◇テストの技法

No.	テスト技法名	手法	効果	留意点
3.9.1	経験及び直感に基づいた技法			
3.9.1.1	アドホックテスト	テスト設計行為を伴わずにテストを実施する技法である	-	・アドホックテストを行う目的を明確にする。例えば、故障を検出するのではなく、出荷前に高品質であることを最終確認するために追加で実施されることがある ・文書化されたテスト設計を伴わないので、テスト実施の記録も残らない場合が多い。そのためテストの目的を満たさず、単なるやみくもなテストとなる傾向がある。 ・アドホックテストで障害が見つかったときには、なぜこの障害を見つけるための、本来のテストケースがなかったのかを分析しなければならない
3.9.1.2	探索的テスト	テスト対象製品の学習、テストの設計、テストの実行を同時並行に行う技法である <目的＞あらかじめテストの順序が決められず直前のテスト結果に従って、次のテストを選択する必要があるとき、あるいは開発の早期段階で短期間で短時間に迅速な品質フィードバックが必要な場合に、スクリプト試験	継続的に調整、変更される計画プロセスに柔軟に対応すべきテストプロセスにおいて、スクリプトテストの欠点であるテスト計画の固定化を補うことができる。また、問題を検出した場合、さらに障害の規模、範囲、バリエーションなどを探索することにより開発者に十分なフィードバックが行える	スクリプトテストと探索的テストは補完的に用いるとよい
3.9.2	仕様に基づいた技法			
3.9.2.1	ブラックボックステスト	テスト対象のソフトウェアを暗箱（ブラックボックス）に見立てて、暗箱の中で動作の良否を、機能に対する入力と出力（結果）という外部に見える現象から判断するテストの総称である。仕様に基づいた技法と同義である <目的>仕様をもとにテスト対象となるソフトウェアを入力と出力から動作の良否を判断する	プログラムのロジックに関係なく、入力と出力を仕様で判断するため客観的な良否判定ができる	プログラムロジックに基づいた仕様があることに気づかないことがあり、テスト工数を増やしてもソースコードに対する網羅率が上がらない状況に陥る可能性がある。このため、ロジックを考慮するグレーボックステストの利用や、コンポーネントレベルテストにはホワトボックステストを併用するなどの配慮が必要である
3.9.2.2	同値分割	テスト対象が同じ振る舞いをすると仮定できる入力や出力などの値の集合や範囲を「同値クラス」としてまとめ、同値クラス内の代表値のみをテストすることによりテストの数を削減する技法である <目的>テスト対象が同じ振る舞いをする範囲を特定し、合理的にテストの数を減らす	同値クラス内については代表値によるテストだけで十分であるという根拠により、テストの数を削減できる	プログラムロジックや複合条件による振る舞いの違いがあるため、同値クラスの見極めが難しい場合がある。グレーボックステストの考え方を利用するとよい
3.9.2.3	境界値分析／境界値テスト	プログラムの誤りが同値クラスの境界部分に存在しやすいことに着目し、境界値を分析してテストを設計する技法である <目的>同値クラスの境界部分に生じやすいプログラムの誤りを発見する	プログラム内のループ終了条件や判定条件の不等号式などの誤りを検出できる	プログラムによっては、複数条件で判定が変わったり、内部条件で隠れた境界値が存在したりすることがある。単一の同値クラスや外部仕様のみに着目した境界値分析では、こういった誤りは見つけ出せないことに注意が必要である
3.9.2.4	デシジョンテーブルによるテスト	テスト対象の仕様をデシジョンテーブルと呼ばれる表にまとめ、これに基づいてテストケースを作成する技法である <目的>テスト対象の仕様を論理的に整理してテストケースを導出することによりテスト漏れを防止する	デシジョンテーブルを用いることでプログラムの仕様を論理的に整理できるため、的確なテストケースが作成できる	動作に影響を及ぼさない条件が存在する列を一つにまとめることによりデシジョンテーブルを圧縮することができ、ケースが削減できるが、プログラムのロジック（条件判定の順序）が変わる場合は圧縮できない。誤って圧縮した場合はテスト漏れとなる
