４．開発プロセス

４－１　開発プロセスについて

（１）システム開発の基本的な考え方

情報システムの開発プロセスは、計画あるいは調達した情報システムを具現化する工程であり、利用する段階、すなわち情報システムの運用・評価プロセスにおいてどれだけ効果をもたらすかは、このプロセスに依拠している部分が多い。開発プロセスの質が高ければ、運用・評価プロセスにおいて障害の発生や、保守作業等を最小限にとどめることが可能であり、利用者側の使い勝手も向上し、利用促進も図られる。

一方、情報システムに関わる技術の高度化や複雑化が進んでいること、過当競争が進み十分な開発能力がないまま応札するベンダーが出てきていること等を背景に、昨今、開発プロセスにおいてシステム開発に失敗する事例も散見される。同様の情報システムの開発経験があるベンダーにおいても、必ずしも経験者が対応するとは限らないし、既存の情報システムとの連携等を考慮すると、当該情報システムの経験だけでなく、幅広い見識が必要になる。

情報システムの開発に際しては、通常、プログラムと呼ばれるソフトウェアを作成したり、多様なハードウェアやネットワーク機器を活用したりするための様々な専門言語が用いられる。このため、行政職員の中にはベンダーにシステム開発を一任するような事例も少なくないが、このことが返ってリスクを高める。委託する場合においても、委託先とコミュニケーションを密にし、適切な進捗管理を行い、開発プロセスを着実に推進するように努めることが不可欠である。

（２）開発プロセス編の構成

情報システムの開発プロセスにおいて実施すべき作業は、実際に情報システムを開発するという「システム開発作業」と、その進捗管理を行う「プロジェクトマネジメント」に大きく分けることができる。ただし、「システム開発作業」に関しては、ほとんどの場合、ベンダー等の委託先が行うことになり、自治体において行うべき作業は、ベンダーが行うシステム開発

作業のチェックや管理となる。技術的に詳しい知識を有していなくても、システム開発がどのような流れでどのような事を行うかを把握しておくことで、ベンダーとのコミュニケーションがスムーズになり、開発プロセスの質を高めることが可能になる。また、自治体職員においても理解できるような説明を要求すること自体も重要な業務となろう。

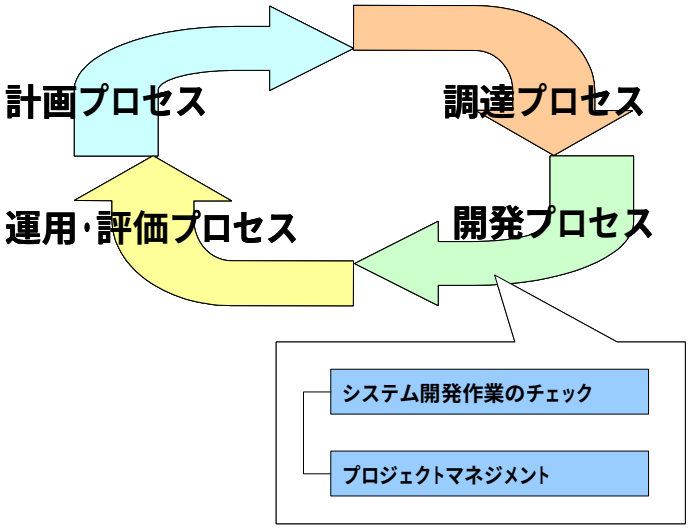


図５－１５　開発プロセスの構成

４－２　システム開発作業の精査

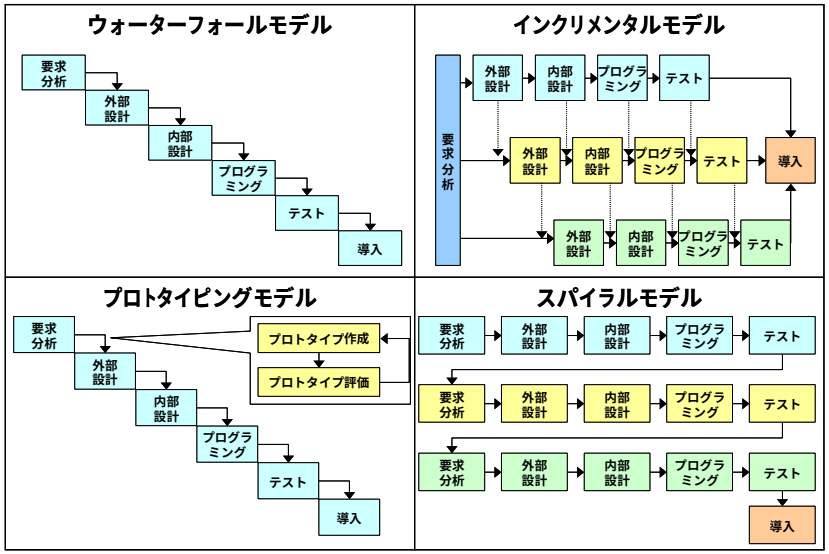
（１）システム開発作業の流れ

情報システムの開発を行う手順に関しては、主に表５－１７、１８に示す四つの手法が存在するが、本稿ではシステム開発において基本となる図５－１６に示すウォーターフォールモデルを用いて各作業工程の解説を行うこととする。ウォーターフォールモデルとは、文字通り水が滝から流れ落ちるように、システム開発プロセスを順番に実施、完了して行くことから付いた名であり、古典的な開発手法ではあるが、未だに基本として使われている。他のモデルは基本的にウォーターフォールモデルを応用した手法であり、導入する情報システムの特性を考慮して適宜、手順を組み替え、

表５－１７　情報システムの主な開発手法

	ウォーターフォールモデル	インクリメンタルモデル	プロトタイピングモデル	スパイラルモデル
特徴	・要求分析から設計、テスト、導入まで、上流の作業工程が完了した後に下流の作業工程に移っていく手法 ・規模の大きなシステム開発に適合	・要求分析において全体を定義した後、いくつかのフェーズに分けて順次開発作業を進め、先に進めたフェーズを後のフェーズに反映する手法	・要求分析の後、簡単なプロトタイプを作成することで要求を具体的に明確化し、その後の作業手順を推進する手法 ・規模の小さなシステム開発に適合	・全体をいくつかのフェーズに分け、ウォーターフォールモデルを短期間で繰り返し行う手法 ・開発期間の短縮が目的
メリット	・全体構造の把握が容易	・重要な部分を先行開発することで全体的なリスクを軽減 ・全体設計を待たずに開発を推進	・下流の作業手順における手戻りの減少・利用者側の満足度向上	・プロトタイプ作成と同様の効果を創出 ・下流の作業手順における手戻りが容易
デメリット	・下流の作業工程でないと明確にならない事項の存在 ・下流の作業手順において手戻りが発生した場合の工数増大 ・要求分析が長期化する可能性	・全体の進捗管理の複雑化 ・手戻りが発生した場合の対応が複雑化 ・フェーズの前後関係の変化による遅延	・プロトタイプ作成工数の増大 ・利用者ニーズの増大	・最終的な導入完了時点の見極めが困難 ・フェーズの前後関係の変化による遅延

