情報基礎シリーズ7

システム開発

電子開発学園出版局

情報基礎シリーズ7

システム開発

電子開発学園出版局

＊ 本書に掲載した会社名・製品名等は、一般にそれぞれ各社の商号・登録商標または商標です。

はじめに

コンピュータが開発されて以来、コンピュータは多くの分野で利用されてきました。現在では例えば、旅券を発行する、ご飯を炊く、ゲームをする、ニュースを知る、電話をするなど私たちの生活の中に根付いた分野で利用され、なくてはならない物として、また身近な物として存在しています。また現在のネットワーク社会はコンピュータなくして成り立ちません。

コンピュータはこれらの多くの分野で実現する様々な事柄を実現するために、“情報”を取扱い、制御し、処理しています。コンピュータで取り扱い、処理する“情報”とはいったいどのようなもので、それを取扱い、処理するためにどのような技術が使用されているのでしょうか。本書を含む「情報基礎シリーズ」では、このような知識、技術を“インフォメーションテクノロジ（IT)”と総称しております。

この“インフォメーションテクノロジ”を学ぶことは、様々な情報とそれを取り扱うコンピュータを学ぶことであり、それにより現代社会の基盤となるコンピュータについて理解を深めることとなり、そしてその社会についても知ることになります。

本書は、多くの分野から成立している“インフォメーションテクノロジ”のうち、システム開発を基礎から学ぶためのテキストであります。

本書によって、情報とはなにか、情報技術とはなにか、コンピュータはどのようなものかを学び、理解を深め、情報処理技術者試験の取得に役立てていただければ幸いです。

編著者

目　　次

情報基礎シリーズ7

システム開発

第１章　ソフトウェア開発管理

１．１　代表的なソフトウェア開発手法

システム開発の中核となるソフトウェアの開発工程では、作業の効率化や品質管理のために１９７０年代から工学的手法が盛り込まれ、さまざまな手法が編み出された。

本章では、代表的な手法について説明する。

１．１．１　ソフトウェア開発モデル

ソフトウェア開発では、その開発工程（開発プロセス）の形（モデル）を編み出し、これに沿って開発していくことで品質､効率の上昇を狙っている｡これを**ソフトウェア開発モデル**と呼ぶ。

開発プロセスとしては、この分け方もいくつかあるものの、一般的な分け方としては次のものがある。

・基本計画　　… 顧客の要望を聞き、ソフトウェアの基本計画を練る
・外部設計　　… 画面や帳票などの設計を行う
・内部設計　　… データ構造やモジュール分割などの設計を行う
・プログラム設計　… プログラミングのための設計を行う
・プログラミング　… プログラムを作成する
・テスト　　… プログラムのテストを行う
・運用・保守　　… 実際にシステムを稼働させ、その管理や保守を行う

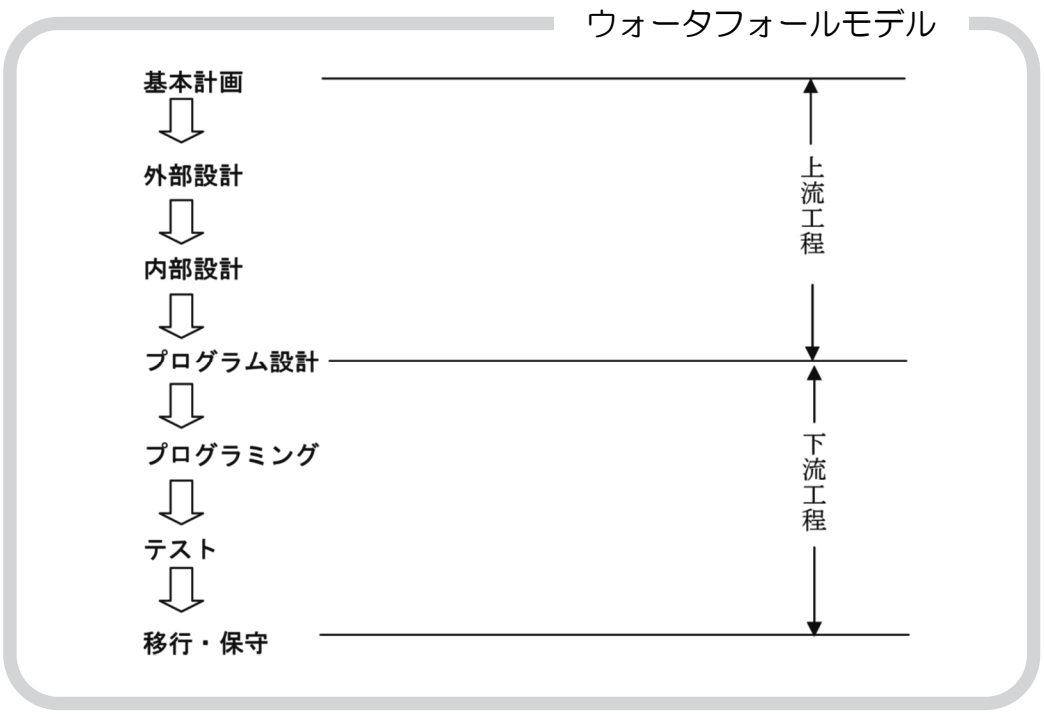
これらの開発プロセスをどのように行っていくかを決定するモデルが開発モデルである。

