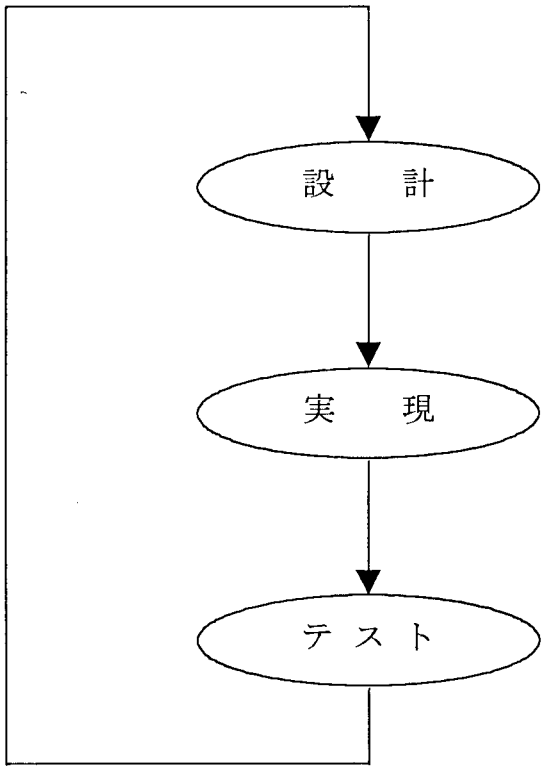
VI テスト管理と検査（ソフトウェアとテスト）

1 ソフトウェアのテスト

ソフトウェアのライフサイクルは、開発と保守の２段階に大別できる。さらにそれらは、図VI－１のように設計・実現・テストという工程に分割される。



図VI－１　設計・実現・テスト工程

設計工程では、要求分析により、どのようなソフトウェアを開発し、どのような変更をするかを決定する。

実現工程では、設計に基づき、どのように実現すべきかを検討し、ソフトウェアの開発・変更をする。

テスト工程では、開発・変更の結果、そのソフトウェアが、設計どおりのものか確認する。設計・実現・テストを１つのサイクルとして、ものを開発・保守していくのは、ソフトウェアだけではなく、ハードウェアも同様であるが、基本的な違いは、そのライフサイクルの長さにある。

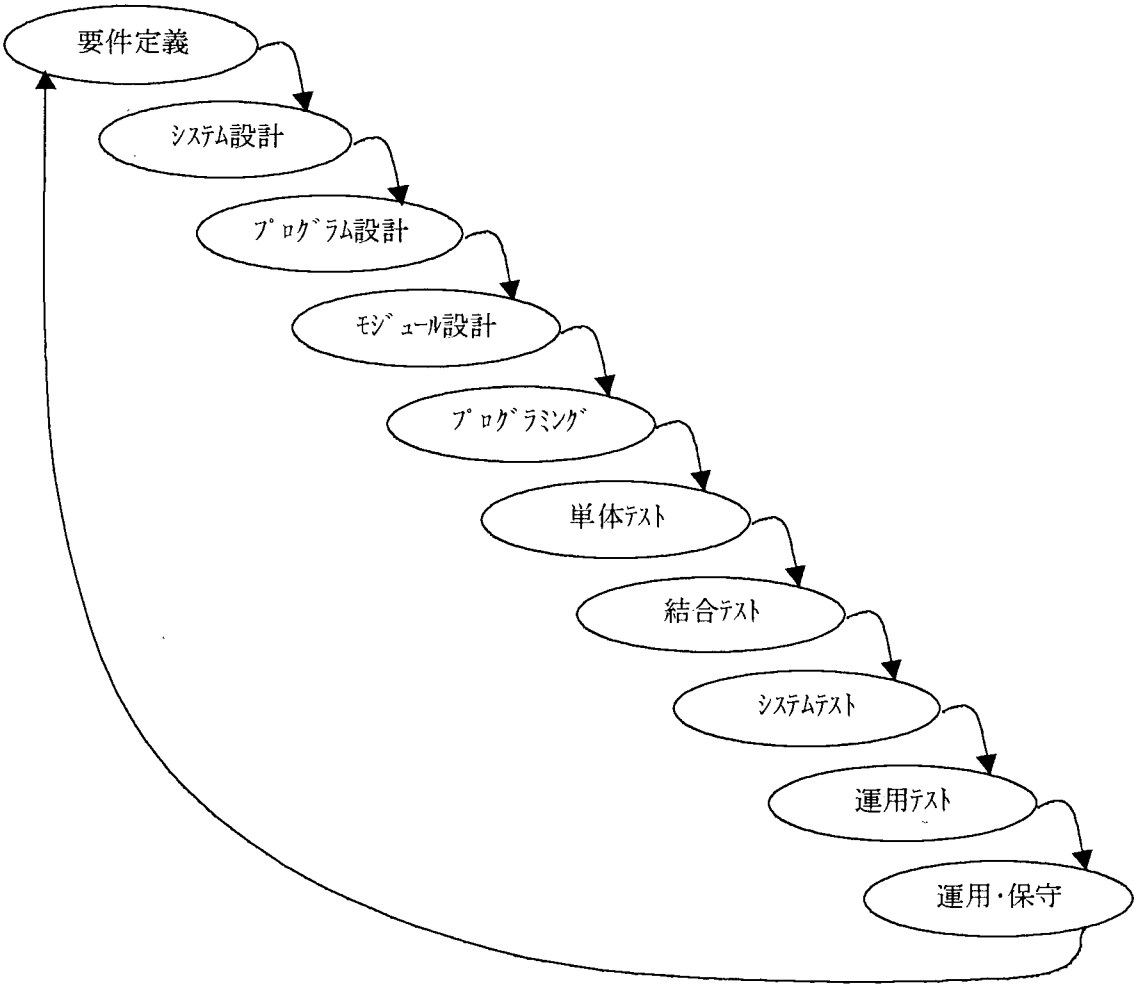
一般的にソフトウェアの寿命は、ハードウェアよりも長い。例えば、ハードウェアをリプレース等により交換してもソフトウェアは継承する場合が多い。