表５－１８　開発手法のイメージ



最適な手法を選択することが必要である。筆者の限られた経験を考慮すると、時間を要するスパイラルモデルはほとんど使われることなく、最近ではプロトタイピングモデルとパッケージソフトを用いて開発期間を短縮する事例が増えてきているようである。システム開発はベンダー等に外部委

託することになると予想されるが、情報システムの開発担当組織では手順を委託先と協議の上、決定し、各作業工程が適切に遂行されているかを管理しなければならない。ただし、計画／調達プロセスにおいて既に実施されている作業工程、あるいは情報システムの特異性から必要ないと考えられる作業工程（新規開発の場合は「情報システムの移行」は必要ない）に関しては適宜、省略することになる。開発プロセスで検討する事項の中には計画プロセスで作成した開発計画素案と同様の事項も少なくないが、委託ベンダー等との協議の上、詳細まで落とし込み、それを確定させる点が計画プロセスと異なる。

なお、最近ではパッケージソフトを用いて情報システムを構築する事例も増えており、これに関しては各作業工程の解説の後に別途説明する。

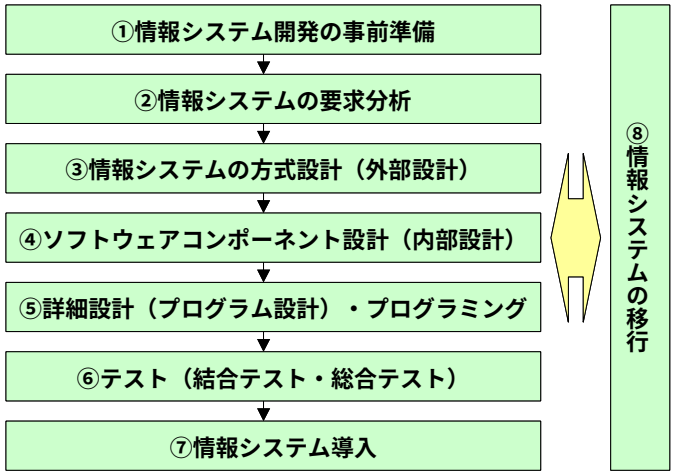


図５－１６　本稿で基本とするシステム開発の手順

（２）情報システム開発の事前準備

計画プロセスで作成した開発計画素案、調達プロセスで作成した RFP 等を基に、より詳細を定めた情報システムの開発実施計画を作成しなければならない。ベンダー等に開発を委託する場合、開発実施計画について委託先と事前に協議することで実効性のあるものとし、その内容について合

意を形成しておくことが不可欠であり、開発実施計画の内容そのものが契約資料となる場合もある。

システム開発の手順（ウォーターフォールモデルでない可能性もある）や、各作業項目の具体的な内容、遂行に要する工数や期間及びスケジュール等を明確にするとともに、後述のプロジェクトマネジメントの具体的な内容に関しても記述する。開発工数に関しては、正確性にも限界があるが、あまりにも精度が低い見積が委託先から出てこないよう表５－１９に示すような見積手法の概要を把握しておくことが望ましい。現状では類推法を用いる場合が多いように見受けられるが、先進事例や類似事例を踏まえた他の手法の活用についても可能性を検討したほうが良い。また、計画プロセスで検討した政策的な側面に関しても今一度見直しを行い、当該情報システムの政策的な位置付け、目的や目標を確認し、委託先企業等ともこれらの情報の共有化を図る。

表５－１９　主な開発工数見積手法

手法	概　　要
類推法	過去に開発した情報システムから工数を類推する。情報システムの規模、対象業務、プログラミング言語、開発ツール等から類似事例を探す。
標準タスク法	開発プロセスに係る個々の作業（タスク）別に標準的な工数算定基準を決めておき、タスクごとに算出した工数を合計する。
COCOMO	（COnstructive COst MOdel）の略。ソフトウェアのステップ数（ソースコードの行数）から過去の類似事例を参考に開発工数を算出する。
ファンクションポイント法	外部入力、外部出力、外部インターフェースファイル、内部論理ファイル、外部照会の五つの機能数を計算し、これに複雑度、システム特性による影響度をかけ合わせてファンクションポイント（FP）を算出する。FPを開発工数に変換するには経験に基づくが算定式が必要となる。

一方、開発作業を行う要員、その役割や責任・権限を規定した組織体制及び開発環境についても明らかにし、必要な開発環境を整備する必要がある。開発環境としては、コンピュータ等のハードウェア環境、OS や開発ツール等を含むソフトウェア環境、および LAN 接続等のネットワーク環境がある。開発環境に関しては、情報システムの実際の運用状況にできるだけ近い環境とすることが望ましく、オンラインで処理する情報システム

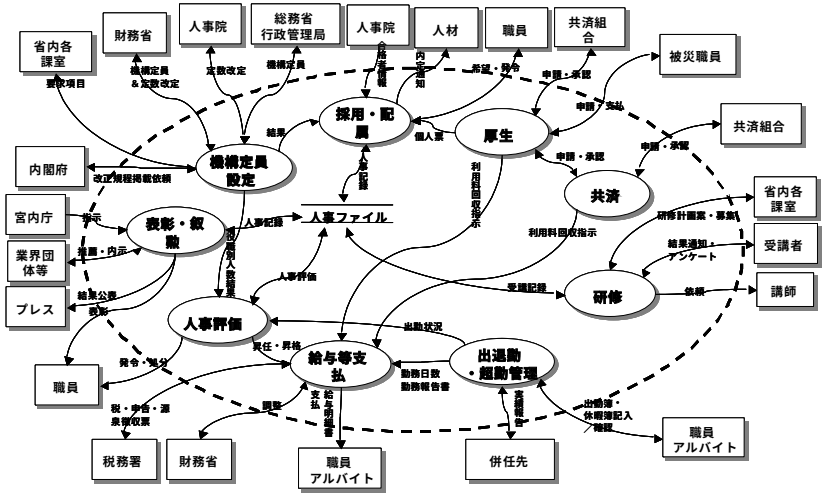
では同様のネットワーク環境が必要になる。特に出先機関等の遠隔地での利用が想定される情報システムでは、本庁と出先機関を結ぶ通信回線と同等の環境でテストすることが不可欠である。また、ハードウェア等が機能するための電源、あるいは作業者が開発を行うために必要なスペース等を確保することも忘れてはならない。