（１）ウォータフォールモデル

モデルの中で最も古い手法ではあるが、大・中規模のシステム開発手法として定着しているモデルが**ウォータフォールモデル**である。ウォータフォールモデルでは全体をいくつかの段階（フェーズ）に分け、段階的に開発していく方法であり、原則的に一度先のフェーズに進んだ後は前のフェーズへ後戻りしない。そこから、ウォータフォール（waterfall：滝）の名前がついている。

このウォータフォールモデルは､「計画→設計→製造→テスト→運用」の開発プロセスの流れに沿ったモデルであり、従来からの開発、特に大規模な基幹システムを開発する際にこの方法が取り入れられてきた。

この方法の長所は、システムの開発プロセスに沿って作業が進められるので作業手順がわかりやすいことや、工数の見積り、進捗状況管理に適していることである。



（２）ウォータフォールの問題点とプロトタイプ

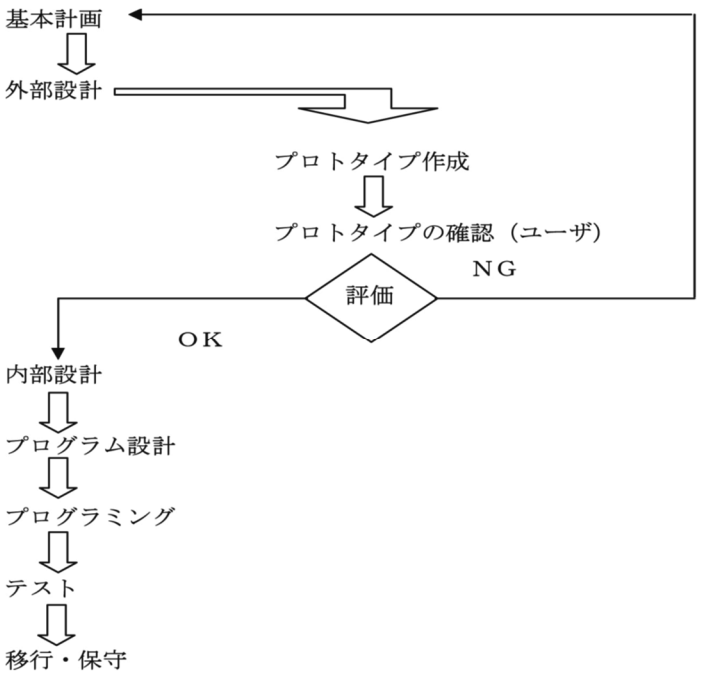
ウォータフォールモデルは優れた手法ではあるが、仕様変更が発生した場合の対応のしづらさがある。開発作業の後半にならないとユーザの要求が目に見える形で確認できないため、ユーザ側と開発者側での食い違いの発見が遅れるという問題点がある。そのために後戻り作業が発生した場合には開発の遅れを取り戻すのが非常に難しい、仕様のミスや設計の間違いなどの上流工程に問題があった場合に致命的な問題が発生するという難点もある。

そのため、まず最初に試作品（**プロトタイプ**）を作成し、それに対してユーザと確認を行い、ユーザと開発者間での食い違いをなくし、また仕様のミスなどを発見しやすくするという手法がとられる。

(a)　プロトタイピングモデル

プロトタイプを作成する手法が**プロトタイピングモデル**である。開発プロセスの初期に、目に見える形でのプロトタイプを作成し、ユーザからの確認と評価を得る方式である。その後の手法はウォータフォールモデルで行うことが一般的である。

プロトタイピングモデル



プロトタイプでのユーザ側の評価を受け、さらに基本計画を修正しもう一度プロトタイプを作成するか、このまま開発を続けていくかを決定する。

ユーザからの確認を行うため、ウォータフォールモデルで起きる問題が発生しにくいという利点がある一方で､プロトタイプを見てしまったがためにユーザ側の意見が分散しやすい、プロトタイプを作成する分の時間と費用が特にプロトタイプを数度作り直した場合には大きく必要となってしまうなどの問題点もある。

（ア）使い捨て型プロトタイピング

作成したプロトタイプは実際のシステム開発では使用せず破棄し、ユーザとの確認・評価用だけに使用するプロトタイピングである。この場合、より手早くプロトタイプを作成するためにツールなどを活用する。