したがってライフサイクルで考えた場合、開発段階より保守段階の方が長くなる。

ソフトウェアの保守には、下記の２通りのケースがある。
・設計段階のエラーにより発生する不具合を解決する
・ 環境の変化（法制度の変更、業態の変化等）へ対応するための変更
ソフトウェアのテストは、特に設計段階のエラーにより発生する不具合を、開発段階で発見し、問題の発生を未然に防ぐために必要となる。

2 ソフトウェアのライフサイクル

従来の一般的なソフトウェアのライフサイクルにウォーターフォールモデルがある。
これは図Ⅵ－2に示すように、開発工程を何段階かに分割したもので、前の工程の出力が次の工程の入力になる。



図Ⅵ－2　ウォーターフォールモデル

単体テストは、プログラムを構成するモジュールの機能が設計どおりのものであるか確認

し、結合テストでは、システムを構成するプログラムがプログラム設計どおりであるか確認するのが目的である。

またシステムテストは、プログラミングにより実現されたシステムがシステム設計された外部仕様どおりであるか確認し、運用テストでは、システム全体を利用者の立場から本稼動の環境でテストを行う。運用テスト後、システム運用・保守の段階に入るが、変更要求があった場合、変更についての要求定義から同様の手順を踏むことになる。

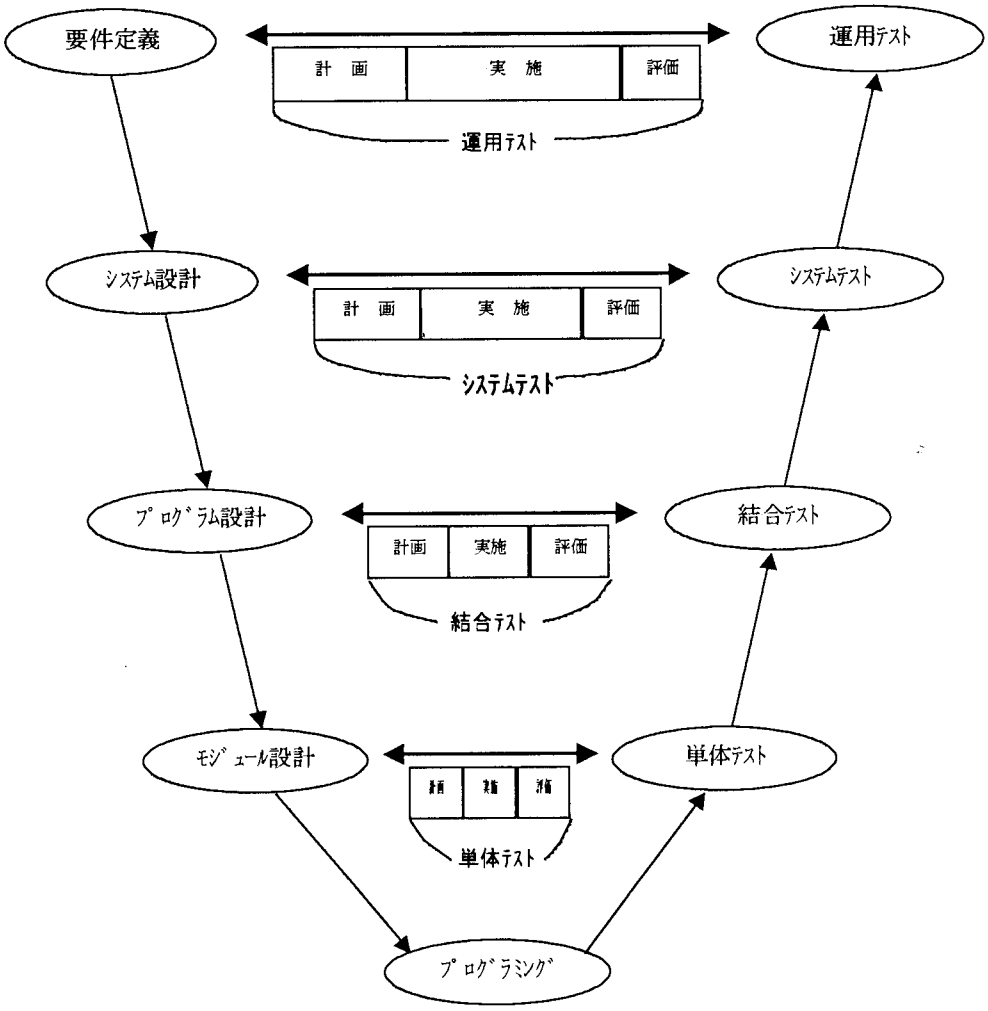
