Ⅵ　テスト

到達目標

（1）テストの種類と目的を理解させる。

（2）品質管理について理解させる。

1　テストの種類と目的

テストは、ソフトウェア・ライフサイクルモデルの中で、プログラミングの次の工程に位置付けられる。テストとは、システム開発において、プログラムあるいはシステムの動作が仕様を満足しているかどうかを試験する工程である。

テスト工程では、以下のようなテストが実施される。

①　モジュールテスト（単体テスト）
②　プログラムテスト（結合テスト）
③　システムテスト（総合テスト）
④　第三者検証
⑤　運用テスト

このうち、①～③は、システム開発を行う側が実施するテスト、⑤は、システムを利用するユーザ側が行うテストである。また、④は、システム開発者でもユーザでもない第三者が行うテストである。

システムを開発する側のテスト設計については、Ⅴですでに述べた。ここでは、全ての種類のテストについて述べる。

2　モジュールテスト（単体テスト）

モジュールテストは単体テストとも呼ばれ、各モジュールが単体で正しく機能しているかを試験するものである。モジュールテストは、モジュールテスト設計で作成したモジューテスト仕様書にしたがって実施する。

モジュールテストには、以下の２種類の考え方がある。

（1）機能テスト

テストの対象となるモジュールの内部を全く知らないものとして、外部とのインタフェースだけに着目し、引数に適当な値を設定した後、モジュールの機能をテストする方法である。

機能テストは、ブラックボックステストとも呼ばれる。

この方法では、テスト対象のモジュールのために、ドライバとスタブを用意するのが一般的である。また、モジュールをコーディングした以外の第三者にもテストができるという特徴がある。

（2）構造テスト

テストの対象となるモジュールの内部を知っている人が、モジュールの内部構造に着目し、モジュール内の適当な位置で実行を止めて、変数の値を確認しながらモジュールをテストする方法である。構造テストは、ホワイトボックステストとも呼ばれる。

この方法では、テスト対象のモジュールのためにデバッガを使うのが一般的である。また、モジュールの中の適当な位置にPRINT文を挿入し、端末やプリンタ出力を見ながらテストする方法もある。

構造テストは、モジュールをコーディングした本人が実施しないと効率が悪いという特徴がある。

なお、モジュールテストを行った結果は、モジュールテスト成績書に書き込む。例えば、第1回目のテストを2月4日に実施した場合は、テスト月日の1番目の欄に2／4と日付を記入する。そして、テストの結果が満足できたときには、日付を○で囲んでおく。もし、テストの結果が満足できないものだったときは、○は記入しない。

2回目のテストでは、○のついていない項目だけをテストする。例えば、第2回目のテストを2月8日に実施した場合は、テスト月日の2番目の欄に2／8と日付を記入する。そして、第1回目のテストと同じく、テストの結果が満足できたときには、日付を○で囲んでおく。もし、テストの結果が満足できないものだったときは、○は記入しない。

3回目以降のテストも同様である。

表Ⅵ-1に、モジュールテスト成績書の例を示す。

3　プログラムテスト（結合テスト）

プログラムテストは、モジュールテストにおいて単体では正しく機能していることが確認されているモジュールをそれぞれ結合して、プログラム全体で正しく機能しているかを試験するものである。プログラムテストは、結合テストとも呼ばれる。

結合テストは、以下の3種類に分類することができる。

（1）ボトムアップテスト

プログラム構造図の最も下位のモジュールを開始点に、次第に上位のモジュールを結合しながらテストを行い、最終的に最上位モジュールまで結合してテストする方法をボトムアッ