3.9.2.5	原因結果グラフによるテスト	プログラム仕様を、入力（原因）と出力（結果）の論理関係、及び原因間の関係（制約条件）を原因結果グラフと呼ばれるグラフで表現し、それに基づいてテストケースを作成する技法である <目的>テスト対象プログラムの仕様を、原因と結果に着目して系統的に整理してテストケースを作成する	テスト仕様を導出する際の偏りや漏れが防止できる。また、副次的にプログラムの仕様のあいまいな部分が検出できる	対象プログラムが多機能な場合には、原因結果グラフの作成に時間を要する可能性がある。このため、適用に際しては、経済的な合理性も考えて、対象範囲や詳細化レベルを決めておく必要がある
3.9.2.6	状態遷移テスト	テスト対象プログラムの仕様を状態遷移表を使って整理し、これに基づいてテストケースを作成する技法である <目的>テスト対象プログラムの状態の遷移に着目して漏れのないテストを実施する	機能及び状態の遷移に関するテスト漏れを防止できる	状態以外の事象は状態遷移図からは判断できないため、デシジョンテーブルによるテストなど他のテスト技法と組み合わせる必要がある
3.9.2.7	ランダムテスト	ランダムに抽出したテストデータによりテストを行う技法である <目的> ・乱数を用いて大量のテストデータを簡単に生成する ・規則性のない入力によるランダムテストの結果から統計的にソフトウェアの信頼性を予測する	乱数を用いてテストデータが生成できるので、テスト自動化がしやすい	ランダムに生成されたテストの結果判定が難しい場合があるので、機械的に結果判定できる仕組みを作っておく必要がある
3.9.2.8	モデルベースドテスト	テスト対象システムの振る舞い（仕様）をモデル化し、そのモデルに基づいてテストケース、テストスイート、テストスクリプトなどを自動生成する技法である ＜目的＞テストの設計、資材作成（テストスクリプト、結果判定情報、テストデータなど）、実行を効率化して確実化する	テストの効率性や自動化に加え、設計のあいまいさの検出も期待できる	・テスト技術者にモデル化に関するスキルが求められる ・モデル化、及びそのモデルからテストの生成、実施、結果の評価を行うためのツールが不可欠である
3.9.2.9	要因分析技法[富士通]	ソフトウェアの外部仕様を、要因（因子）とその要因がとり得る状態（水準）の形式で整理して「要因分析表」を作成し、それに基づいてテストケースを設計する技法である <目的>外部仕様に基づくテストケースを設計する	テスト条件を抽出・整理する過程、組み合わせを決定する過程が定型化でき、質の高いテスト設計ができる。また、テストレビューもしやすくなる	一つの要因分析表で因子の数が多くなりすぎると、組み合わせが膨大になったり、テストの実施がしにくい場合がある。その場合はテスト分類のステップに立ち返って、分割方法を見直すとよい。
3.9.2.10	CFD技法	テスト対象の入力（原因）と出力（結果）を同値に分割した同値分割図をもとに原因と結果を"流れ線"でつないだ原因流れ図（CFD）を作成し、これに基づいてデシジョンテーブルを作成してテストケースを抽出する技法である <目的>複雑な論理関係をもつ仕様のプログラムに対して、網羅的でかつ少ない数のテストケースを作成する	・同値分割図として集合論で用いられているベン図の記法を使うことにより、抽出した同値の補集合の有無がチェックでき、同値の漏れを防止できる ・実装情報を利用する事により、テストケース数が増大することを防止できる	本技法は継続的に改善されており、左述の方法はCFD技法（2008年版）である。適用する際には最新情報を確認するとよい
3.9.3	コードに基づいた技法			