ところで大規模システムでは、要件定義やシステム設計に不具合があった場合、システム設計段階で発見された欠陥はそれが大きければ大きいほど致命的なものになる。
ウォーターフォールモデルの開発では、システムテストの段階で品質を検証するのは遅すぎるということになる。この問題の解決策として、ウォークスルーやインスペクションの徹底やＤＯＡ（データ中心アプローチ）による要件定義がある。

3 テストの方法

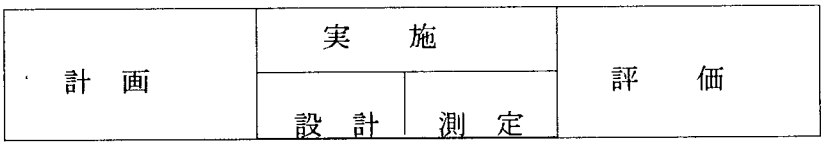
単体テストでは、１つのモジュールの機能が正確な動作をすることを確認し、結合テストでは、複数のモジュールの集まりであるプログラムを個別にテストする。システムテストでは、複数のプログラムがシステム要求を満足していることを確認する。
それらの確認手段は、テストデータを用意し実験することである。

テスト計画の技術、そしてテスト結果の分析評価技術は、推論又は証明の方法である。
単体テストで機能確認する場合、もしある機能が正確に動作しないとすれば、どの部分が悪いか、どんなデータを入力すればその部分が動作するかの推論が必要である。
結合テストでプログラム設計の要求を満足しているか確認する場合、もしモジュール間のインターフェイスがとれていなければ、どんなデータを入力すれば、その部分が検証できるかの推論が必要である。またシステムテストでシステム要求を満足しているかどうかを評価する場合も、もし満足されていなければ、どんなデータを入力したときに、どの程度システム要求を下まわるかの推論が必要になる。