プテストと呼ぶ。

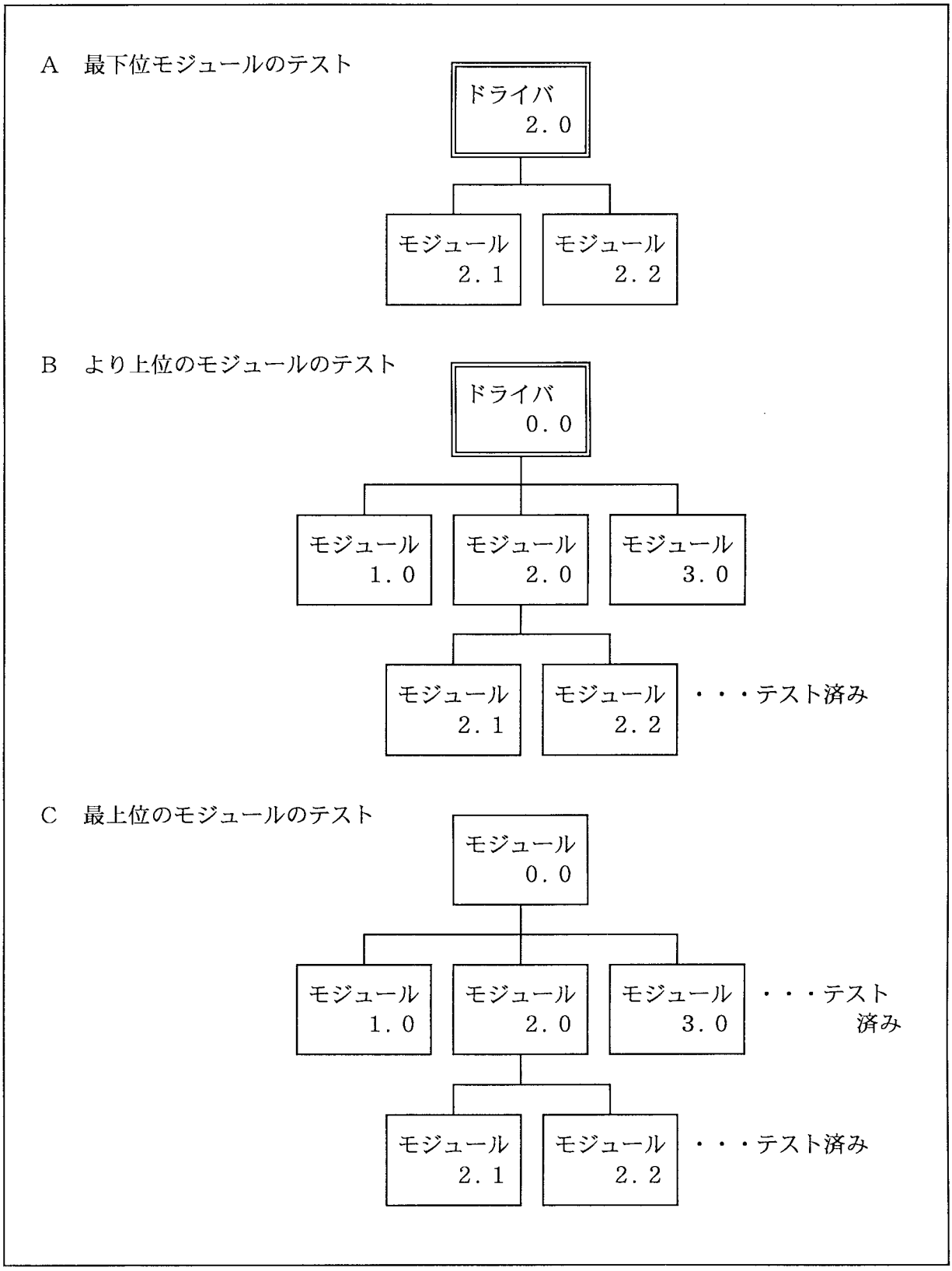
この場合、下位モジュールからテストを始めるために、テスト対象部分の上位モジュールに相当するドライバプログラムを作成しておく必要がある。

図Ⅵ-１にボトムアップテストの順序を示す。

表Ⅵ-１　モジュールテスト成績書の例

モジュール番号	２．１	モジュール名	ｘＡｄｄ	分類記号	Ｍ２１

分類	テスト内容	テスト方法	テスト結果確認方法	テスト月日			
Ｎ１	正と正の数の加算	dProcドライバから、引数 iParm1=6 iParm2=9 で呼出す。	呼出し後、引数が iResult=15 iError=0 である。	(2/4)			
Ｎ２	負と正の数の加算	dProcドライバから、引数 iParm1=-2 iParm2=4 で呼出す。	呼出し後、引数が iResult=2 iError=0 である。	2/4	2/8	(2/9)	
Ｎ３	負と負の数の加算	dProcドライバから、引数 iParm1=-55 iParm2=-9 で呼出す。	呼出し後、引数が iResult=-64 iError=0 である。	2/4	(2/8)		
Ｅ１	オーバフローエラーの処理	dProcドライバから、引数 iParm1=32766 iParm2=4 で呼出す。	呼出し後、引数が iResult=0 iError=1 である。	(2/4)			



