# 第 20 回初級ソフトウェア品質技術者資格試験 問題の解説

※出題した問題の一部を解説付きで公開いたします。正しい知識の習得と理解の深耕にご活用ください。

■問題：

CFD 技法に関する次の文章の（　　）の中にあてはまる語句として、もっとも適切な組合せを選べ。

CFD 技法とは、テスト対象の入力と出力を（　①　）に着目して分割した（　①　）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を作成し、これに基づいて（　②　）を作成してテストケースを抽出する技法である。テスト対象の実装情報を用いて合理的にテストケース数を削減する、いわゆる（　③　）のアプローチが採られている。

【選択肢】

| | | | |
|---|---|---|---|
| ア | ①境界 | ②状態遷移図 | ③グレーボックステスト |
| イ | ①同値 | ②デシジョンテーブル | ③グレーボックステスト |
| ウ | ①境界 | ②デシジョンテーブル | ③ブラックボックステスト |
| エ | ①同値 | ②状態遷移図 | ③ブラックボックステスト |

【正解】

イ

【出題分野】

この問題は、SQuBOK（ソフトウェア品質知識体系ガイド）第 2 版 樹形図の「第 3 章 ソフトウェア品質技術」の「3.9 テストの技法」からの出題である。

この問題は、仕様に基づいた技法のひとつである CFD 技法について基本的な考え方を確認する問題である。

【選択肢の解説】

①同値と境界

同値とは、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である。それらの集合や範囲を「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技法を「同値分割」という。同値分割により、テストの数を削減することができる。同値分割は、同値クラス内のすべての値をテストするのではなく、代表的な値でテストすればよいという考え方に基づく方法である。

境界とは、同値クラスの下限、上限の値を指す。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（プログラム内のループの終了条件や判定条件の不等号などに誤りが生じやすい）ことに着目し、境界値を分析してテストを設計する技法を「境界値分析／境界値テスト」という。

②状態遷移図とデシジョンテーブル

プログラムの実行時の進行段階を「状態」として抽出し、それら「状態」の変化（遷移）の条件や伴う動作を図式化したものを状態遷移図という。また、「状態」の変化に起因する「イベント」によりどのように状態が変化するかを表形式で表したものを状態遷移表という。状態遷移図や状態遷移表に基づいてテストケースを作成する技法を状態遷移テストという。

デシジョンテーブルとは、プログラムの仕様に基づいて、プログラムの入力を「条件」、実行結果を「動作」として定義するとともに、「各条件の成立状況」と「対応する動作の生起関係」を整理し、これらを表形式にまとめたものをいう。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。

③ブラックボックステストとグレーボックステスト

ブラックボックステストとは、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、暗箱の中の動作の良否を、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り、仕様に基づいた技法と同義である。

仕様が明示されない（暗黙の期待など）ものは、機能がプログラムのロジックに依存することがあることや、インターフェース仕様の確認のためには内部状態まで踏み込む必要があることから、場合によりプログラムロジックやメモリーの内容を参照するテストを行うことがある。このようなテストは、完全なブラックボックステストではないことからグレーボックステストと呼ばれる。したがって、選択肢イが適切である。

【解説】

CFD 技法とは、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。

CFD の作成の際にテスト対象の実装情報を用いて合理的にテストケース数を削減する、いわゆるグレーボックステストのアプローチが採られている。

CFD 技法の目的は、複雑な論理関係を持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ことである。

CFD 技法の効果は、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ことや、実装情報を利用することにより、テストケース数が増大することを防止できることが挙げられる。

なお、CFD は当初は Case Flow Diagram の略称とされていたが、現在は Cause Flow Diagram の略称である。

■問題：

セーフティに関する次の文章の（　　）の中にあてはまる語句として、もっとも適切な組合せを選べ。

セーフティの概念を理解するためには、システムによって人間の生命が損なわれたり、身体に害が及ぼされたり、社会に広範な悪影響が与えられることを指す（　①　）という概念と、危害を発生させてしまう原因を指す（　②　）という概念を理解する必要がある。また、セーフティにはハザードの発生を抑制する（　③　）と、システムにハザードが起こっても危害にいたらない性質である（　④　）がある。

【選択肢】

| | | | | |
|---|---|---|---|---|
| ア | ①ハザード（hazard） | ②危害（harm） | ③機能安全 | ④本質安全 |
| イ | ①ハザード（hazard） | ②危害（harm） | ③本質安全 | ④機能安全 |
| ウ | ①危害（harm） | ②ハザード（hazard） | ③本質安全 | ④機能安全 |
| エ | ①危害（harm） | ②ハザード（hazard） | ③機能安全 | ④本質安全 |

【正解】

ウ

【出題分野】

この問題は、SQuBOK（ソフトウェア品質知識体系ガイド）第 2 版樹形図の「第 1 章ソフトウェア品質の基本概念」の「1.1.5 セーフティ」からの出題である。