（イ）骨格型プロトタイピング

作成したプロトタイプに対し、それを中核として肉付けしていくことで、最終的なシステムを開発していくプロトタイピングである。この場合、作成するプロトタイプはシステムの中核となるので試作品といえど堅固な作りにする必要がある。

(b)　ＲＡＤ

ＲＡＤ（Rapid Application Development）は高速（Rapid）にシステム開発を行う手法で、一般的にはプロトタイピングを利用して行う手法である。ＲＡＤは特に少人数の開発者とユーザでチームを組み、ユーザの希望を随時取り入れながら、何度もプロトタイプを作成していくことで高速にシステムを開発していく。そのためにＣＡＳＥツールやＧＵＩツールなどを使用する。

プロトタイピングと、後述するスパイラルモデルを組み合わせたような手法で、画面などのインタフェースの開発では、ウォータフォールモデルなどよりも高速で開発ができることからこの名前がついている。

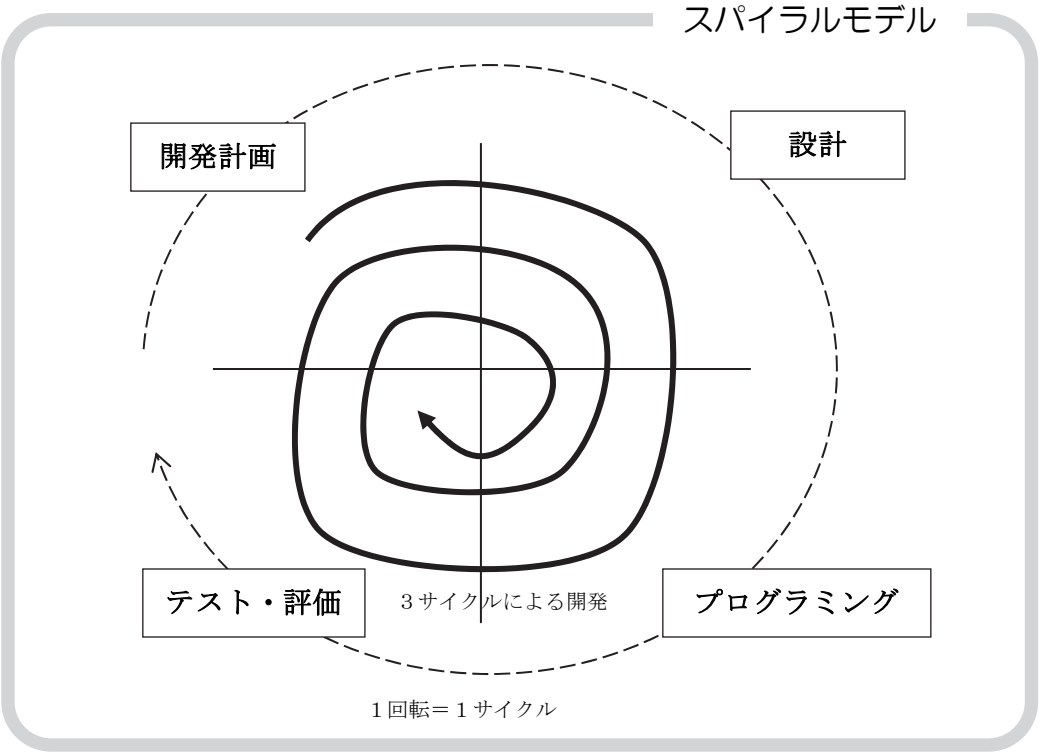
（３）繰り返し型開発モデル

繰り返し型開発モデルもウォータフォールモデルの問題点を解消すべく考えられたモデルで、名前の通り“繰り返し”を行っていくことが重要となっている。ソフトウェアを徐々に開発していき、ユーザの評価、再設計、開発というプロセスを繰り返すことでソフトウェアの完成度を高めていく手法である。前述のＲＡＤもプロトタイピングと繰り返し型開発モデルを組み合わせた手法である。

繰り返し型開発モデルは、ユーザからの仕様変更などにも対応でき、特に技術の進展スピードが速く、システムを開発中に新たな要望が出てきやすい近年の状況などにも向いていると考えられている。

(a)　スパイラルモデル

スパイラルとは「渦巻き」のことで、同モデルはウォータフォールモデルとプロトタイプモデルの両方の手法を取り入れている。ウォータフォール的な「計画→設計→製造→テスト」という開発工程を“１回転（サイクル）”分とし、このサイクルを何回か繰り返すことで開発を完了させる形になる。サイクルを行う回数は事前に決定されて運用される。