3.9.3.1	ホワイトボックステスト	テスト対象の内部構造に着目したテストの総称であり、コードに基づいた技法と同義である <目的>ソフトウェアの網羅構造に依存する障害をみつける	論理の矛盾に関連する障害を発見できる	・網羅性基準を厳しくし、網羅率を上げることでテストの信頼性は向上するが、反面、テストケースの数が膨大になる。そのため、要求される品質に応じて網羅基準や網羅率を検討する必要がある ・ホワイトボックステストは主に単体テストで実施されるが、コードだけでなくモジュールや機能に着目することで、他の結合テスト、システムテストといったテストレベルでも実施できる
3.9.3.2	制御フローテスト	プログラムの制御構造をフローグラフに表現し、グラフを網羅するようにテストを設計する技法である ＜目的＞プログラムが意図したパスと異なるパスを実行することによって発生する障害を検出する。また、網羅基準を用いることにより、実施したテストの網羅性を測定する	採用した網羅率に対して、テストによってどれだけ網羅できたかを判定し、テストの実施率を評価できる。また、テストが実施されていない命令文や分岐文を確認し、不足しているテストケースを追加できる	・高い網羅率を満たそうとするとバグ数が膨大になる可能性がある ・プログラムのコードが正しいという前提でテストを行うため、誤った仕様に基づいてプログラムコードが書かれている場合に問題を検出できない ・網羅率はあくまで目安であり、値が100%になったとしてもテストが十分とはいえない
3.9.3.3	データフローテスト	プログラムの制御フローをもとに、プログラム中の変数のライフサイクル（定義、使用、消滅）に着目してパスを選択することでテストを設計する技法である <目的>データの状態に関連した一連の事象を解析し、データの使用、データの形式、種類や初期値に関する障害を検出する	ソースコード中に占めるデータ宣言文の割合の増加とともに、データに関する障害も増加しており、そのような障害の検出には効果的である	データフローの変則の内容や傾向は、アプリケーションにより異なるため、個々のプログラムごとに確認する必要がある
3.9.3.4	トランザクションフローテスト	システムの処理（トランザクション）をフローグラフに表現し、グラフを網羅するようにテストを設計する技法である <目的>トランザクションフローの妥当性を網羅的に確認する	トランザクションをモデル化して網羅的に確認できる。トランザクションフローを作成する過程で、システムの仕様を明確にする効果もある	外部システムよりもたらされる障害などによる異常系トランザクションなど、テストで作り出すのが困難なトランザクションがある
3.8.4	フォールトに基づいた技法			
3.9.4.1	エラー推測テスト	対象プログラムに発生し得るエラーを推測し、そのエラーを見つけ出すための特別なテストケースを設計する技法である <目的>特別なテスト設計を用いず、エラーの検出力の高いテストケースを作成する	特定のエラーに焦点を当てたテストケースを効率的に設計できる	・推測するエラーのリストの充実度によって、テストの効果が大きく左右される ・他のテスト設計技法と補完的に用いるとよい
3.9.4.2	ミューテーションテスト	テスト対象のプログラムを一定の規則に基づいて変更あるいは変異させたときに、その変異をテストで検出できるかどうかにより、テストケースの集合であるテストセットの十分性（障害発見能力）を測る技法である <目的>テストセットの十分性を評価する	テストセットの十分性が評価できることにより、テストセットの改良、ひいてはテスト対象のプログラムの品質向上が図れる	・変異規則や変異の個所が適切でないと、テストセットの評価精度は落ちる ・自動化されたミューテーション解析システムが不可欠である

3.8.5	利用に基づいた技法			
