への対応）

**• コスト管理**（ランニング費用、追加ライセンス）

**★ 「ちゃんと業務改善につながっているか」を総務部門が確認する役割**があります。

———

非エンジニアが特に意識すべきポイント

1. **目的と課題を正しく伝える**（IT用語より自社の困りごとを明確に）
2. **コストとサポート体制を理解する**
3. **現場で使いやすいかを試す・確認する**
4. **導入後の社員教育・サポートを準備する**

———

■**リスク管理**（非エンジニア向け解説）

システム導入の各ステップには、必ず「つまずきやすいリスク」があります。

非エンジニアが理解しておくと「なぜこの確認が必要なのか」が見えやすくなります。

———

システム導入プロセスと主なリスク

**① 要件定義**

**リスク**

• 目的や課題が曖昧で「なんとなく便利そうだから導入」となる

• 必要な機能を洗い出せず、導入後に「この機能が足りない」と気づく

• 実際の利用者（現場社員）の声が反映されない

★ **結果**：システムがあっても業務改善にならない。

———

**② ベンダー選定**

**リスク**

• 「価格」だけで決めてしまい、サポートや信頼性が不足する

• 見積もりに含まれていない追加費用が後で発生（導入支援、カスタマイズなど）

• 提案書の専門用語を理解しないまま契約してしまう

★ **結果**：思ったより高額になり、サポートが弱いベンダーに縛られる。

———

**③ 設計**

**リスク**

• 業務フローとの整合性を見逃し「システムに合わせて業務を無理に変える」ことになる

• データ連携や帳票フォーマットが合わず、手作業が残る

• 非エンジニアが口を出さないため、使い勝手が二の次にされる

★ **結果**：現場が「前のやり方の方がマシ」と感じてしまう。

———

**④ 開発・設定**

**リスク**

• ベンダー任せにして進捗や内容を確認しない

• 「標準設定」で導入され、実際の業務に合わない

• 仕様変更や追加要望でコスト・納期が膨らむ

★ **結果**：完成しても「こんなはずじゃなかった」となる。

———

⑤ **テスト**

**リスク**

• ベンダーの動作確認だけで済ませ、現場テストを怠る

• 想定外の入力や利用ケースを試さない（例：誤入力時の挙動）

• 短期間で形式的に終わらせてしまう

★ **結果**：導入後にトラブル多発 → 業務が止まる。

———

⑥ **導入・展開**

**リスク**

• 操作説明やマニュアルが不十分 → 社員が使いこなせない

• 導入時に社内の抵抗感（「面倒」「慣れない」）が強く、定着しない

• サポート窓口が曖昧で、トラブル時に誰に相談すればいいか分からない

★ **結果**：システムが形骸化して結局使われなくなる。

———

⑦ **運用・改善**

**リスク**

• 運用ルールが整備されず、属人的になる

• ランニングコストが予想以上にかかる（追加ライセンス・保守費用）

• 法改正や業務変化に追従できず、時代遅れのシステムになる

• ベンダー依存が強すぎて、自由に改善できない

★ **結果**：使い続けるほど不便＆コスト負担が増す。

———

まとめ：非エンジニアが特に注意すべきリスク

• **要件定義の甘さ**（導入目的が曖昧）

• **コスト見積もりの盲点**（導入費用だけで判断）

• **現場視点の不足**（使い勝手の確認を怠る）

• **教育と定着不足**（導入しても使われない）

———

****■システム導入の契約形態**（非エンジニア向け解説）

契約形態は大きく分けて **「どうお金を払うか」** と **「誰がどこまで責任を持つか」** で整理できます。

———

**① 契約の基本タイプ**

**1. パッケージ買い切り型**

• **内容**：ソフトウェアを一度購入し、社内PCやサーバーにインストールして利用。

• **特徴**

• 初期費用は高いが、その後の月額費用は少なめ

• 保守やアップデートは別契約になることが多い

• **リスク**

• 数年後に古くなり、追加費用が必要になる

• バージョンアップ時に再度大きな出費がかかる

———

**2. ライセンス契約型**

• **内容**：利用者数や端末数ごとにライセンスを購入し、使用権を得る。

• **特徴**

• 人数が増えると費用が増える

• 契約更新が必要な場合が多い

**• リスク**

• 契約違反（人数超過利用など）でペナルティの可能性

• ランニングコストが見えにくい

---

**3. サブスクリプション型（SaaS型・クラウド型）**

**• 内容**：月額・年額で利用料を支払い、インターネット経由で利用。

**• 特徴**

• 初期費用が安く、すぐ使える

• バージョンアップや保守はサービス側が対応

• 利用人数に応じて従量課金されることが多い

**• リスク**

• 長期的には総費用が高くなる可能性

• ベンダーがサービスを終了すると利用できなくなる

• データがクラウドに保存されるため、セキュリティや解約後のデータ移行に注意

---

**4. 開発請負契約（オンプレミス・スクラッチ開発）**

**• 内容**：自社向けにゼロから開発を依頼する。

**• 特徴**

• 要望に沿ったカスタマイズが可能