図Ⅵ-1　ボトムアップテストの順序

ボトムアップテストには、次のような特徴がある。

イ　テストの前半部から、別々のモジュールに対して並行的にテストが実施できる。

ロ　モジュール単体の機能を確実にテストできる。

ハ　インタフェースの矛盾の発見が、テストの後半部まで遅れる。

（2）トップダウンテスト

プログラム構造図の最上位モジュールを開始点に、次第に下位のモジュールを結合しながらテストを行い、最終的に最も下位のモジュールまで結合してテストする方法をトップダウンテストと呼ぶ。

この場合、上位モジュールからテストを始めるために、テスト対象部分の下位モジュールに相当するスタブプログラムを作成しておく必要がある。

図Ⅵ-２に、トップダウンテストの順序を図示する。

トップダウンテストには、次のような特徴がある。

イ　テストの前半部では、並行的にテストを実施することが困難である。

ロ　モジュール単体の機能を、充分にテストできないことがある。

ハ　インタフェースの矛盾の発見がテストの前半部で現れ、対処しやすい。

ニ　モジュール階層の上位のモジュールは、充分なテストができる。

（3）ビッグバンテスト

モジュールテストを、ドライバとスタブを使って十分行った後、全てのモジュールを一気に結合してテストする方法をビッグバンテストと呼ぶ。このテストは、宇宙の誕生がごく短い時間に急速に起こった現象、すなわち「ビッグバン」にちなんで命名された。