3.9.5.1	運用プロファイルによるテスト	ソフトウェアが実際に運用される際にどのように利用されるかを確率分布により表現した利用パターン（運用プロファイル）をもとに運用時と同じ条件下でテスト対象を動作させ、ソフトウェアの信頼性を評価する技法である <目的>実際の運用に近い条件でテストを実施することにより、本稼働後のソフトウェア信頼性の予測の確度を高める	本稼働後のソフトウェア信頼性を予測するテストが体系的な方法で実施できる	・運用プロファイルをもとにしたテストでは、使用頻度が低いと判断された機能のテストが不十分になり、意図しないソフトウェアの使われ方による信頼性の低下が懸念させれる。運用プロファイルは典型的なユーザだけでなくすべてのユーザーをモデル化するとよい ・通常、実際の利用者が直接起動るもの以外は、明示的に運用プロファイルとしてモデル化されないので、他のソフトウェア、ハードウェア、ドライバーが関係する動作環境の考慮が不足する。これらも含めた運用プロファイルを作成するとよい
3.9.5.2	ローカライゼーションテスト	開発国以外の国で、あるいは開発国以外の言語で適切に動作するかどうかを確認するテストである <目的>ソフトウェアを新たな地域に適用させるために、多国語対応のソフトウェア開発において、各国語及び各国の利用環境に合わせたローカライズ（機能の追加・変更）を行い、その機能仕様通りに正しく動作することを確認する	輸出入などにより開発国以外の国で使用されるソフトウェアについては、変更すべき観点の洗い出しと変更作業に見落としがなくなる	・プログラムがローカライズしやすいように作られていない場合、コードを書き直すことになり、新たな障害を埋め込む恐れが生じる ・同一言語であっても、国や地域によって意味が異なることがあるため注意を要する ・言語の変換の必要がなくても、文化や習慣の違いを考慮して変更が必要となる場合がある ・テストはローカライズ先の言語や習慣に堪能な人が行うとよい ・広い意味でオフショア先からの受入テストもローカライゼーションテストの性格を持っている
3.9.5.3	ユーザー環境シミュレーションテスト	顧客納入前に顧客システムに近いシステムを構築して実施する最終的な試験のことである <目的>顧客に提供するソフトウェア製品、ハードウェア製品、サービスなどを含めた、システムとしての品質を保証する。	社会や顧客ビジネスに大きな影響を与えるシステムのトラブルや初期障害を未然に防止し、その結果として社会や顧客からの信頼感を向上できる	◎顧客システムに近いシステムの構築と差異の認識 ハードウェア構成、ソフトウェア構成、ネットワーク環境、データベース、運用、負荷、障害などの観点でできるだけ顧客の実運用に近い環境を実現すること。それがシステム検査の成否にかかわる。その上で実際の顧客システムとの違いを明確にして検査すること ◎システム検査の対象とすべきシステムの選択 すべてのシステムに対してシステム検査が必要なわけではない。社会的影響度、顧客ビジネスへの影響度、システムの新規性、性能要求が厳しいものなど、あらかじめ選定基準を明確に信頼性て、システム検査が必要なシステムを漏らさないようにするとよい
3.9.5.4	整合性確認テスト	ソフトウェア製品検査の最終段階でソフトウェア間の整合性を確認するテストのことである <目的>ソフトウェア間の整合性を確認して、提供またはサポートするソフトウェア製品群での典型的なシステム構成での、重大な障害や初期障害の発生を防ぐ。	システム的な観点でのソフトウェア製品の品質向上が図れる	・関連するソフトウェアが多い場合、確認すべきインタフェースも多岐にわたる。個別のインタフェースの確認は事前に確実に終わらせておくこと。また何を確認するか優先順位を明確にすること ・製品の組み合わせとして他社製品を含むことも多いが、他社製品では多くの場合内部仕様はブラックボックスであり、検査観点の設定や問題発生時の解析が自社製品に比較して難しい。事前の情報収集、ノウハウの蓄積を徹底する必要がある ・整合性確認テストでは、ユーザー環境シミレーションテストとは異なり、特定の顧客の設備、プログラム、データまでは借用しない場合が多い。いかに特異な顧客環境を包含した環境を作りだすかが重要である