ウォーターフォールモデルでは、図Ⅵ－3のように設計とテストを対応づけたＶ字のカーブをえがくことになる。このテストは、計画・実施・評価という工程があり、単体テスト、結合テスト、システムテスト、運用テストの４段階のテストがあるので、４つのテストサイクルが動くことになる。



図VI－3 ウォーターフォールモデル



図VI－4 テストの工程

テスト設計工程（図VI－4）では、テストケース（どのようなテストを実施するか）を作成し、それに基づき測定するのに必要な、ハードウェア・ソフトウェア環境などを整備する。この工程が完了したのち、次の工程の測定を実施する。

テストの設計、テスト結果の評価は、経験と知識を必要とする作業である。

4 テストケースの設計技法

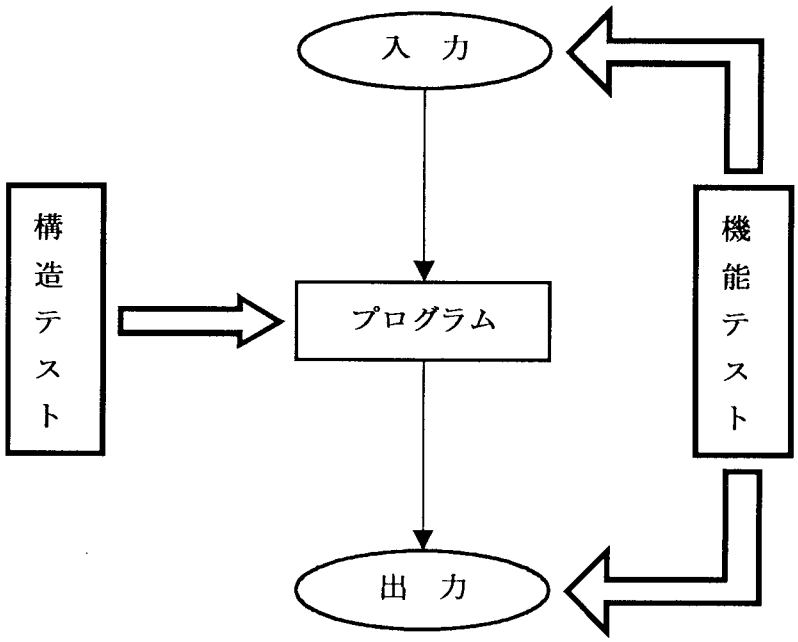
テストケースは、システムの効率的なテストを行なうためのテスト条件である。
プログラム中のすべての条件を網羅するテストデータを用意したり、実データ以外の例外的

なデータも用意し、システムの信頼性を確保する必要がある。そのために、テストケースの設計はテスト計画段階における重要な作業である。

テストケースの設計技法は、図VI－ 5 のように大きく機能テストと構造テストに分けられる。

機能テストは、プログラムをブラックボックスとみなして、プログラムの内部構造（条件、命令の順序等）にまったく関係なくある入力を与えた時に、仕様どおりの出力が得られるかに着目してテストケースを作成する方法である。入力される可能性のあるさまざまな組み合わせのテストケースを推測し、プログラムの機能をテストするのが目的である。
ブラックボックステストと呼ばれる。

それに対して、構造テストは、プログラムの内部構造に基づいて、テストケースを作成する方法である。すべての命令を実行できるようにテストデータを用意し、プログラムを網羅的にテストするのが目的である。単体テストの段階で行なうことが多い。
ホワイトボックステストと呼ばれる。



図VI－5 プログラムとテストケース設計技法

テストケースの設計は、通常この２つの方法を併用することが多い。
またテストケース数を比較した場合、ホワイトボックステストは、プログラムの内部構造を網羅するようにテストケースを用意するため、ブラックボックステストより多くなる。

(1)　ブラックボックステスト

プログラムの仕様書に基づいて設計するブラックボックステストの代表的な方法を説明する。

① 原因･結果グラフ法

プログラムの仕様は、ある条件（原因）に対して、ある機能（結果）を対応づけたものである。この原因と結果の関連を調べグラフを作成し、表形式の決定表（Decision Table）からテストケースを作成する方法である。