図Ⅵ-３に、ビッグバンテストの順序を図示する。

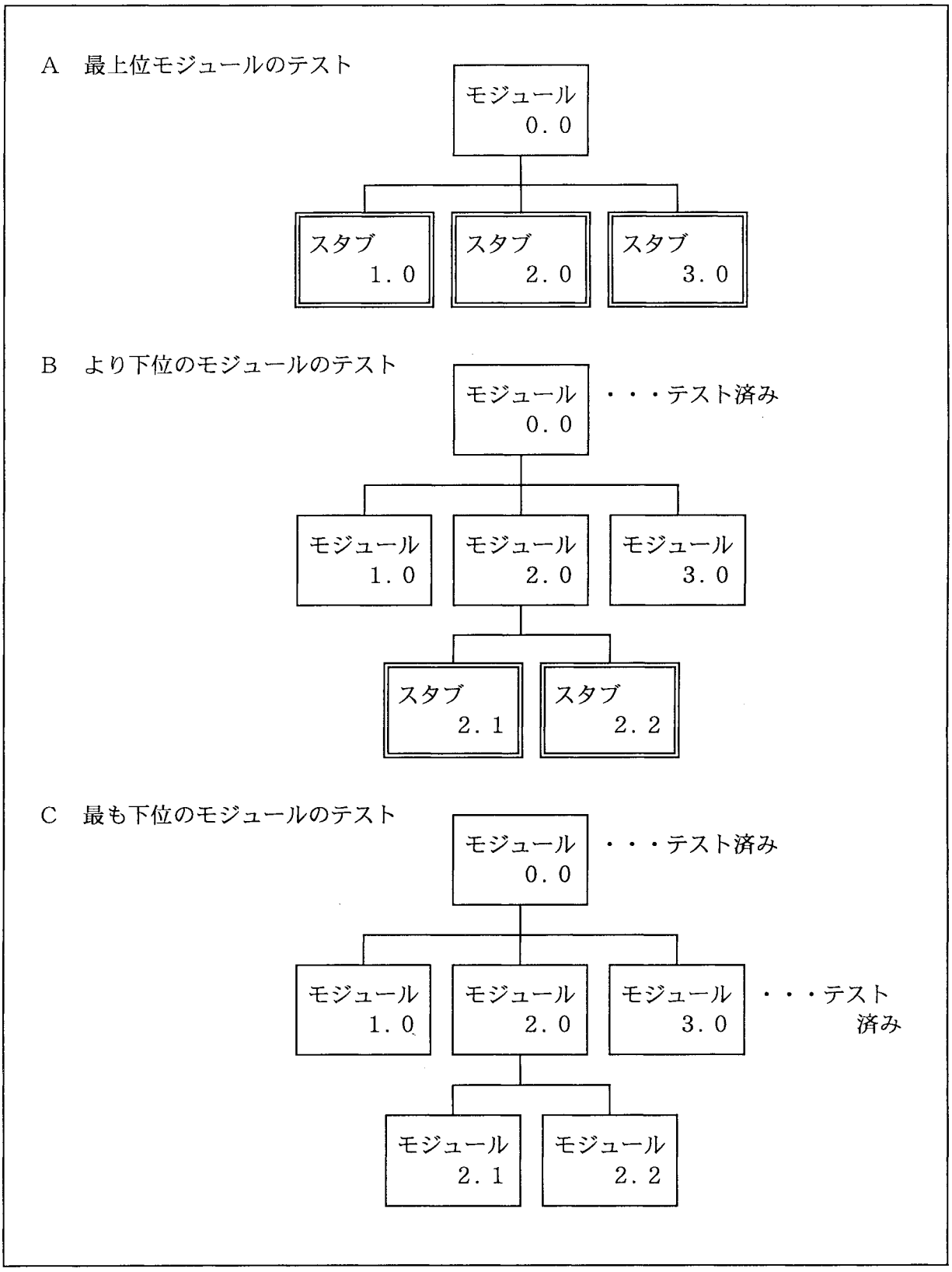
ビッグバンテストは、次のような特徴がある。

イ　結合テストの時点では、ドライバとスタブが不要である。

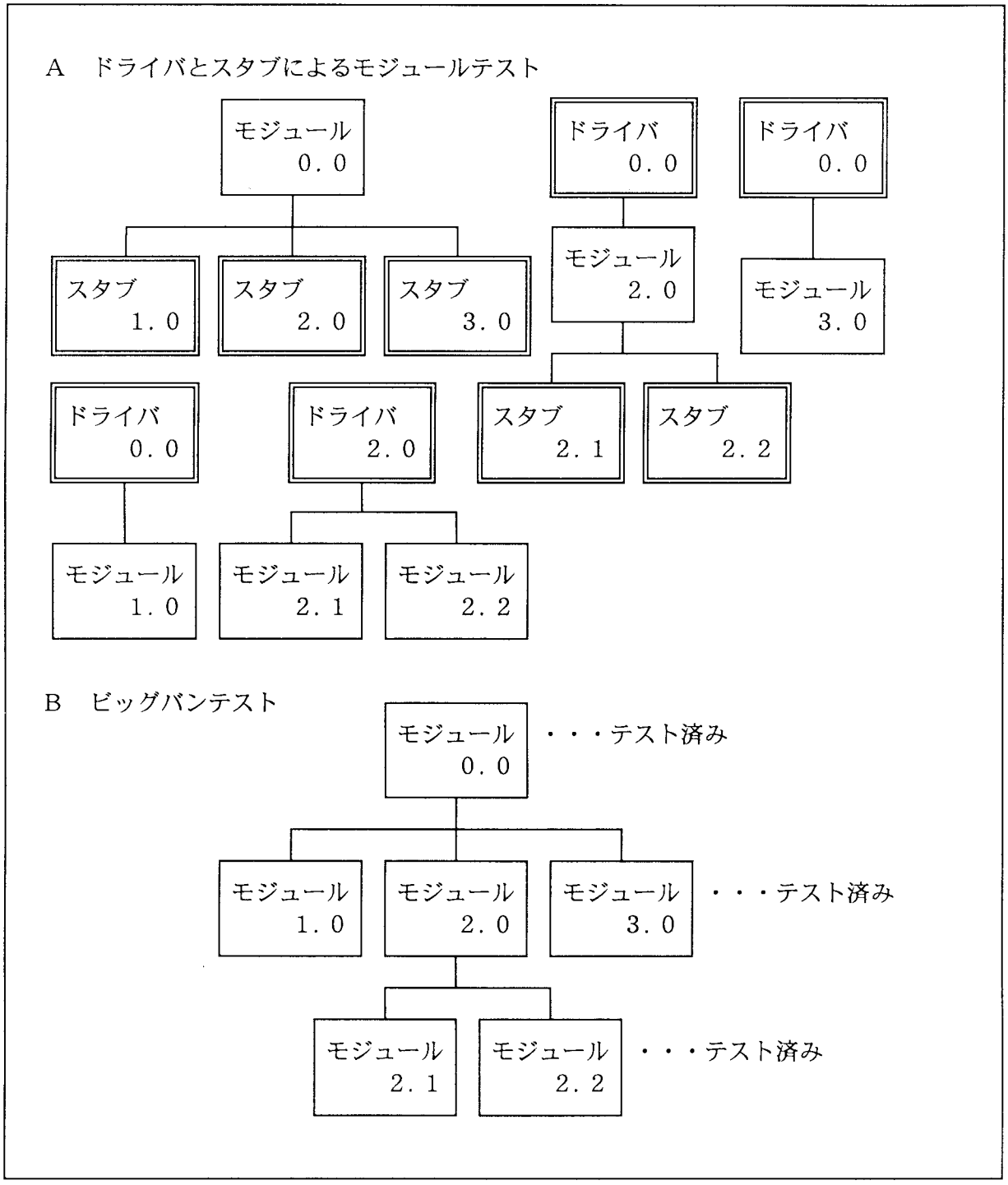
ロ　全てのモジュールの機能が、充分にテスト済みである。

ハ　特定の経路のテストは困難である。

ニ　エラーが発生した場合、発生モジュールの特定が困難である。



図Ⅵ-2　トップダウンテストの順序



図Ⅵ-3　ビッグバンテストの順序

ところで、プログラムテストでは、あるバグを修正したために、テスト済みの問題のなかった部分に別のバグが発生することがある。そのような現象を退行あるいは劣化という。

退行が発生していると、引き続き実施するプログラムテストで不具合が発生した場合に、テスト対象モジュールで発生した不具合なのか、退行による不具合なのか、切り分けができなくなる。そこで、レグレッションテストを実施する。

レグレッションテストは、バグの修正などのときに、テスト済みの問題のなかった部分に、新しい別のバグが発生していないかを確認するためのテストである。このテストは、プログラムテストで手戻りが発生することを防止するには、非常に効果的な手段である。

4　システムテスト（総合テスト）

システムテストは、プログラムテストで正しく機能していることが確認されているプログラムに対して、システムの当初の目的や性能などの要求仕様を満足しているかどうかを総合的に検証するものである。

システムテストには、以下のような種類がある。

（1）性能テスト

システムが、ユーザの要求する速度性能を満足しているかどうかを調べるテストである。システムの速度性能を考える際には、次の点を考慮しなければならない。