3.8.6	ソフトウェアの形態に基づいた技法			
3.9.6.1	オブジェクト指向テスト	OO（オブジェクト指向）パラグラムによって開発されたソフトウェアの特徴を考慮して行われるテストのことである <目的>オブジェクト指向パラダイムで開発されたソフトウェアに対して、開発手法を考慮したテストを行う。	オブジェクト指向パラグラムで開発されたテスト対象に対して、既存のテスト技法を適用し、かつOO特有のテストを行うことができる	・テスト技術者にモデルを理解するスキルが求められる ・クラス間の構造（設計構造）に焦点を当てたテスト設計に力点をおく必要がある ・継承したからといって、テスト結果もそのまま流用してよいわけではない
3.9.6.2	Webシステムのテスト	Webシステムの特徴を考慮してテストする <目的>Webシステムのテストでは、Webシステムの特徴を考慮してテストする。	Webシステムの構造や環境に起因する障害を検出できる	・利用環境の組み合わせをすべてを考えると膨大な数になり、すべてテストするのは現実的ではない ・直交表などを用い、組み合わせパターンを減らす工夫が必要である ・セキュリティ、パフォーマンス、使用性など専門的な知識を必要とするテストが多い
3.9.6.3	GUIテスト	GUIの各ボタンなどが動作することだけでなく、GUIのウィンドウ間の遷移を状態遷移図に表現し、遷移パスを網羅するようにテストを設計する。 <目的>GUIに対するアクションやイベントの動作と、動作後の遷移先に着目し、設計意図通りの一連のGUI遷移パスにてアクションやイベントが発生することを確認する	GUIの遷移を明確にし、かつ遷移パスを通るテストを実施することで、網羅性を確保できる。また、遷移パスが複雑すぎる場合には、GUIの構造を見直すたもの判断材料となる	あくまでGUIの遷移に着目したテストであり、GUIののデザインや使いやすさについてはユーザビリティテストなどにより別途確認する必要がある
3.9.6.4	サーバーサイドのテスト	Webアプリケーションをはじめとするクライアント・サーバー型のシステムにおいて、サーバーサイドのアプリケーションの特徴や動作上の制約を考慮したテストを行う <目的>クライアント・サーバーシステムのサーバーアプリケーションやサーバーコンポーネントに対してテストを行う	ユーザーインタフェースやメッセージといったクライアントの機能にとらわれず、それらを模擬することでサーバーの評価ができる。サーバー側のアプリケーションの動作を記録し、障害の再現を容易にするための方法を提供できる	サーバーアプリケーションあユーザーが直接触れることのない構造になっているため、エラーが発生しているした場合に、発生時期、エラー発生の経緯、使用ユーザーなどの状況が分からない場合が多い。このため、テスト実行時には、インターフェースレベルでの妥当性確認だけでなく、アプリケーションの動作を追跡、分析、監査するためのログファイルの作成や、サーバーリソースのモニタリングを行い、サービスの障害分析やパフォーマンスの評価を行うとよい
3.9.6.5	データベーステスト	データベースにアクセスするアプリケーションの機能テストとともに、データの内容とデータベースの完全性をテストする <目的>データベースにデータが適切に格納され、更新、検索できること、またそのデータの内容の正しさやデータの破損がなくデータベースの構造が正しいことを確認する	-	ブラックボックステストにおいて、返される結果が正しくても、データベースに書き込まれた内容が正しいとは限らない。また、返される結果が誤っていたとしても、その原因初めてデータベースにあるとは限らない。その一因としては、クライアント・サーバー型のシステムにおいては、クライアント、サーバー、そしてデータベースと層が分かれており、各層の間での通信や情報の受け渡し、サーバー層の中でのデータの変換など、さまざまな中間処理が存在することが挙げられる。また、同一トランザクション内での処理の中断や、複数のトランザクションを同時に処理する場合などを考慮したテストが必要である