• 納品後はシステムを「自社資産」として所有できる場合が多い

**• リスク**

• 初期費用・期間が大きい

• 要件変更で追加費用が膨らむ

• ベンダーに依存しやすい（その会社にしかメンテできないケースが多い）

———

**② 契約方式（費用・責任の決め方）**

**1. 請負契約**

• **内容**：「完成したもの」を納品する契約

• **特徴**

• ベンダーが完成責任を負う

• 仕様通りでなければ無償修正を要求できる

• **リスク**

• 要件定義が曖昧だと「仕様外」とされて追加費用が発生

• 完成するまで使えない（柔軟性が低い）

———

**2. 準委任契約**

• **内容**：一定の作業時間や人員を提供してもらう契約（成果ではなく「労務提供」に対して支払い）

• **特徴**

• 柔軟に対応できる（途中で方向転換可能）

• 大規模開発やアジャイル開発でよく使う

• **リスク**

• 成果物保証がなく「時間を使った分だけ支払う」ため、進捗管理が甘いと費用が膨らむ

———

**3. 保守・サポート契約**

• **内容**：導入後の運用支援、障害対応、アップデート対応など

• **契約形態**

• 定額契約（毎月固定額でサポート）

• チケット制（問い合わせ回数ごとに課金）

• 従量課金制（発生した工数分だけ支払う）

**• リスク**

• サポート範囲が曖昧だと「それは契約外です」と追加費用が発生

• 緊急対応の時間保証（SLA）がないと業務が止まる可能性

———

**③ 非エンジニアが見るべきチェックポイント**

1. **契約期間**：自動更新か？解約条件は？
2. **費用構造**：初期費用・月額費用・追加費用（カスタマイズ、人数増、データ移行）
3. **サポート範囲**：障害時、どこまで無償で対応してもらえるか
4. **データの帰属**：クラウド型の場合、解約時にデータをどう受け取れるか
5. **責任分担**：ベンダーと自社のどちらがどこまで責任を負うか

———

まとめると：

**• パッケージ買い切り** → 初期費用高いが長期的には安く済むことも

**• サブスク型** → 初期安いが長期費用は高くなりがち、解約・データ移行に注意

**• 請負契約** → 完成責任はベンダーだが、仕様漏れは自社負担

**• 準委任契約** → 柔軟だが費用が読みにくい

———

**■システム導入における契約形態の比較表**

この表を見ながら、非エンジニアの方は

• 「費用の支払い方はどうなるか？」

• 「解約やサービス終了の時にデータをどうするか？」

• 「サポートはどこまで無料で受けられるか？」

を必ず確認するのがポイントです。

---

## システム導入における契約形態の比較表

| 契約形態 | 概要 | メリット | デメリット・リスク | 向いているケース |
|---|---|---|---|---|
| **パッケージ買い切り型** | ソフトを一度購入して社内で利用 | ・長期的には費用を抑えやすい<br>・資産として所有できる | ・初期費用が高い<br>・バージョンアップ時に追加費用<br>・古くなるリスク | 長く同じ業務フローで使う予定のシステム |
| **ライセンス契約型** | ユーザー数や端末数ごとに使用権を購入 | ・必要な分だけ導入できる<br>・利用人数に応じて調整可能 | ・人数が増えると費用が膨らむ<br>・契約違反リスク（人数超過など） | 利用人数が安定している業務 |
| **サブスクリプション型（SaaS/クラウド）** | 月額・年額払いでクラウドサービスを利用 | ・初期費用が安い<br>・すぐ使える<br>・自動で最新バージョンに更新 | ・長期的に費用が高くなる可能性<br>・サービス終了リスク<br>・解約後のデータ移行が課題 | 短期間で導入したい場合、在宅勤務や多拠点利用 |
| **開発請負契約（オンプレ・スクラッチ開発）** | ゼロから自社専用に開発を依頼 | ・業務に完全にフィット<br>・自社の資産になる | ・初期費用・期間が大きい<br>・要件変更で追加費用増<br>・ベンダー依存になりやすい | 独自業務が多く、既存パッケージでは対応できない場合 |
| **請負契約**（成果物納品型） | 「完成物」を納品してもらう契約 | ・ベンダーが完成責任を持つ<br>・仕様通りでなければ無償修正可 | ・要件定義が曖昧だと追加費用発生<br>・柔軟性が低い | 要件が明確に決まっているシステム |

| 契約形態 | 概要 | メリット | デメリット・リスク | 向いているケース |
| --- | --- | --- | --- | --- |
| **準委任契約**（作業時間提供型） | 作業時間や人員を提供してもらう契約 | ・柔軟に仕様変更できる<br>・アジャイル開発に向く | ・成果物保証がない<br>・管理が甘いと費用が膨らむ | 要件が流動的なプロジェクト |
| **保守・サポート契約** | 導入後の運用・障害対応を依頼 | ・安心して運用できる<br>・アップデートや障害対応を任せられる | ・サポート範囲が曖昧だと追加費用発生<br>・SLAが不十分だと業務停止リスク | 中長期的に安定運用したい場合 |