（３）情報システムの要求分析

計画プロセスで検討した情報システムの範囲、要件、概要や、RFP 等を基に、より詳細なシステム要求事項を分析し、要求仕様書[1]として文書化する必要がある。なお、この要求分析と後述の外部設計に関しては、調達プロセスと前後することもある。これらの概要設計作業を別途行った後、それに基づいて内部設計以降の開発作業を調達することも少なくない。

要求分析に際しては、システム化の範囲となる業務について調査を行い、業務フロー、業務量、業務に用いている帳票、業務遂行上の問題点や課題、利用者ニーズ等を抽出する。ただし、利用者毎にニーズの内容が異なる場合もあるため、抽出したニーズをそのまま要件とすることは望ましくない。そこで、業務フローや情報の流れに注目して、求められる機能要件を定義することが必要である。昨今では、図５－８に示したような業務フロー図だけでなく、業務処理に使われる情報の流れを模式的に示した機能情報関連図（DFD：Data Flow Diagram）等が要求分析に用いられる。DFD は四つの記号を使って情報システムにおけるデータの流れや処理を表す手法であり、現行の DFD を再構成した理想の DFD から要求事項を定義することができる。DFD は前述した WBS 同様、業務毎に階層化、細分化する必要があり、細分化した DFD を論理化（スピード向上のためファンクション内の流れを組み替えること）、抽象化（分類のための集合に論理的なつながりを表す適切な名称を付けること）という作業を経て理想とする姿に再構成（最適化）することになる。

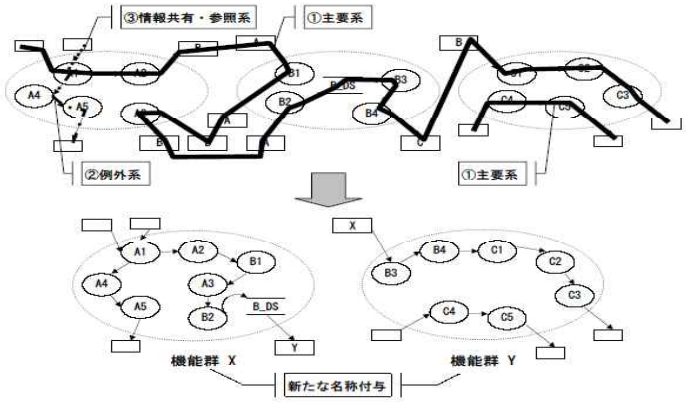
[1] RFP を要求仕様書と訳す場合も多いが、本稿では調達において提案を依頼する RFP と情報システムの要件を確定する要求仕様書を分けて考えている。



記号	説明
ファンクション	ファンクションを示す （「～する」という動詞表現になる）
ターミネータ	対象とするシステム範囲から見た外部環境で、外部のシステム、人、組織など、情報の発信地や受信地を示す
情報の流れ	情報流れを記述する （情報には情報名を付けて記入する）
ファイル	情報の一時的な貯蔵であり、情報の滞留を示す （一本線のファイルは、ファンクション内に限定しているファイルを示す）

出典：ITアソシエイト協議会「業務・システム最適化計画について」

図５－１７　DFD の例



出典：各府省情報化統括責任者（CIO）連絡会議事務局「業務・システム最適化計画策定指針」

図５－１８　DFD の再構成の例

要求仕様書とは別に、要求分析等を踏まえ業務を実際にどのような手順で処理するか、その手順を規定した業務運用手順を作成する必要がある。業務運用手順は、以降の作業工程における検討を踏まえ随時、精査を行い、最終的には運用マニュアルとなる。

また、業務フロー図や DFD から導き出す要求事項は個々の業務処理に求められるものであり、実際には情報システムそのものの運用という業務が存在する。したがって、運用業務や情報システムの移行を想定した要求事項を抽出するだけでなく、信頼性、効率性等、情報システム全体に求められる要求事項も合わせて要求仕様書に整理することが必要である。要求仕様書は、計画プロセスで整理した表５－８や表５－９のような各要件を詳細化、確定化した内容となる。

（４）情報システムの方式設計（外部設計）

要求分析を基に、外部設計として情報システムの大まかな設計を行う。外部設計は、利用者の立場から認識できる情報システムの外枠の部分を対象としており、以下に示すような事項の設計が求められる。外部設計は、要求仕様書を踏まえ、かつ、利用者の意見を聴きながら進めることが重要であり、発注者である自治体が主体的に関わることが不可欠である。

①システム構成

情報システムにおいてどのような処理形式を採用するかについて決定する必要がある。大きくは、汎用機等による集中処理と、C/S のような分散処理に分けることができ、昨今では Web をインターフェースとし、ブラウザをクライアントとする分散処理が主流になってきている。しかし、すべての情報システムにおいて分散処理が望ましいわけではないので、業務等の特性を考慮し、処理形態を選択することが望ましい。また、C/S を構成するサーバにはウェブサーバ、ファイルサーバ、プリントサーバ、アプリケーションサーバ、データベースサーバ等、多様な形態があり、C/S で情報システムを構築する際には要求事項に応じたサーバ構成を検討すること必要である。加えて、昨今では、情報システム間の連携や共通機能を提供

するミドルウェアの活用も進んでおり、クライアントとサーバによる単純な二層構造ではなく、三層構造になることも少なくない。

処理形式を踏まえ、システムを構成するハードウェア、ソフトウェア、ネットワークの大まかな構成を決定しなければならない。ソフトウェアとしては OS やアプリケーション等、ハードウェアとしてはコンピュータの処理能力等が検討事項となる。

また、情報システムの規模が大きくなると構造が複雑化し、開発が難しくなる。そのため、情報システムをいくつかのサブシステムに分割することも必要である。計画プロセスの図５－１０のように情報システムを構造化するとともに、各サブシステムの機能や、上位システムと他のサブシステムとの間のデータの受け渡し方法等についても明らかにする。加えて、サブシステムを結合して情報システムとする際のテスト要求事項を定義する。