ｲ　スループット

単位時間当たりの処理量のことをスループット（throughput）という。バッチ処理のシステムで、１日に処理できるジョブが何個であるか、あるいはトランザクション処理システムで、１秒間に処理できるメッセージは何個であるかなどである。スループットが向上するということは、単位時間当たりの処理量が増大すること、すなわち性能が向上することを意味する。

一般には、対話型処理よりバッチ処理（一括処理）の方が、スループットが高いと言われている。それは、対話型処理のようにシステムの処理に人間の介入が多い場合、いわゆるコンピュータが遊んでいる時間が長くなるのに対して、逆に、バッチ処理のようにシステムの処理に人間がほとんど介在しない場合は、コンピュータが遊んでいる時間が減らせるからである。

銀行業務のような、多量のトランザクション処理を行う分野のコンピュータにとっては、特に重要なテスト項目である。スループットがユーザの要求を満たしていないと、最悪の場合、業務がこなせないといった事態が発生するからである。

ところで、今日では、汎用コンピュータよりもパソコンのユーザが圧倒的に多くなり、しかもパソコンの性能が急激に向上したので、スループットの向上は、もっぱらパソコンの性能向上にまかせ、ユーザにとって利用しやすい対話型処理を採用することが多い。

ロ　レスポンスタイム

コンピュータに問い合わせや要求を行ってから、応答が始まるまでに要する時間のことを応答時間あるいはレスポンスタイム（response　time）という。シングルタスクの対話型処理のシステムでは、ユーザは端末に要求を行った後、応答が画面に現れるまで待っていなければならない。人間が端末の前でいらいらせず待っていられる時間は、数秒である。もし、何十秒もレスポンスタイムがかかるとしたら、ユーザは待ちきれなくて、途中で別のことを考えてしまうであろう。