3.9.6.6	並行プログラムテスト	並行処理における制御方法、あるいは特徴的な誤りに着目してテストを行う <目的>同時並行的に実行されるプログラムによる複数の処理に対してテストを行う	-	・アプリケーションの動作は環境に依存する。例えば、スレッドは常に同一の順序で実行されるわけではない。また、サーバーのプロセッサの数によっても動作は変わり得る ・マルチスレッドの問題を見つけるには、問題が発見されるまで単純なテストを何度も繰り返し実行する必要がある
3.9.6.7	プロトコル適格性テスト	通信プロトコルに準拠しているかどうかを検証するためのテストを行う <目的>ネットワーク構成を持つコンピューターシステムにおいて、コンポーネント（ノード、エンティティなどプロトコルにより呼称はさまざま）間の有線または無線通信が通信プロトコルに則って行われているかをテストする	通信プロトコルに準拠していることを検証することにより、異ベンダー間のコンポーネントと通信できることを保証したり、通信関連法規に則っていることの証明を認定機関より得たりするための材料をえることができる	通信プロトコルでは、代表的な例外・代替シーケンスのみが示されることが多く、想定され得るシーケンスの組み合わせすべてが網羅されているわけではないことが多い。またプロトコルを実装する手段を詳細に指定しないことも多く、ベンダー間でプロトコルに対する解釈が異なる場合も多い。このような場合、テストの目的に基づいて、どのようにプロトコルの規定を判断するかの基準を持たないと、テストの期待結果が不明確となる
3.9.6.8	実時間のテスト	実時間アプリケーションに対し、割り込み処理やデータのタイミング、タスクの並列性などを考慮してテストを行う <目的>実時間システムの特徴を考慮したテストを行う	-	・実時間のシステムでは、ある状態ではテストが成功しても、別の状態では失敗する可能性がある ・ソフトウェアのテストの際に、ハードウェアの障害による影響を考慮しなければならない
3.9.6.9	モバイルアプリケーションのテスト	常時携帯される端末の特徴を考慮してテストを行う <目的>モバイル端末の特徴を考慮してモバイルアプリけーションのテストを行う	モバイル端末に特有の制約や特徴を考慮したモバイルアプリケーションのテストが行える	個人向けモバイル端末のアプリケーションの場合は、提供開始直後から多数の利用者が感想を発信することが多く、評価レポートなどに要約されて全世界で共有される。このため、出荷初期の品質に関する評判がビジネスの観点からも必要である。重大な障害が発生していないことに加えて、利用者の期待を満足するような性能の重要性が相対的に高い

3.9.7	組み合わせの技法			
3.9.7.1	直交配列表（実験計画法）を用いたテスト	組み合わせテスト設計において、すべての組み合わせを網羅するのではなく、実験計画法で用いられる直交配列表の性質「任意の二つの因子間ですべての水準の組み合わせが同一回数存在する」を利用して、二つの因子間の組み合わせを網羅したテストケースを設計する技法である <目的>合理的にテスト数を削減し、実施可能な数の組合せテストを設計する	ソフトウェアの障害原因は、一つの因子によるシングルモード・フォールト、及び二つの因子の組み合わせによるダブルモード・フォールトが多いという調査報告事例があるが、本技法を適用すればこの範囲の障害を検出するテストが確実に実施できる	・三つ以上の因子の組み合わせは網羅されないので、必要に応じて三つ以上の因子を組み合わせを考慮したテストケースの追加が必要である ・組み合わせを決定する際、禁則と呼ばれることがある組み合わせ不能な関係などの因子間の関係を考慮した処理が必要である ・直交配列表を用いたテストとAll-pair法を用いたテストを比較すると、二因子間の組み合わせの網羅性は同等、テストケース数はAll-pair法の方が少ない、三つ以上の因子の組み合わせの網羅性は直交配列表の方が高い。三因子以上の組み合わせの網羅性をある程度確保した場合は直交配列表を用いたテストが有効である