②ユーザーインターフェース

情報システムでは、利用者がユーザーインターフェースを介して操作したり、情報の入出力を行うことになる。そのため、ユーザーインターフェースの使い勝手が悪いと、業務の効率が低下したり、利用者が情報システムを利用しなくなる可能性もある。特に中心となるユーザーインターフェースである端末表示画面は、直感的で分かりやすく、迅速に操作を覚えられることが望ましい。そのため、画面上のボタンやメニュー等のレイアウト、メニューや画面の階層構造に関して、利用者参加のもと慎重に設計を行うことが必要になる。業務を処理するための帳票等が事前に存在する場合には、それらを参考に入出力画面を設計することも有効である。

前述したように Web をユーザーインターフェースとした情報システムが主流になってきているが、Web の表現にも限界があることに十分に留意する必要がある。このような問題に対応するため、リッチクライアントと呼ばれるソフトをブラウザに入れる事例も出てきているが、それではブラウザを使うことのメリット（保守の容易さ）が半減してしまう。

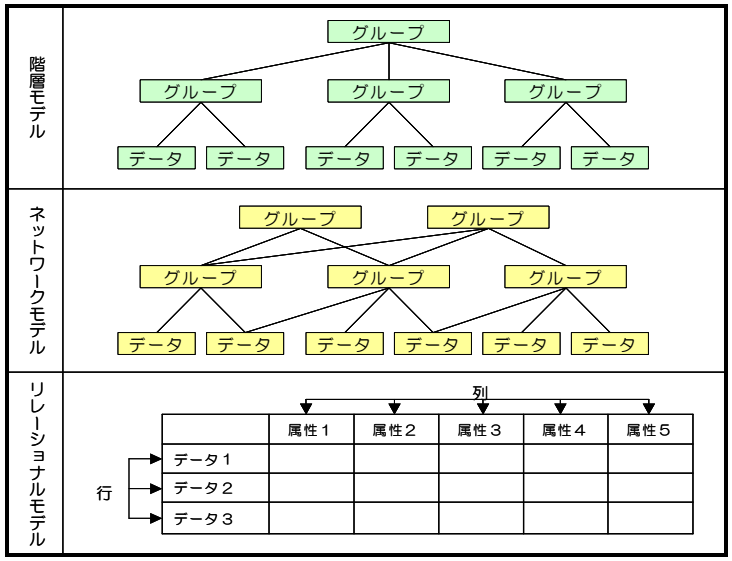
③データベース

DFD 等を基にデータベースの構造をデータモデルとして定義することが望ましい。データモデルには階層モデル、ネットワークモデル、リレーショナルモデルの三つが存在する。

階層モデルは関連性のあるデータを一つのグループに集約し、グループを更にグルーピングすることでツリー状の階層構造を形成するモデルである。社会における組織等の階層構造に類似するため直感的で分かりやすいが、データの関連性が単一であるため、あるデータには一つの経路からしかたどり着くことができない等の問題点を持つ。

ネットワークモデルは、階層モデルの問題点を改善したモデルであり、データが上位のグループに対して複数の経路で関連付けることが可能である。

表５－２０　三つのデータモデル



リレーショナルモデルは、データ同士の関係性を表形式で表現するモデルであり、他の二つのモデルと異なり、データの独立性が高い。1 行が一

つのデータを表し、列が属性を表すことになる。他の二つのモデルでは、データの検索や更新を行う際に上位との関連性を意識する必要があるが、リレーショナルモデルではこのような関連性を意識することなく検索や更新が行える。

上述のような特性から通常は情報システムで活用するデータベースはリレーショナルモデルが使われている。リレーショナルモデルを用いたデータベースの外部設計では、DFD のファイルごとにリレーショナルモデルの表を作成し、これを正規化する必要がある。正規化とは、データの関連性をできるだけ簡潔にして、データの追加、更新、削除が容易な形に加工することである。

④セキュリティ

セキュリティのリスクが顕在化していることを考慮すると、セキュリティという切り口で外部設計を行うことも重要である。情報システムのセキュリティに関しては、セキュリティポリシーを定めている自治体では、これに基づいてセキュリティに関する機能や要件の設計を行うこととし、必要に応じて総務省「地方公共団体情報セキュリティ管理基準」等を参考とすることが望ましい。

セキュリティを高め情報システムの機密性、保全性、可用性を維持するためには、表５－２１に示すようや四つの機能が必要であり、情報システムの特性を考慮し、必要な機能の設計を行うこと求められる。

表５－２１　情報システムに求められるセキュリティ機能

機能	内容	例
抑制機能	職員等による不正行為の発生を抑制する	職員認証、操作ログ管理　等
防止機能	障害や職員による事故等の発生を防止する	利用者の操作支援、ハードウェアの二重化　等
検知機能	エラー、事故、不正等を検知する	不正アクセス検知、遠隔モニタリング　等
回復機能	障害が起こった常井に速やかに復旧する	バックアップ、遠隔操作　等

（５）ソフトウェアコンポーネント設計（内部設計）

外部設計が利用者から見える部分の設計であるのに対して、内部設計は利用者からは見えない情報システムの中身の設計に当たる。つまり、外部設計で実現するために、情報システムの中身にどのような構想や機能が求められるかを示すものである。具体的には以下に示すような事項の設計を行うことになるが、専門的な知識を要するため、ベンダーが主体となって進めることになろう。ただし、ここでベンダーに「丸投げ」するのではなく、設計内容についてコミュニケーションを図り、できるだけ内容を理解しておくことが重要である。

①コンポーネント設計

外部設計で決定したシステム構成を基に、情報システムあるいはサブシステムを、更に細かなプログラム単位であるソフトウェアコンポーネントに分解する。コンポーネントはプログラムの特性や機能等から複数の種類に分けることが可能であり、通常、異なるコンポーネントの組合せによって（サブ）システムを構成することになる。（サブ）システムを構成するコンポーネント間における情報の受渡し、あるいは各コンポーネットと外部