② 境界値テスト法

プログラムの処理が正しく実行されるかどうか、仕様の入力と出力の条件を決定する境界値を中心にテストケースを作成する方法である。

③ 同値分割法

プログラムの入力の使用条件から、たとえば正常に処理されるデータと異常に処理されるデータに分類するように、入力データの範囲をいくつかに分類し、各分類からその分類の少なくとも１つの代表値をテストデータとする方法である。

(2) ホワイトボックステスト

プログラムの内部構造・制御の流れに基づいて設計するホワイトボックステストについて説明する。

① 命令網羅法

構造テストのうちもっとも基本的な方法である。

この方法は、プログラム内の命令が少なくとも１回は実行されるようなテストケースを作成する方法である。

② パス網羅法

この方法は、プログラムのすべての内部構造をテストする方法である。

プログラム内の判定条件で、真偽の組み合わせを満たすように、すべての分岐命令が少なくとも１回は実行されるようなテストケースを作成する方法である。

5 クライアント／サーバ上におけるテスト

企業において基幹業務にコンピュータシステムを導入した場合、システムの品質・信頼性の向上は重要な課題である。これは、従来のオフコン系システムはもちろん、クライアント／サーバシステム（図Ⅵ－6）においても同様である。クライアント／サーバシステムは、CUI(Character User Interface)からGUI(GraphicalUser Interface)の技術によるアプリケーションプログラムの開発が一般的となり、またオープンシステムの思想によりシステムを構築する場合、複数のメーカーの製品を導入することになる。

このような環境下での、テストの計画・実施・評価の作業量は膨大なものになる。

(1) GUIアプリケーションプログラムのテスト

クライアント／サーバ上でのアプリケーションは、オブジェクト指向の開発ツールを利用するのが一般的である。作成されたアプリケーションを、あたかも人間が操作しているかのようにテストを行う自動化テストツールがある。テストツールは、ＧＵＩやオブジェクトの機能を満たしているかどうか、自動的にテストを実行することができる。

また、クライアント／サーバ上でのシステム開発手法にプロトタイプを利用する方法がある。ユーザの仕様を満たしているか確認するたびにアプリケーションを変更し、テストを繰り返すことになる。このような場合、テストツールにテストパターンを登録しておくことにより、テストパターンを利用し、変更された仕様の確認ができることになる。このようにテストツールを利用することにより、テストに要する時間を大幅に削減することができる。

(2) クライアント／サーバ上の性能テスト

クライアント／サーバ上において、GUI アプリケーションの予想したレスポンスが得られないケースがある。この場合、自動生成された SQL によるネットワークトラフィック量の増大によることが多い。トラフィック量を最小にするため、SQL をカプセル化し たり、RPC (Remote Prodedure Call) を使うなどの最適化をすることになる。

また、WAN 環境でのシステムの運用は、LAN 環境に比べ伝送速度が遅くなるため、大量の検索結果が WAN 上を流れる結果、レスポンスが悪化する場合がある。この場合も、上記のように SQL を見直し、サーバとデータベースの構造にあった検索を行なうことで改善されることになる。