ウォータフォールモデルにより作成されたプロトタイプに対し、テスト、ユーザの評価を得てさらにそれに対し修正、新機能の追加をしていく形になる。１サイクルごとにテスト・評価が行われて現在の開発におけるリスクが判明し、そのリスクが最小になるように開発を行う。そのため、大規模なシステムや未経験分野のシステムの開発に向いているとされる。

スパイラルモデルでは、システム全体の設計から、機能を独立性の高い複数のシステム（サブシステム）に分割し、このサブシステムを順次作成していく形になる。

(b)　段階型モデル

段階的モデル（Incremental Model：インクリメンタルモデル）は、サブシステムを段階的に開発していく手法である。それぞれのサブシステムはウォータフォールモデルで開発し、それを同時並行して開発できる場合は、並行して開発していく。開発するサブシステムのうち優先度の高いサブシステムから順次優先して作成していき、完成したサブシステムを追加していく形になる。

機能を追加していく形になるので、最初の段階からテストが可能であり、またスパイラルモデルと異なり、サブシステムを同時並行で開発できる。

(c)　進展的モデル

進展的モデル（Evolutionary Model）は、初期は最小限の機能を持つシステムを構築し、開発工程を繰り返すことで、機能を追加しシステムを成長させていく手法である。成長モデルとも呼ばれる。

（４）アジャイル開発

アジャイル（Agile）とは「機敏な、俊敏な」という意味であり、**アジャイル開発**は素早く、適応的にシステムを開発する手法である。アジャイル開発はそれ自体が開発の手法ではなく、ＸＰ（Extream Programing）、スクラム、ユーザ機能駆動開発（ＦＤＤ：Feature-Driven Development）などの高速で適用的なシステム開発手法（アジャイルプロセス）に共通するシステム開発の考え方、原則、取り組み方のことである。

アジャイルプロセスの多くで取り入れられている特徴としては、システムを多数の小さな機能に分割し、短い期間（イテレーションと呼ぶ）で機能を開発する。このイテレーションを繰り返し行い、システムを成長させていくという特徴がある。また、開発は少人数のチームで行い、コミュニケーションを重視し、ユーザもチームの一員であるという考え方がある。

他にも、システムは変化するものであるという考え方を受入れることで実際に適用した開発を行い、動作するソフトウェアの開発を重視するため、他の開発手法で作成される開発文書が必ずしも作成されないなどの特徴もある。

（５）ソフトウェアプロダクトライン開発

システムを開発する場合に、同種、同系統のシステムを効率的に開発できるように、共用や再利用が可能なようにシステムを開発する手法を**ソフトウェアプロダクトライン開発**と呼ぶ。ソフトウェア、システムを量産できるようになるため、組み込みソフトウェアのように類似のソフトウェアを作成する場合などで利用される。

共有・再利用されるソフトウェアはコアセットと呼ばれ、このコアセットが利用できるソフトウェアの範囲をドメインと呼ぶ。コアセットを作成するドメインエンジニアリング、コアセットを使って開発するアプリケーションエンジニアリング、ドメインを管理する、という３つの段階でソフトウェアプロダクトライン開発は行われる。

１．１．２　ソフトウェア開発の評価と手法

システム開発・ソフトウェア開発において、その規格や評価手法が存在する。またソフトウェア開発の手法としてソフトウェア再利用とリバースエンジニアリングについて説明する。

（１）ソフトウェアライフサイクルプロセス

ソフトウェアライフサイクルプロセス（ＳＬＣＰ：Software Life Cycle Process）は、ソフトウェア、システムに関連する人々（開発者、ユーザ区別なく）が“同じ言葉”で“同じ物差し”を使って作業範囲や内容を明確にし、相互理解や齟齬（そご）の解消を行い、開発とその取引を正常化することを目的として規格化されたものである。

例えば“保守”という言葉１つとったとしても、開発者側が考えている“保守”と、ユーザ側が求めている“保守”では範囲が異なり、それによりトラブルが発生することがありえる。このような事態を防ぐためにＳＬＣＰが標準化された。

(a)　共通フレーム２０１３

ソフトウェアライフサイクルプロセスの国際規格として、ＩＳＯ／ＩＥＣ１２２０７がある。これをＪＩＳが規格化したものが、**ＪＩＳ　Ｘ０１６０**である。

それ以前にも情報処理振興事業協会（ＩＰＡ）を中心に、産業界、大学、公官庁によるソフトウェアライフサイクルプロセスについての検討が行われており、１９９４年には“ソフトウェアを中心としたシステムの取引に関する共通フレーム”（共通フレーム９４）と呼ばれる規格ができた。