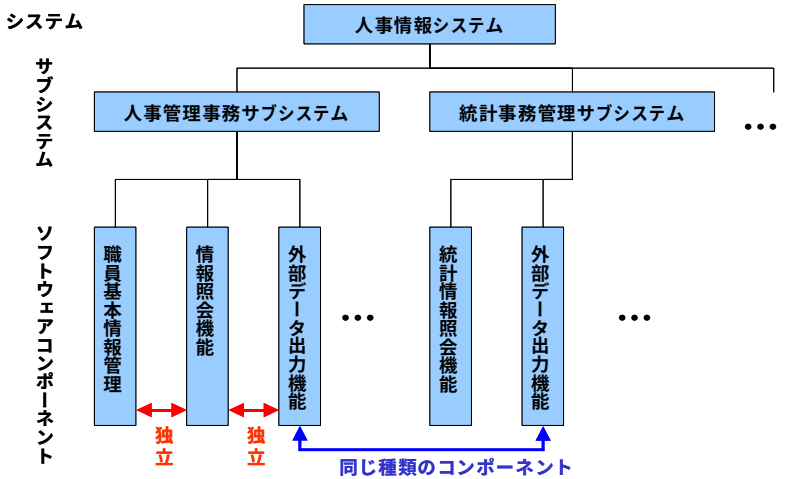


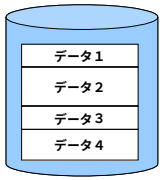
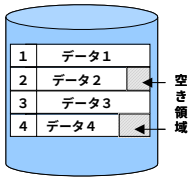
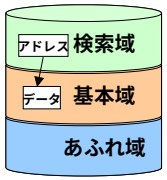
図５－１９　コンポーネント設計の例

システムとの情報受渡し方法について設計し、コンポーネントを結合する際のテスト要求事項を定義する。なお、各コンポーネントは機能的に互いに独立するように設計されることが望ましく、複数のサブシステムで同じ種類のコンポーネントを活用することもある。

②データベースの内容

正規化したデータモデルを基に、データを格納するファイルの編成方法やレイアウトについて設計する。代表的なファイル編成方法としては、表５－２２に示すような三つの方法が挙げられる。この三つ以外にも複数の順編成ファイルから構成され、そのディレクトリ（登録簿）を持つ区分編成という編成方法や、三つを仮想化して統合した仮想編成方法等がある。また、ファイルは通常、レコード（1 件のデータ）の集まりとして構成されており、レコードはフィールドと呼ばれるデータ構成項目に分けることができ、これらのレイアウトを決定することになる。

表５－２２　主なデータ編成方法

編成方法	順編成	直接編成	索引編成
内　容	記憶装置の先頭から順にデータを記録してファイルとする方法。データの操作をファイルの先頭から行う必要があるため、データの更新や削除が頻繁に行われるデータベースには適さない。	個々のデータに対して記録するアドレス情報を付加する方法。アドレスを指定することでデータごとの更新、変更が可能である反面、データの後ろに空き領域が発生する可能性がある。	索引行域、基本域、あふれ域の三つから成る編成方法。基本域にデータを格納し、そのアドレス情報を索引域に記録する。あふれ域はデータが基本域からあふれた場合にのみ使う。
イメージ			

一方、データベースの物理的な側面に関しても検討する必要があり、ファイル編成方法や業務内容等をもとにデータベースの容量やアクセス頻度等を決定する。データベース容量が大きくなり、アクセス頻度が高いので

あればデータベースサーバの負荷が大きくなり障害等の原因になる。その場合、データベースの分散配置についても検討することになる。データベースの分割の仕方には表５－１８に示したリレーショナルモデルの行によって水平に分割する方式と、列によって垂直に分割する方式があり、データベースの利用方法を考慮して望ましい方式を選択する。また、利用者がこのようなデータベースの分割を意識せずに利用できるようにすることが望ましい。

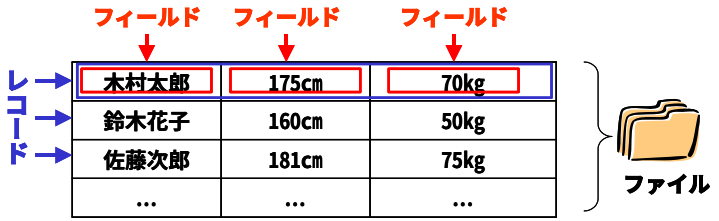


図５－２０　ファイル・レコード・フィールドの関係

（６）詳細設計（プログラム設計）・プログラミング

内部設計が終わったら、ソフトウェアを作成するためのプログラミングを行うことができるよう、内部設計を基に、より詳細化したプログラム設計を作成することになる。プログラムに関しては、専門的な知識を要する部分であるため分かりにくいが、細かな進捗管理を行うためには、その作業の構成を把握しておくことが望ましい。また、プログラム設計に基づいてプログラミングを行い、合わせてデータベースの開発を実施することが求められる。

①プログラム設計

ソフトウェアコンポーネント（プログラム）をプログラミングに適切なより詳細な機能単位であるユニット（モジュール）に落とし込む。コンポーネント単位でもプログラミングすることは可能であるが、プログラミングの量が多くなるため、複雑化し途中で混乱することが想定され、複数の人数で作業することも難しくなる。独立性の高いユニットに分割すること

で、ユニットごとに完結した分かりやすいプログラミングが可能になり、プログラムの修正等も容易になる。ユニットの分割方法には表に示すようなものがあり、分割後は分割結果が適切かどうかを評価しなければならない。また、上位概念と同様に、ユニット間における情報の受け渡し方法について設計し、ユニットを結合しプログラムとする際のテスト要求事項を定義する。

表５－２３　ユニット分割の方法

分割方法	内　　容
STS分割	データの流れに注目し、入力処理機能（Source）、変換処理機能（Transform）、出力処理機能（Sink）に分割する。
TR分割	処理（Transaction）の種類により分割する。STSではうまく分割できない処理も存在する。
共通機能分割	共通する処理機能を取り出して別のユニットとして分割する。
ジャクソン法	入出力のデータ構造の対応からプログラムを構造化する方法で、基本、連接、選択、反復の四つの構造を用いて表現する。

②プログラミング

プログラム設計に基づきコーディング（プログラムを記述する）によってソースコードと呼ばれるプログラムの原図を作成する。昨今注目を集めているオープンソースソフトウェアとはこのソースコードが文字通りオープンに（公開）され、誰でも改変、再配布が可能になったものである。コーディングを行う際には、プログラマーが個々に異なる形態で記述しないよう基本的な方針や守るべきルールを定めておく。コーディングによって作成されたソースコードはそのままのコンピュータ上で動かすことはできず、コンパイラあるいはインタープリタと呼ばれるソフトウェアを用いてコンピュータ上で稼働するオブジェクトコードと呼ばれる形態に変換する。実行可能になったものをプログラム設計のテスト要求事項に基づき、コンポーネント毎あるいはユニット毎に単体テストし、正しく動作するかを確認する。