このレスポンスタイムを短縮するためには、後に述べるチューニングを行うことが必要となる。しかし、別の方法でレスポンスタイムの長さをごまかしてしまう方法も存在する。

例えば、銀行の自動現金引出し機は、ホストコンピュータと通信を行い、ユーザの残高確認や預金からの引出し処理をしなければならない。その場合、レスポンスタイムが数十秒かかることになる。ユーザの中には、自動現金引き出し機が全く応答しないと、故障したかと思い、機械を叩いたり蹴ったりする人もいる。そこで、レスポンスタイムが数十秒かかるこのようなシステムでは、画面上の矢印を動かしたり、紙幣を数えているような本来必要のない音を発生し、動作していることをユーザに知らせ、レスポンスタイムの長さをごまかすといった手法が採用されている。

なお、マルチタスクの対話型処理システムは、レスポンスタイム中でも、ユーザは他の作業を行うことができる。この意味では、マルチタスクの対話型処理システムは、きわめて優れたシステムであるといえる。

ハ　ターンアラウンドタイム

コンピュータにジョブを投入してから、完全な処理結果が得られるまでに要する時間のことをターンアラウンドタイム（turnaround time）という。ターンアラウンドタイムは、主にバッチ処理に対して用いられる。

もし、システムが、上で述べたような要求性能をわずかに満たしていないときは、チューニングという作業を行い、要求性能に達するよう努力することとなる。

チューニングとは、要求性能を満たすためシステムに小規模な改修を施すことである。

まず、無駄な処理時間のかかっている箇所を、性能測定ツールなどで特定する。次に、その部分の処理を見直し、性能を向上させることができるかどうかを検討する。例えば、繰り返し処理で時間がかかっているときは、その中の命令をより実行時間の短い命令に変更し、処理時間を短縮する。

しかし、チューニング作業を行っても、上で述べたような要求性能を全く満たすことができなかったときは、次のような手段をとる。

(イ)　コンピュータの変更

もし、同じシステムが動作可能な互換性をもつさらに高速なコンピュータがある場合合は、そちらにコンピュータを変更し、要求性能を満たす。例えば、より高速なＣＰＵをもつコンピュータや、マルチＣＰＵ構成のコンピュータに変更することによって、システムはより高速に動作する。

当然、より高速なコンピュータに変更するためには、余分な費用が発生する。この場合は、誰がその費用を負担するかが問題となる。ユーザ側が負担しない場合は、システム開発側が負担をすることも考えなければならない。

ただし、互換性をもつさらに高速なコンピュータがない場合は、この手段は使えない。

(ロ)　システムの再設計

最初からシステムの設計をやり直す。この場合、処理方式を基本から洗い直す必要が生ずる。この結果、ユーザに納期遅れという最悪の結果をもたらすことになりかねず、また、膨大かつ余分なシステム開発費用がかかることになる。

そこで、このようなことが起きないよう、要求性能を満たせるかどうか、あらかじめ試作システムを作り、そこで検証することも必要である。

(ハ)　ユーザの了解

ユーザに、要求性能を満たしていないまま、システムを稼働させることに対しての了解をとる。この場合、システム開発側がユーザに違約金を支払うこともあり、実際に、そのような例も幾つかある。

（２）負荷テスト

システムの、限界に近い、あるいは限界を超えるような負荷をかけ、それでもシステムが問題なく動作するかどうかを調べるテストである。

開発するシステムについては、動作する上での制限が明確になっていなければならない。例えば、２バイトの整数型変数を使ったデータベースでは、65535 個までの表しか作れないといった制限である。負荷テストでは、例えば、65536 個目の表を作るといったシステムの制限を超えるような項目を設定し、システムがそれでも正常に動作するかどうかをテストする。

（３）機能確認テスト

ユーザが要求した機能が、全て盛り込まれているかどうかを確認するためのテストである。

5　第三者検証と品質管理

（1）第三者検証

モジュールテスト、プログラムテスト、システムテストは、全てシステムを開発した当事者が行うテストである。しかし、システムを開発した本人では、ユーザの立場からテストを行うことができない。それは、システムが動作する前提条件を知り過ぎているために、無意識のうちに問題点を無視してしまう可能性があるからである。

システムの本当のユーザは、システムを開発した本人しか知らない前提条件に一切関係なく、マニュアルに書かれている内容にしたがってシステムを利用する。そこで、システム開発者自身が、予想もしないような問題が発生することにもなりかねない。

そこで、システムを開発した当事者でも、システムを利用するユーザでもない第三者が、システムのユーザへの引き渡し前に、客観的な立場からテストを行う。これを第三者検証という。