3.9.7.2	All-pair法を用いたテスト	組み合わせテスト設計において、すべての組み合わせを網羅するのではなく、「任意の二つの因子間ですべての水準の組み合わせが一つ以上存在する」という条件で、二つの因子間の組み合わせを網羅したテストケースを設計する技法である。Pairwise法ともいう <目的>合理的にテスト数を削減し、実施可能な数の組合せテストを設計する	・ソフトウェアの障害原因は、一つの因子によるシングルモード・フォールト、及び二つの因子の組み合わせによるダブルモード・フォールトが多いという調査報告事例があるが、本技法を適用すればこの範囲の障害を検出するテストが確実に実施できる ・直交配列表を用いた場合よりも少ないテストケース数で二つの因子の組み合わせを網羅できる	・三つ以上の因子の組み合わせは網羅されないので、必要に応じて三つ以上の因子を組み合わせを考慮したテストケースの追加が必要である ・組み合わせ決定のためのツールには組み合わせ不能な関係などの因子間の関係を考慮した処理が必要である ・All-pair法を用いたテストと直交配列表を用いたテストを比較すると、二因子間の組み合わせの網羅性は同等、テストケース数はAll-pair法の方が少ない、三つ以上の因子の組み合わせの網羅性は直交配列表の方が高い。テストケース数の少なさを優先する場合はAll-pair法を用いたテストが有効である
3.9.8	リスクに基づいた技法			
3.9.8.1	テストマネジメントにおけるリスクベースドテスト	プログラムの機能や特性ごとに問題（品質リスク）が発生する可能性（確率）と影響度を分析して、テスト範囲や優先度、リソースの割り当てといったテスト計画を立案し、それらをマネジメントする技法である<目的>品質リスクを考慮してテストをマネジメントする	リスクに基づく系統的なテストマネジメントが行える	適宜ステークホルダー（利害関係者）を巻き込んでリスク分析やテスト計画のレビューを行うのがよい
3.9.8.2	テスト設計におけるリスクベースドテスト	プログラムが障害を起こしそうな状況（リスク）を想定し、それらの障害が実施に発生するか否かを確認するためのテストケースを設計する技法である <目的＞リスクを考慮してテストケースを設計する	障害の起きやすい部分、プログラマーの人的要因を含む障害を起こしやすい要因を考慮した的確なテストケースが設計できる	機能網羅的な技法ではないので他のテスト設計技法と補完的に用いるのがよい
3.9.9	テスト技法の選択と組み合わせ			
テストを構成すると五大要素		テストの実施者／網羅性（カバレッジ）／問題やリスク／作業内容／結果の判定方法（合否判定）		
3.9.9.1	機能的なテスト設計と構造的なテスト設計の組み合わせ	（目的）機能的なテスト設計と構造的なテスト設計とのバランスを取り、テストの質と生産性を両立させる	設計モデルをもとにしたテスト設計やグレーボックステストを行うことで、テスト対象の規模が能大になっても扱う情報量を増大させず、テスト精度を向上できる	テストを設計する技術者がソフトウェア設計を十二分に理解しなくてはならないし、ソフトウェア設計を担当する技術者がテストを十二分に理解しなくてはならない
3.9.9.2	確定的なテスト設計と非確定的なテスト設計の組み合わせ	（目的）確定的に設計するテストケースの要素や粒度と、テスト実施時まで設計を遅らせる要素や粒度とのバランスシートを取り、テストの質と生産性を両立させる	テストの質と生産性を両立させるとともに、障害の作り込まれやすさの知見を蓄積したり、要求仕様や設計、実装に対する評価眼ができる	障害の作り込まれやすさの知見や要求仕様・設計・実装に対する評価眼などのノウハウを得ることなくランダムテストなどの非確定的なテスト設計を行うと、テスト設計をしている組織の組織能力が向上しない