プログラミングを進める一方で、内部設計やプログラムとの関連性を考慮し、データベースの開発も併行して行う。データベースに関しては専用

のデータベースソフトウェアを用いる場合が多いが、プログラミングによって開発を行う場合もある。

（７）テスト（結合テスト・総合テスト）

情報システムが稼働する段階において、欠陥が見つかり、運用に支障を来すことがないよう、入念なテストを行うことが不可欠である。

単体テストを終えたユニット、あるいはコンポーネントを結合し、内部設計に基づいてサブシステムとして正常に動作するか結合テストを行う。同様に結合テストを終えたサブシステムを結合し、ソフトウェア、ハードウェア、ネットワーク等を整備して、情報システムとして稼働するか総合テストを行う。結合テスト、総合テストはそれぞれ内部設計、外部設計においてとりまとめたテスト要求事項に基づいて実施されることになり、各テストと設計レベルの対応は図５－２１に示すとおりである。テスト結果に不具合がある場合は、対応する各段階において設計を見直すことも必要になる。

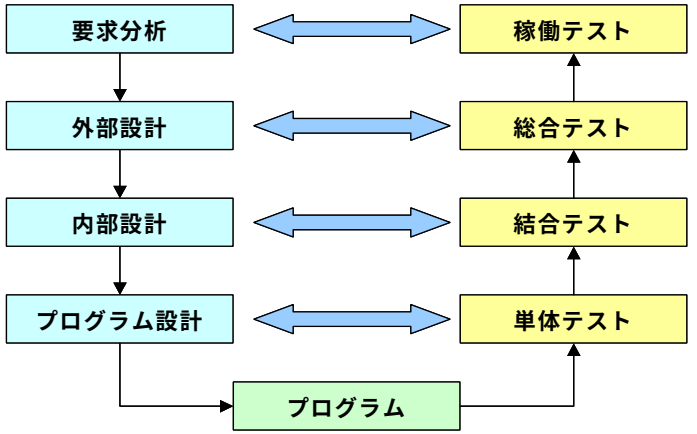


図５－２１　設計とテストの関係

テストの方式には、ホワイトボックス方式とブラックボックス方式がある。ホワイトボックス方式では、入力と出力だけでなく、その処理の仕方

が設計に基づいているかをチェックするもので、プログラム設計の単体テストに用いられる。一方、ブラックボックス方式は内部での処理の仕方にこだわらず、入力に対して求められた出力が行われるかをテストするもので、主に外部設計の総合テストに用いられる。この他、結合テストでは、構成で下層に位置するコンポーネントからテストしていくボトムアップ方式、上層からテストしていくトップダウン方式又は一度にすべてを結合してテストするビッグバン方式がある。

（8）情報システムの導入

総合テストを終えた情報システムを導入するため、運用環境を整える必要がある。運用環境は要求仕様書に沿って整備し、利用者参加のもと業務運用手順に基づいて実際の業務処理を行う等の稼働テストを実施しなければならない。また、業務運用手順や稼働テスト結果等を踏まえ、操作マニュアル、運用マニュアルを整備するとともに、利用者に対して情報システムの利用がスムーズに行えるよう必要な研修を提供することが不可欠である。最近では、開発中の情報システムとは別に研修用のサーバ等を構築し、早くから研修を行う事例も増えてきているようだ。

開発を委託している場合は、委託契約書及び要求仕様書、業務運用手順等をもとに、稼働テスト結果を踏まえ納品された情報システムの検収を行わなければならない。要求仕様書に基づいたシステム環境になっているかどうかだけでなく、品質要求が満たされているかどうかを検証する必要があり、一定の検査期間を設け、多様な業務処理形態や運用形態を試すことになる。また、情報システムに関する要求仕様書、各種設計書、テスト報告書、各種マニュアル等の書類一式を検査することも必要である。検収する際には、委託契約書に基づき瑕疵担保責任の範囲を再度確認するとともに、保守を委託する場合は保守の範囲も明確にしておかなければならない。

また、セキュリティポリシーを設けている自治体では、ポリシーに沿って当該情報システムの「セキュリティ実施手順書」を作成しなければならない。

開発の遅れや職員の業務負担等を理由に、この検収がきちんと行われて

いない場合が多く、運用・評価プロセスにおける問題の原因となっている。検収作業の厳格化を図ることは、開発プロセスにおける最も重要な課題の一つと言える。

（9）情報システムの移行

既存の情報システムがあり、これを刷新するために情報システムを開発したのであれば、導入に際して新規システムへの移行を行わなければならない。移行作業が業務処理に支障を来さないよう、移行計画を立てて、これに基づいて効率的に実施することが必要である。

データやプログラムの移行は、新規情報システムの開発やテスト等の作業工程において要求される適切なタイミングで行うとともに、必要に応じて、データやプログラム等を変換するためのツールの開発や調達も行うことが望ましい。

移行計画に基づき情報システムの切替、移行等のリハーサルを実際に行うことが望ましい。リハーサルに問題がなければ、計画どおりの日程で切替、移行作業を行い、移行終了後に作業結果の評価を実施する。情報システムの移行をスムーズに行うためには新旧両方の情報システムが並行稼働する形態が望ましいが、情報システムの設置スペース等を考慮する必要がある。並行稼働が無理な場合は、事前に移行できるデータ等をすべて用意しておき、切替時期にはハードウェア等の物理的な入れ替えを中心に作業することが望ましい。

なお、既存の情報システムをハードウェア部分の刷新を中心とした開発の場合、リバースエンジニアリングと呼ばれる手法を活用することも考えられる。リバースエンジニアリングとは、機械等を分解、解析し、その仕組みや構成を真似て互換製品を作ることを一般に言うが、ここでは既存の情報システムの設計情報を抽出して新たに構築する情報システムに活用することを指す。これによって一から設計する手間を省くことが可能であり、最新の技術や改良すべき事項を反映し、新たな情報システム開発を効率的にすることが期待できる。近年、汎用機等のレガシーシステムの維持費用が問題になっていることから、これをオープン系に移行するレガシー・マ

イグレーションが注目されているが、これにもリバースエンジニアリングが用いられている。

ただし、既存の情報システムのソフトウェアがオープンソースソフトウェアでなく、著作権が自治体にない場合には、著作権侵害になる可能性があるので配慮が必要であろう。ちなみに図５－１６に示すような通常の進め方をリバースエンジニアリングと対比する場合、フォワードエンジニアリングと呼ぶ。