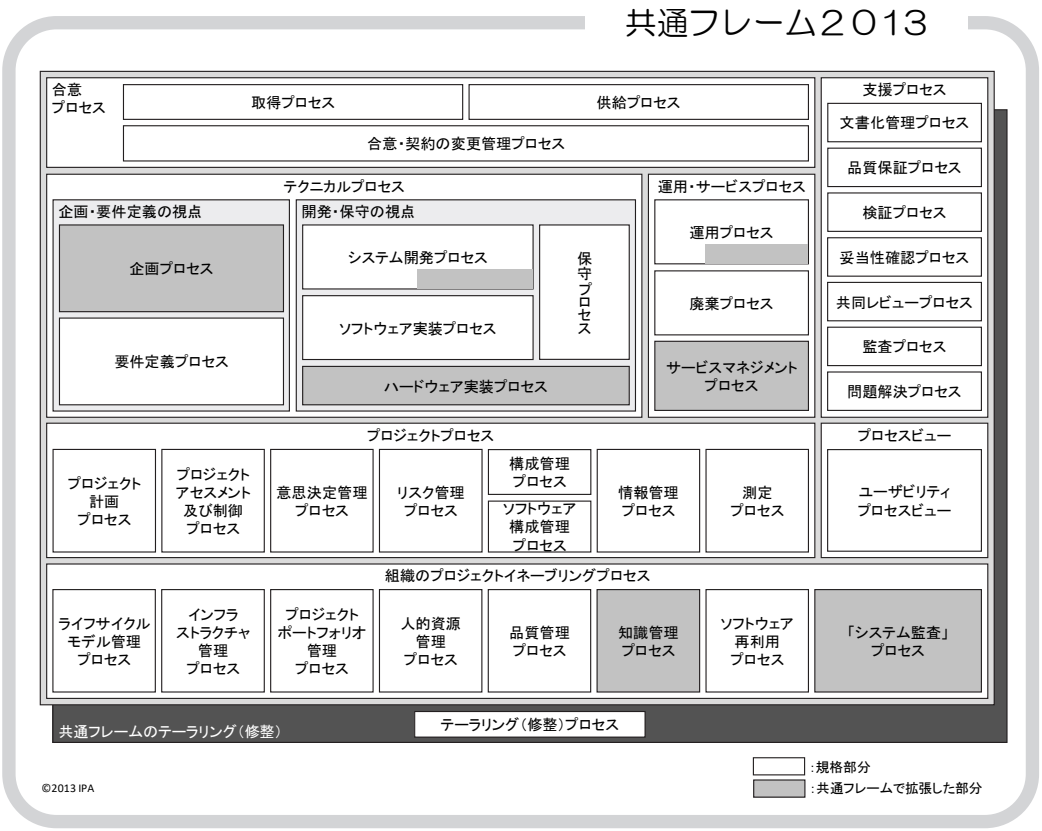
この共通フレーム９４を、国際規格とＪＩＳ規格に合わせて改訂したものが**共通フレーム２００７**（**ＳＬＣＰ－ＪＣＦ２００７**：Software Life Cycle Process - Japan Common Frame2007）である。さらに、共通フレーム２００７を改訂したものが**共通フレーム２０１３**（**ＳＬＣＰ－ＪＣＦ２０１３**：Software Life Cycle Process - Japan Common Frame2013）である。

(b)　共通フレーム２０１３の内容

（ア）プロセス

共通フレーム２０１３ではシステム開発で生じる作業を、役割の概念でまとめた“プロセス”、プロセスを個々に細分化した作業である“アクティビティ”、アクティビティで実行すべき作業である“タスク”、タスクの作業についての事例を“リスト”というように階層化している。

共通フレーム２０１３で規定されているプロセスは次の図のようなものがある。



（イ）システムとソフトウェア

共通フレーム２０１３でのシステムは、“事業”、“業務”、“システム”、“ソフトウェア”と階層構造で規定されている。

・事業　　… 企業の経営のことで、システム化の方向性の決定や計画、評価を行う
・業務　　… 企業が行う仕事・作業のことで、販売や物流などがある
・システム … コンピュータが行う業務で、ソフトウェア、ハードウェア、ネットワークなどからなる
・ソフトウェア… 情報システムを実装する工程である

共通フレーム２０１３では、システムとソフトウェアは明確に区分されており、それぞれシステム要求事項とソフトウェア要求事項、システム方式設計とソフトウェア方式設計のように要求事項や設計が分かれている。