(3) クライアント／サーバー環境での確認テスト

クライアント／サーバー環境のシステムでは、システムの開発環境と実動環境との違いにより、さまざまなテストが必要となる。

①マルチ・プラットフォーム上でのテスト

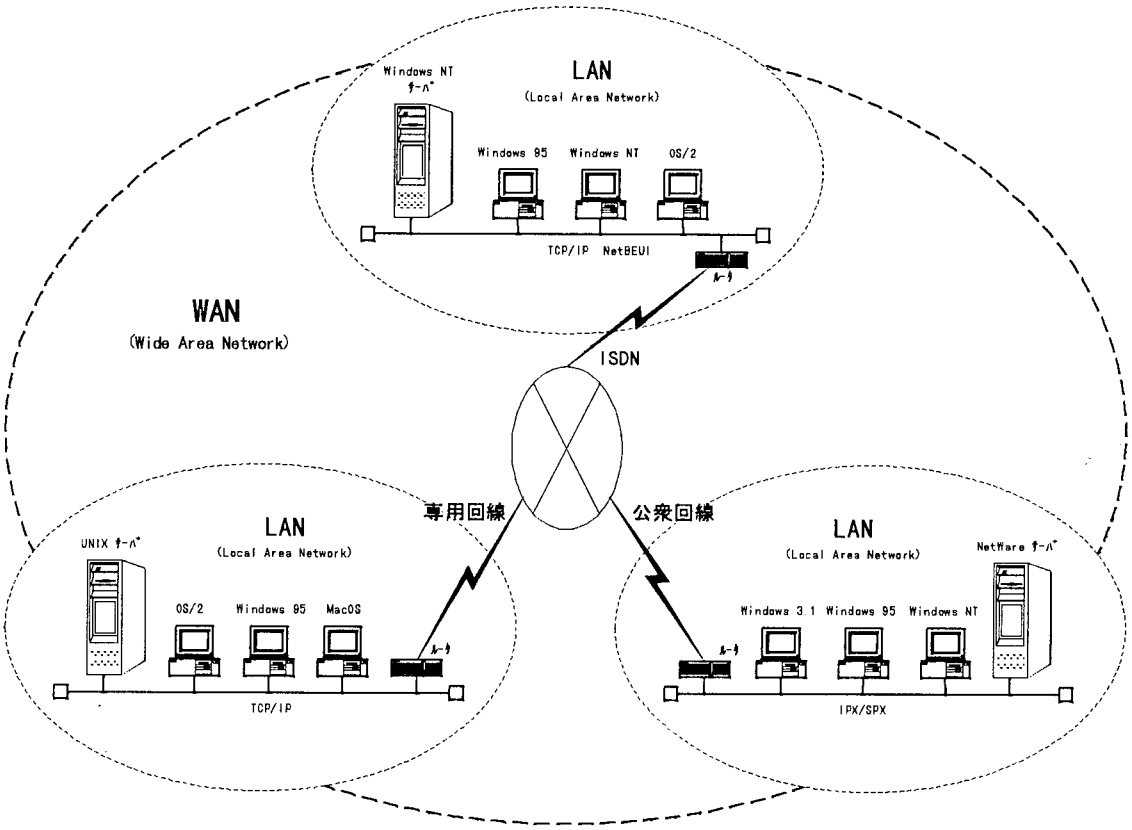
システムの開発環境では、クライアントが単一のプラットフォームの場合が多いが実動環境ではマルチ・プラットフォーム環境のことが多い。

たとえば図VI－6にあるように、OS としては Windows 3.1、Windows 95、Windows NT など、またネットワークとしては、TCP/IP、IPX/SPX、NetBEUI などの環境がある。これらの環境下でのテストが必要となる。

②マルチ・ユーザテスト

システムの開発時には、クライアントが数台という規模でアプリケーションのテストを行なうが、実動時にはクライアントが数 10 ～数 1000 台という規模の環境になる。多数のクライアントを同時に動かし、予想どうりのアプリケーションの処理性能が得られること、また複数のアプリケーションを同時に動かし、データベースの排他制御を確認するなどのテストが必要となる。

システムの開発時には、クライアントが数台という規模でアプリケーションのテストを行なうが、実動時にはクライアントが数 10 ～数 1000 台という規模の環境になる。多数のクライアントを同時に動かし、予想どうりのアプリケーションの処理性能が得られること、また複数のアプリケーションを同時に動かし、データベースの排他制御を確認するなどのテストが必要となる。



図VI－6　LANと WAN

③サーバの負荷テスト

マルチ・ユーザテストにもあったように、サーバに対して多数のクライアントから同時にアクセスした場合、また開発したアプリケーション以外のアプリケーションが 同時に動作した場合、さらに１台のサーバつまりシングル・サーバとは限らず、マルチ・サーバ環境でのテストも必要となる。

④ユーザテスト

最後にユーザ自身がテストを行なう。ハードウェア環境またデータベースの内容も本稼動と同様にして、業務の流れに基づいて処理し、データの内容を確認するテストを行なう。データの正確性、操作性などのユーザのストレステストも必要となる。

なおクライアント／サーバ上の、統合テスト環境を提供する自動化テストツールとして、マーキュリー・インタラクティブ社、SQA 社などのテストツールがある。

これらのテストツールには、クライアント／サーバ上でのテストの計画・設計・測定・評価まで管理できる機能を持ったものもある。