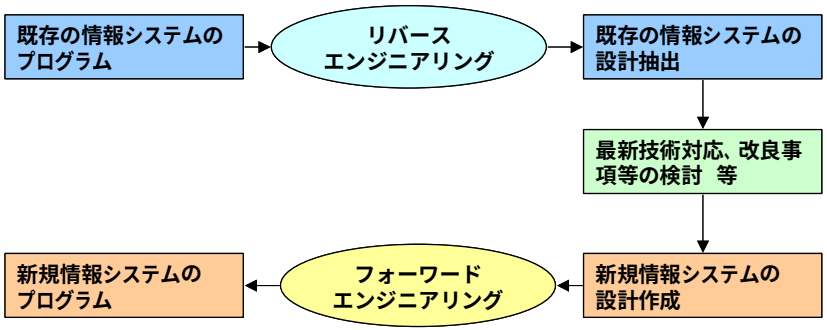


図５－２２　リバースエンジニアリングのイメージ

（１０）パッケージソフトの活用

上記で説明したような内容が一般的な情報システムの開発手順ではあるが、昨今では、情報システムを構成するソフトウェア部分が業務毎に標準化され、「パッケージソフト」として販売されている場合も多い。したがって、まったく新規の情報システム開発だけでなく、パッケージソフトを活用した情報システムの開発も選択肢として検討することが望ましい。パッケージソフトを活用すると、システム開発の期間や費用を小さくすることができる反面、細かな要求への対応が難しいと一般的に言われているが、自治体の業務ニーズに合致しているのであれば、パッケージソフトでも問題ない。ただし、ほとんどのパッケージソフトは著作権がベンダー側に帰属している点に留意し、二次利用等を考えている場合には逆にパッケージソフトを使わない方向で検討することもあり得る。

パッケージソフトをそのまま導入する際には、ハードウェアやネットワークと合わせて総合テストを行う程度で、開発作業はあまり発生しないが、実際にはカスタマイズを行う場合も多い。パッケージソフトをカスタマイズして情報システムを構築する場合は、図５－２３に示すようにカスタマイズ部分に上述の開発手順を適用することが望ましい。要求分析の結果とパッケージソフトの機能を照らし合わせてギャップ分析を行い、その違いを明らかにするとともに、追加・修正（カスタマイズ）を行う部分を決定する。その後は、追加・修正する部分に関しては、上述した内容と同じ作業を行い、最終的にパッケージソフト本体と追加・修正部分を結合してテスト、導入を行うことになる。

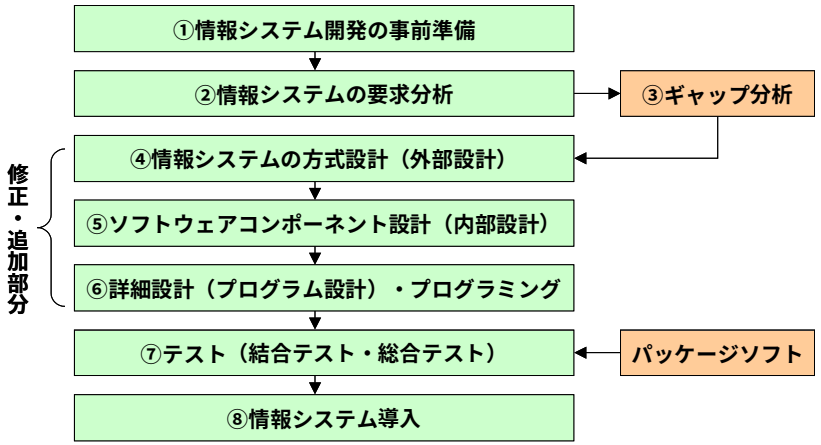


図５－２３　パッケージソフトをカスタマイズして情報システムを開発する場合の開発手順

４－３　プロジェクトマネジメント

（１）プロジェクトマネジメントの必要性

情報システムに係る技術が複雑化してきているため、当初において計画したスケジュール、品質等を開発プロセスにおいて達成することが容易ではなくなってきており、中には情報システムの開発に失敗する事例も出てきている。また、前述したように、情報システムの開発を委託先に「丸投

げ」する状況を改善し、情報システムの費用対効果を更に高めることが求められている。

したがって、情報システムの開発、特に大規模な情報システムの開発においては、開発プロジェクトのマネジメントを行うことが不可欠である。

情報システムの開発が自治体の職員の本来の業務でないことや、現状における知識や労力を考慮すると、PMBOK[2]等で標準化されている厳密な意味でのプロジェクトマネジメントを実践することは難しい。そこで、以下に示すような作業を中心として、情報システムの開発プロジェクトを少しでもマネジメントしようとすることが重要となる。

システム開発を委託するベンダーに対しては、PMBOK 等に準拠した厳格なプロジェクトマネジメントを要求し、プロジェクトマネジメント（ベンダー）を管理するプロジェクトマネジメントという位置付けとなる。

（2）進捗管理

情報システムの開発が開発実施計画のスケジュールに沿って進められているかを定期的にチェック（委託先のモニタリング）を行わなければならない。昨今では、作業の進捗状況を金銭換算して評価、管理する EVM（Earned Value Management）という手法が注目されているが、実施するための知識や労力、各作業の金銭換算の難しさ等を考慮すると、今後普及が進むにしても、その活用範囲は非常に大規模な情報システム等に限られるのではないかと予想される。したがって、計画プロセスで説明した WBS 等を用いて開発プロセスをできるだけ細分化し、作業毎の計画スケジュールと実際の進捗状況を符合させる程度の進捗管理が妥当な努力目標ではないかと考える。ただし、作業の進捗度合は%等で定量化し、その際に目安となる指標を作業毎に設定しておくことが望ましい。例えば、要求分析であれば利用者へのヒアリング実施数、プログラミングであれば完了したコンポーネントの本数等が目安となる指標として挙げられる。スケジュールが当初の計画よりも遅れている場合には、原因を究明し、早急に対策を行うことが望ましい。進捗遅延の主な原因と対策としては、表５－２４に示すような項目を挙げることができる。

[2] Project Management Body of Knowledge の略。米国の非営利団体 PMI（Project Management Institute）が策定したプロジェクトマネジメントの知識体系であり、業界標準となっている。「ぴんぼっく」と読む。