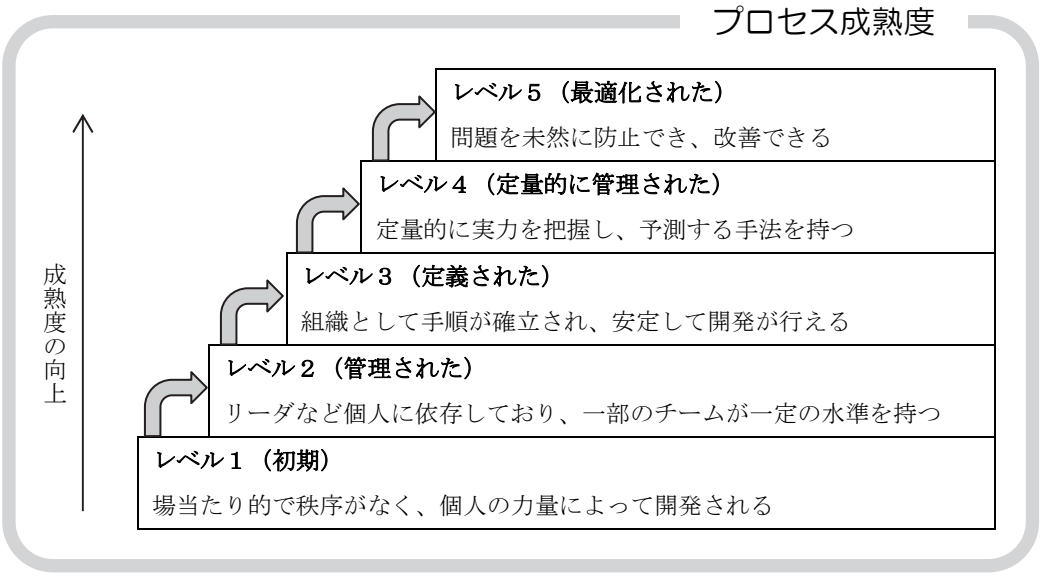
（２）プロセス成熟度

システム開発、ソフトウェア開発を行うためには、開発者の能力・経験などだけでなく、開発企業の組織としての能力も必要となる。そこでユーザがソフトウェア開発を開発企業を選択する際の基準として、企業・組織の持つ組織的能力を評価する必要がある。この組織的能力を**プロセス成熟度**と呼び、その評価基準としてＣＭＭＩが存在する。

(a)　ＣＭＭＩ

ＣＭＭＩ（Capability Maturity Model Integration：能力成熟度モデル統合）はアメリカ合衆国カーネギーメロン大学のソフトウェア工学研究所が、ソフトウェア開発企業が持つプロセスに対する能力を客観的な評価できる基準として開発されたものである。

ＣＭＭＩはシステム開発、ソフトウェア開発、リスク管理、プロジェクト管理、調達能力などの企業が行う開発プロセスを１～５までの段階で評価する。



（３）ソフトウェア再利用

ソフトウェアの再利用とは、既存のソフトウェアを活用して新たなソフトウェアを作成することで、この場合のソフトウェアとはプログラム（コード）だけでなく、設計書やテストケースなども含まれる場合もある。工業製品で既存の一定水準を持つ規格のボルトやナットなどの部品を利用して新製品を製造するように、ソフトウェアでも部品を利用して新たなソフトウェアを開発できるようにすれば、作業量の軽減、品質の向上、保守や改訂が容易になるなどの利点を見込むことができる。

(a)　再利用の段階

ソフトウェアの再利用としては、次のような段階がある。

（ア）プログラムのコピー

新たに作成するソフトウェアでも使用する、または似ている過去のプログラムの一部をコピーして使用する。ただしこれは、コピーにより同じようなコードがプログラム内に複数存在することになる場合がある、プログラムで使用する変数の整合性が取れないなど、プログラムの修正や保守の段階で非常に問題が多発する。

プログラムのコピーは過去のコードをもう一度利用しているので再利用ともいえないことはないが、問題が多く再利用の利点がみられない。

（イ）個人、チームでの再利用

個人やチーム単位で過去に作成したソフトウェアを再利用する。ただし、これらの再利用は、個人で作成したものであったり、たまたま使えるから使用するという形での再利用が多く、計画性に欠けるため、どのソフトウェアが再利用できるかわからないなどの欠点があり、再利用の利点である作業の軽減などがあるとは言えない。

（ウ）計画的な部品化と再利用

企業、または組織が再利用を前提として、ソフトウェアを“部品”と考えて設計、作成に取り組むことで、計画的な再利用ができることになる。これを**部品化**と呼ぶ。

部品化のためには、次のような項目を検討する必要があるため、再利用を計画的に進めていくことが必要である。

【部品の標準化】

部品で使用する変数の命名基準や、入出力データ（インタフェース）の統一などを行う。

【部品の体系化】

どのような部品が存在し、それをどのように使用するか、使う場合の注意点は何かなどをまとめた部品ライブラリを整備する。また、様々な用途の部品を用意し、再利用率を高める。

(b)　再利用の種類

再利用に使われる部品としては、機能ごとに独立させ再利用できるようにした**コンポーネント**、**モジュール**、日付計算や情報のフォーマット、入出力装置へのアクセスなどの使用頻度の高い汎用的なプログラムをまとめた**ライブラリ**、オブジェクト指向設計で使用されるクラス、汎用的でソフトウェアの基盤となるライブラリやクラスをまとめた**フレームワーク**などがある。