大手のコンピュータメーカーやソフトメーカーでは、この第三者検証のために、品質管理部門という専門の人員を割り振っている場合が多い。

（2）品質管理

ここで、品質管理と呼んでいるのは、ソフトウェアの品質管理のことである。品質管理部門では、ソフトウェアの品質を様々な基準で評価する。

その基準は、移植性、信頼性、操作性、保守性などである。

イ　移植性

あるコンピュータ環境で稼働しているシステムを他のコンピュータ環境で動作させるために、どの程度簡単にプログラムを移植できるかを表す。最近のシステムは、複数のコンピュータ環境で動作させることが多い。このようなマルチプラットホーム環境に対応するためには、移植性の高いプログラムであることが重要である。

例えば、ＣＰＵやＯＳによって整数型変数の長さが異なることがあり、MS−DOSでは２バイトで定義されているのに対して、ＵＮＩＸでは４バイトで定義されている。このようなマルチプラットホーム環境で動作するプログラムを作成するためには、プログラムの作成時から、整数型変数は２バイト、４バイトというような決め撃ちを止め、どちらでも対応できるようにしておくことが必要となる。

また、移植性の高いプログラム言語を採用することも重要である。このような努力によって、移植作業（コンバージョン作業）の工数を減らすことができる。

ロ　信頼性

システムが仕様の通りに動作するかどうかを表す。バグが少なければ、それは信頼性が高いという。システムが実稼働する前にバグを検出しておく必要がある。

ハ　操作性

システムの操作が分かりやすく簡単かどうかを表す。以前は、操作性があまり重要視されていなかったが、コンピュータが専門家だけでなく、一般のユーザが広く使用することとなり、現在では、操作性のもつ重要性はますます高まってきている。

ニ　保守性

ユーザからの保守要求に対して、容易に対応できるかを表す。保守性を高めるためには、構造化プログラミング手法やオブジェクト指向プログラミング手法の採用、ドキュメントの充実などの対策を講じる必要がある。

実際、保守という作業は、システムが世界のどこかで稼働し続けている間は継続する。一般に、ソフトウェアのライフサイクルの中で、保守はきわめて長期間にわたって続く。したがって、保守性は非常に重要な要素である。

（3）品質評価

品質評価とは、システムの移植性、信頼性、操作性、保守性などを実際に評価することである。

システム設計上の欠陥は、発見が遅れれば遅れるほど修正に手間がかかるので、早期に発見するために幾つかの手法が用いられる。

品質評価の方法としては、レビュー、ウォークスルー、インスペクションなどが用いられる。

イ　レビュー

主に設計工程において、各工程で作成された文書の内容を会議形式で検討するものである。各開発工程に区切りをつけるのがレビューである。

レビューに参加するのは、システム開発担当者、品質管理部門担当者、管理部門担当者などである。また、ユーザ自身が参加することもある。このように複数の人が、それぞれ異なった視点から検討を行うため、システム開発担当者の思い込みによる欠陥を発見しやすい。しかし、ただ漫然とレビューを行ったのでは、それなりの効果しか上げることができない。

ロ　ウォークスルー

レビューを効率よく行うための手法である。システム開発を行うメンバー自身が気づかない設計上の欠陥を複数の第三者の目で検討することによって発見する。ここでは、テストケースを用い、シミュレーションを行ってより詳細な検討を行うこととなる。

実施人数は４～５人でよいが、適切な検討が行える参加メンバーをシステム開発担当者が選出する。また、資料はあらかじめ参加メンバーに配布し、参加者メンバーが事前に資料の検討を行っておく。会議では、資料作成者が詳細を説明し、会議終了後のシステムの修正も資料の作成者が行う。

また、ウォークスルーでは、欠陥を指摘することにとどめ、修正については割愛する。なるべく効率的に会議を運営することが重要である。

ハ　インスペクション

ウォークスルーと同じく、レビューを効率よく行うための手法である。

インスペクションの場合、モデレータという責任者を１人決定する。そのモデレータが、適切な検討が行える参加メンバーを４～５人選出する。そして、あらかじめ資料を参加メンバーに配布し、参加者メンバーは事前に資料の検討を行っておく。会議では、通常、資料作成者が詳細を説明する。ウォークスルーと異なるのは、会議終了後のシステムの修正をモデレータが確認することである。また、検出されたエラーは分析し、同様の欠陥が他では発生しないよう、システム開発メンバー全員に伝達する。