表５－２４　システム開発の遅延の主な原因とその対策

	項目	内　　容
遅延の原因	作業量の見積ミス	各工程の作業量を過小に見積もったため、労力に対して作業量が多くなる。
	設計ミス	外部設計、内部設計、プログラム設計等においてミスがあったため、プログラミングやテストの段階で差し戻しが発生する。
	仕様の追加・修正	何らかの理由によって仕様を追加・修正したため、新たな作業が発生する。
	要員の不足	病気やけが等によって開発要員が不足する。あるいは委託先が計画どおりに開発要員を配置していない。
遅延対策	ｽｹｼﾞｭｰﾙの見直し	稼働時期等を当初のスケジュールよりも後ろにずらして進捗との整合を図る。
	仕様の変更	開発内容を変更し、必要な労力や時間を少なくすることで進捗との整合を図る。
	要員の追加	（委託先に要請して）開発要員を追加し、作業量に対して不足する労力を補う。
	要員の変更	（委託先に要請して）開発要員を適当な人に変更することで、生産性を高める。

（3）品質管理

開発する情報システムが要求通りの品質を確保できるよう、開発プロセスを通して品質管理を行う必要がある。情報システムはハードウェア、ソフトウェア、ネットワーク等から構成されるが、ハードウェアの品質は商品の選択によって容易に管理できるものの、ソフトウェア、特にパッケージソフトではない新規に開発するソフトウェアの品質管理は容易ではない。また、ネットワークに関しては、基盤として全庁的に共通する場合がほとんどであるため、個々の情報システムの開発で検討する必要性は低い。したがって、品質管理においてはソフトウェアの開発に重点を置くことになる。

ソフトウェアの品質は、世界的に標準化が行われており、ISO/IEC9126 として六つの品質特性と 21 の副品質特性が設定されている。ソフトウェアの開発では、必要な品質項目において具体的な評価指標を設定し、テス

トや検査を行う時に、指標を基に評価を行い、品質の確保に努める。

また、ソフトウェアという最終的なアウトプットだけでなく、各作業工程においても品質管理を行う。各作業工程で作成される書類等の成果物についてレビューを行うことが不可欠であり、プログラムのバグの数等、各作業工程に関連する指標により評価を行うことで品質管理を行うことが望ましい。

なお、品質管理だけでなく、コミュニケーションや変更管理、導入後の運用等を考慮しても各作業工程の結果を、要求仕様書、業務運用手順、各設計書、テスト報告書等の形で文書化しておくことが不可欠である。

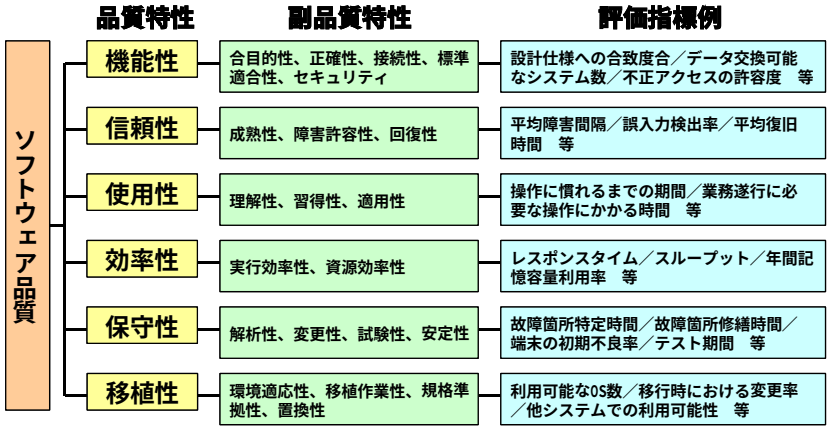


図５－２４　ISO/IEC9126 と品質評価指標の例

（４）コミュニケーション

庁内体制（担当チームやプロジェクトチーム）におけるコミュニケーションや、開発を外部に委託している場合は庁内体制と委託先とのコミュニケーションを円滑に図る必要がある。コミュニケーションには、開発プロセスに係る何らかの意志決定や重要な情報伝達を行う公式コミュニケーションと、それ以外の非公式コミュニケーションに分けることができる。

公式コミュニケーションの手段としては、会議・打合せや文書（電子化したものも含む）の受渡しがあり、手順や規則を事前に定めておくことが

必要である。会議・打合せは定期的に実施し、進捗管理や品質管理等に役立てる。会議や打合せの議事録、成果物の書類等は電子化し、庁内体制内または庁内体制と委託先の間で共有する。

非公式コミュニケーションも情報共有を図り、開発プロセスの質を高める上で重要であり、特に情報システムの政策的な背景や目的等、自治体独自の要素を公式及び非公式、双方のコミュニケーションで共有することが望まれる。また、非公式コミュニケーションの手段としてもグループウェアやメーリングリスト等の活用が有効であろう。

（５）体制管理

システム開発に携わる庁内の体制、および委託先の体制は、開発期間を通して適切なレベルに維持しなければならない。

庁内の体制では、情報システム担当者（チーム）やプロジェクトチーム等が計画プロセスから継続することが望ましいが、人事異動によってチームメンバーや情報システム担当者が替わる可能性もある。体制を維持する上で一度に多くのチームメンバーが入れ替わること、プロジェクトリーダーや情報システム担当者が替わることは望ましくなく、人事異動を考慮した体制整備、あるいは体制維持を考慮した人事異動が望まれる。

一方、委託先においては、システム開発に適当な体制が構築されているか、必要な能力を有した人材が体制の中に配置されているか、適切な役割分担が行われているか等を定期的にチェックする必要がある。

また、開発における生産性を高めるためには、体制内におけるモチベーションを高い位置で維持することや健康管理も重要であり、プロジェクトリーダー等にはこれらの役割も期待される。

（６）リスク管理と変更管理

計画プロセスで検討したリスクは開発プロセスで起こる可能性もあり、継続的なモニタリングが必要である。また、リスクが発生した場合に、開発プロセスを適切に変更することが必要である。例えば、情報システムで処理する業務に関わる法制度が開発期間中に改正された場合、新たな法制

度に対応する形で要求仕様書や各設計を変更しなければならない。また、このような外的なリスクだけでなく、設計ミス等の内的なリスクも存在し、その場合も適宜、上流工程に遡り内容を変更することが望ましい。

　変更に関しては、その必要性や影響度合、費用等を考慮して、庁内の体制において事前に設定した手順を基に決定することが望ましい。また、変更内容を文書化するとともに、開発を委託している場合にはその内容を委託先に通知しなければならない。もちろん細々とした変更のために承認等の手続を踏んでいたのでは作業が前に進まないので、ユニットやコンポーネントレベルの細かなプログラムの変更等に関しては、別途、簡易な手順を設けるとともに、変更履歴をまとめて記録しておく程度に留めることが必要である。