（４）リバースエンジニアリング

既存のシステムやソフトウェアなどを分析することで、そのシステムの設計方針やソフトウェアの動作状態、プログラムコードを解釈することを、**リバースエンジニアリング**と呼ぶ。

自社が開発した既存の情報システムが改修・機能追加などでシステムが膨らみ保守が容易ではなくなっていた場合や、開発が古いため当時の技術者がいないことや設計書が紛失している場合などに、リバースエンジニアリングを行い、設計や仕様を導き出すことに使用される。これにより、保守や拡張を容易にする。

リバースエンジニアリングではコンパイル済みのアプリケーションから、デバッガを用いて変数や動作を確認する、トレーサやメモリダンプによりメモリを確認するなどにより、アプリケーションプログラムのコードを解釈する。他にもプログラム同士の呼び出し関係を明示させるコールグラフを作成するなどを用いて､プログラム間の構造を明らかにすることなどを行う。

また、コードを書き換えて機能を追加することもリバースエンジニアリングになる。

(a)　リバースエンジニアリングの問題点

他社のソフトウェアやシステム製品に対してリバースエンジニアリングを行い、その分析結果に基づいて新たなソフトウェア、システムを開発し販売する場合、その過程で複製や翻訳など元のソフトウェアに対する知的財産権の侵害が発生していると考える場合があるので、リバースエンジニアリングを実行する場合には注意が必要となる。

また、現在の多くのソフトウェア製品は、その利用許諾契約にリバースエンジニアリングの禁止をうたっている。

１．１．３　その他の開発手法

（１）構造化手法

構造化手法は構造化設計とも呼ばれる手法であり、システム、特に大規模なシステムや複雑な処理を持つシステムに対して、システム全体を小さな要素に分割し、それを階層的に組み上げて作り上げる設計方式である。小さな要素として設計していくため、品質を維持しやすくまた保守を容易にすることができる。

(a)　アプローチ

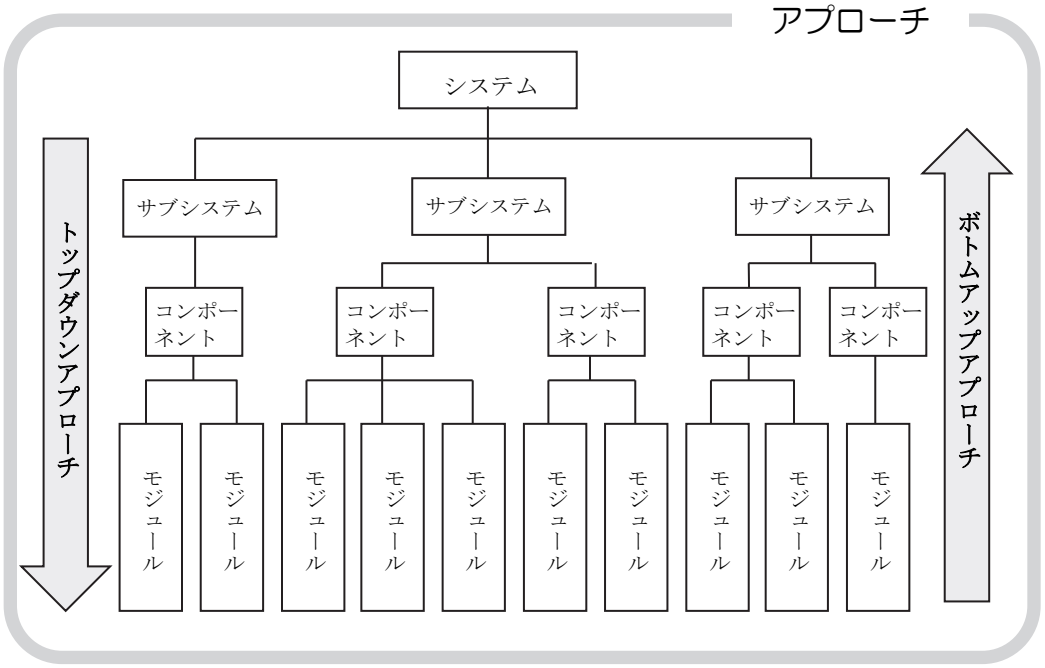
構造化手法を行うためにシステムを要素に分割していくが、そのためのアプローチとして**トップダウンアプローチ**と**ボトムアップアプローチ**がある。

（ア）トップダウンアプローチ

トップダウンアプローチは、**段階的詳細化**とも呼ばれる手法である。まず最初にシステムそのものに要求された問題があり、それに必要な要素を洗い出す、洗いだした要素に対してさらに必要な要素を洗い出す、というように行う手法である。最終的に、これ以上要素に分解できない段階で終了する。システム、サブシステム、コンポーネント、モジュール、ソフトウェアコードのように順に小さな要素に分解していく方法である。システム開発ではトップダウンアプローチが使われることが多い。