ウォークスルーもインスペクションもよく似ているが、ウォークスルーでは、システムの修正を資料の作成者が行うのに対して、インスペクションでは、システムの修正をモデレータが確認する。このようにウォークスルーよりもインスペクションの方が、組織的に作業が行われ、品質管理という面では望ましい。

（4）バグの成長曲線

テストによって検出したバグの発生率を用いて、プログラムの品質管理を行うことができる。

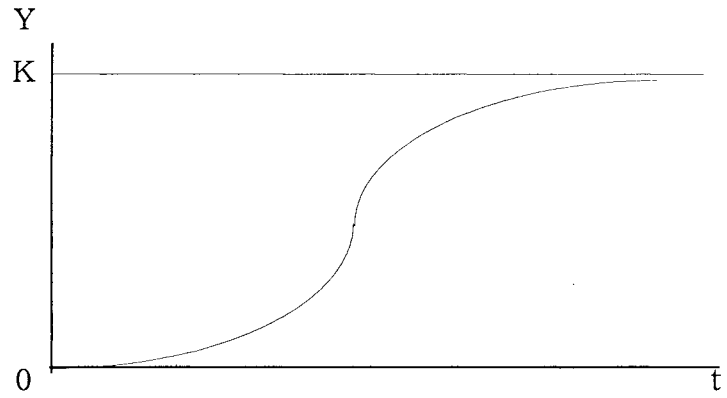
まず、システムに何件ぐらいのバグが含まれているのかをプログラムのステップ数から、テスト設計時に予測する。これをバグ検出目標値とする。

もし、バグ検出目標値を正確に決めようとするならば、一定期間、一定件数のテストを試行し、その範囲中で何件バグが発生するかによって、システム全体のバグ件数を予測すればよい。この手法を検針という。

次に、生物の成長の度合いを示すのに用いられる成長曲線を用い、テスト中に発生するバグ件数の増加を予測する。

最後に、成長曲線と実際のテストによるバグ件数成長グラフを比較しながら、成長曲線上で成長の伸びが止まるのと同様、バグ件数成長グラフ上でバグの件数の伸びが止まったら、システムのバグが出尽くしたと判断する。

ここで使用される成長曲線は、ロジスティック曲線や、コンペルツ曲線である。図Ⅵ-４に成長曲線を示す。



a　ロジスティック曲線

$Y = k / (1 + me^{-at})$

$a > 1$，$m > 0$

Kは、$t \to \infty$のときのYの収束値

b　コンペルツ曲線

$Y = Ka^{b^t}$

$b < 1$，$a < 1$

Kは、$t \to \infty$のときのYの収束値

図Ⅵ-4　成長曲線

6　運用テスト

システムを使用するユーザが、システムの実際の運用と同じ条件を作り出し、システムの性能に問題がないか、要求した機能が盛り込まれているか、操作性に問題はないかといった点をテストするものである。この場合、もし、現在稼働中のシステムが既に存在する場合は、このシステムの運用に、極力影響を与えないように実施しなければならない。

運用テストは、以下のような内容からなる。

（1）受入れテスト

システムを開発した側が、システムを使用するユーザから承認を得て、受け入れてもらうためのテストである。テストのためのデータは、ユーザ側が用意し、要求通りの結果が得られるかをユーザ側が行う。

（2）導入テスト

実際の動作環境にシステムを導入して、システムが正しく稼働する環境であるかをテストするものである。

（3）実地テスト

コンピュータメーカーやソフトメーカーで開発した製品を、開発部門以外の部門で評価、確認してもらうテストである。実際の業務で使用し、評価してもらうことから、実地テストと呼ぶ。最近では、製品の発売前に、機能に制限のある開発途中の製品を、特定のユーザに使用してもらい評価を受けるアルファテストや、開発完了直前の製品を、より多くのユーザに使用してもらい評価を受けるベータテストを実施することも多い。

企業では、最近、ベータテストをうまく活用する例が増加している。

例えば、雑誌の付録のCD−ROMにベータテスト用のソフトウェアを入れ、不特定多数のユーザに評価してもらうというケースである。

ただし、製品発売後もユーザがベータテスト用のソフトウェアを使い続けると、製品の売り上げが減少するため、ベータテスト用のソフトウェアの使用期限を設定し、その期限を超えて利用できないようにすることが一般的である。