（イ）ボトムアップアプローチ

ボトムアップアプローチはトップダウンアプローチの逆に、もっとも詳細な要素から組み上げていく手法である。システムそのものに要求された問題に対し、問題を解決するために必要な機能を見つけ、さらにそれに共通する部分を見つけてまとめ、さらにまとめたもの同士でさらにまとめるを繰り返していく手法である。



(b)　機能分割と構造化

構造化手法ではトップダウンアプローチを用いて、情報システムを機能ごとの要素に分割を行う。そのための手法としては、**ＤＦＤ**や、**Ｅ－Ｒ図**、**状態遷移図（ステートマシン図）**などを用いてシステムを分割し、階層構造にする。ＤＦＤとＥ－Ｒ図については、「１３．１．２」で説明する。状態遷移図は、有限オートマトンでの状態の遷移を表す図である。

（２）形式的手法

形式的手法は数学をベースとした、システム開発の手法である。仕様の定義、プログラミング、検査などで数学的な論理体系を用いて厳密に定義することで、品質の向上を目指すための手法である。

形式的手法では、曖昧な記述になる自然言語を用いて記述していたシステム開発の仕様の定義を、“仕様記述言語”を用いて論理的に仕様を記述する“形式仕様記述”を行う。また検査の際にでも、“形式検証”と呼ばれる方法を用いて検査を行う。

この時使用する手法として代表的なものにＶＤＭ（Vienna Development Method）があり、形式記述言語としては、ＶＤＭ－ＳＬ（Specification Language）やＶＤＭ＋＋がある。このＶＤＭによる開発を支援するためにＶＤＭＴｏｏｌｓが存在する。

（３）マッシュアップ

インターネットを利用して、Ｗｅｂサイトによるネットショッピング、地図検索、天気予報、チケット予約などの、ユーザがブラウザを利用することで受けることができるサービスのことを**Ｗｅｂサービス**と呼ぶ。Ｗｅｂサービスのうち、他のＷｅｂサービスに対してサービスが提供する情報をＸＭＬ形式で提供しているものがあり、これをＷｅｂサービスＡＰＩと呼ぶ。

ＷｅｂサービスやＷｅｂサイトを作成するシステム開発において、これらＷｅｂサービスＡＰＩを組み合わせて、１つの新しいＷｅｂサービスを構築する手法をマッシュアップと呼ぶ。例えば、旅行サービスを行うＷｅｂサービスで、ホテルの予約、鉄道・飛行機の予約、地図検索、天気予報など本来は別々のＷｅｂサービスであるものを、まとめて１つのＷｅｂサービスとして提供するなどがマッシュアップである。

(a)　ＷｅｂサービスＡＰＩの利用

ＷｅｂサービスＡＰＩ提供以前でも、マッシュアップと同様のことを実現することは可能であった。例えば先程の旅行サービスでいえば、ホテル、運送会社など別々の企業にデータ提供をしてもらい、サービスとして提供していた。だが、それぞれ別々の企業が持つ別々のサービスであるため、それを統合して、１つの新しいシステムにするには非常に多くの作業が必要であり、またその品質を維持する努力も必要となる。

だが、ＷｅｂサービスＡＰＩを多くの企業が提供し、またそのＡＰＩのデータ通信用のプロトコルとして**ＳＯＡＰ**や**ＲＥＳＴ**が用いられるようになったため、様々なＷｅｂサービスＡＰＩを容易に利用できるようになった。

マッシュアップはＷｅｂサービスＡＰＩを利用することで、システム開発のコストを大幅に下げることができ、また機能拡張なども容易に行えるという利点があるものの、Ｗｅｂサービスを提供している企業側がサービスの変更や停止に影響されてしまうという欠点も持ち合わせている。

(b)　マッシュアップの技術

マッシュアップで使われているＷｅｂサービスＡＰＩの代表的な技術としては次のようなものがある。

・ＸＭＬ　　… ＳＯＡＰ、ＷＤＳＬ、ＲＥＳＴでデータを表現するために使用される
・ＳＯＡＰ　　… ソフトウェア間でデータを交換するためのプロトコル
・ＷＤＳＬ（Web Services Description Language）…Ｗｅｂサービスの定義を行うための記述言語
・ＲＥＳＴ（REpresentational State Transfer）… ＸＭＬとＨＴＴＰによるＷｅｂサービスのシステム
・Ａｊａｘ（Asynchronous Javascript + XML）… ＸＭＬとＪａｖａＳｃｒｉｐｔを組み合わせ、ブラウザで動的なユーザインタフェースを画面を切り替えずに行うことができる技術