セーフティの概念の理解に必要となる危害（harm）とハザード（hazard）の概念の理解を確認し、セーフティを確保、高めるための考え方（性質）である機能安全と本質安全の理解を確認する問題である。

【選択肢の解説】

セーフティの概念を理解するためには、まず危害（harm）とハザード（hazard）という二つの概念を理解しておく必要がある。危害とは、システムによって人間の生命が損なわれたり、身体に害が及ぼされたり、社会に広範な悪影響が与えられることを指す。ハザードとは、危害を発生させてしまう原因である。

また、機能安全とは、システムにハザードが起こっても危害を回避できる性質を指し、本質安全とは、ハザードの発生を抑制する性質である。したがって、選択肢ウが適切である。

【解説】

安全(セーフティ)という言葉に関しては、事故が絶対に発生しないというニュアンスを安全と捉える傾向も見られるが、国際規格における安全の定義は、このニュアンスとは異なる。国際規格では、リスクが存在しないということを安全と捉えているわけではなく、対象となるリスクが安心して受け入れられる程度に抑えられた状態を安全と捉えている。安心して受け入れられる程度までリスクを抑制することが重要である。

国際規格で定義されているリスクには、受容不可能なリスク、受容不可能でないリスク、受容可能なリスクの 3 種類が存在する。受容可能なリスクとは、社会における現時点での評価に基づいた状況下で受け入れられるリスクをいう。つまり、現時点での評価に基づいて、受け入れられるリスクの程度が異なることを示しており、同じ程度のリスクでも、その対象の利便性や費用対効果、その時代や地域における安全に対する価値観などのバランスによって、受容可能な範囲が異なることを示している。

リスクを低減させるためには、本質安全な設計、付加的保護策、使用上の情報の周知の 3 つがある。このうち、本質安全な設計は、システム特性そのものによって危険源を直接低減する、もしくは危険源に関するリスクを適切に低減するものであり、もっとも優先して取り組むべきリスク低減策である。本質安全な設計で残ったリスクに対しては、付加的保護策を適用する。付加的保護策には、(1) 危険源の隔離もしくは回避、(2) 非常停止、エネルギー源の遮断もしくは放出、(3) 安全機能によって危険源を回避するという 3 つの方法がある。このうち、安全機能によって危険源を回避する考え方が、機能安全の概念である。特にコンピュータを用いたシステムでは、本質安全な設計の考え方やシステムそのものの特性によって本質的に安全を確保することは難しいことが多いため、機能安全でリスクを低減することが多い。また、使用上の情報の周知とは、製品の使用者に必要な情報を適切に周知することであり、マニュアルへの記載や警告表示などによりリスクを低減することなどが該当する。

セーフティを確保、高めるためには、リスク管理の考え方でリスクを評価し、システムあるいはソフトウェアを設計する必要がある。すなわち、ハザードの発生頻度を低くし、ハザードが発生した場合の危害の大きさを低減するような設計を行うことが重要である。

■問題：

プロセスモデルに関する次の文章の（　　）の中にあてはまる語句として、もっとも適切な組合せを選べ。

・ウォーターフォールモデルは、要求定義に始まり、分析、設計、実装、試験、運用に至る体系的な（　①　）型のアプローチである。
・（　②　）モデルとは、渦巻き状に開発を繰り返すことにより、少しずつソフトウェアの定義と実装を拡大、詳細化する開発方法である。
・（　③　）開発とは、多品種製品向けのソフトウェア開発技法である。
・（　④　）開発とは、すでに開発済みのソフトウェア資産に対して、製品価値を高めるための新しい機能の追加や、既存機能の改善を行う開発のことである。

【選択肢】

| | | | | |
|---|---|---|---|---|
| ア | ①並列 | ②反復型開発 | ③マルチプロダクト | ④回帰 |
| イ | ①逐次 | ②スパイラル | ③プロダクトライン | ④派生 |
| ウ | ①並列 | ②スパイラル | ③プロダクトライン | ④回帰 |
| エ | ①逐次 | ②反復型開発 | ③マルチプロダクト | ④派生 |

【正解】

イ

【出題分野】

この問題は、SQuBOK（ソフトウェア品質知識体系ガイド）第 2 版 樹形図の「第 2 章 ソフトウェア品質マネジメント」の「2.2 ライフサイクルプロセスのマネジメント」からの出題である。

この問題は、プロセスモデルについて基本的な考え方を確認する問題である。

【選択肢の解説】

ウォーターフォールモデルは、要求定義、分析、設計、実装、試験、運用の工程を順番に実施するソフトウェア開発モデルであり、その名称は工程が後戻りなく逐次進む様子を滝の水が流れ落ちるさまに例えたものである。

また、ソフトウェア開発の最終段階で致命的な問題が発生することを防止するために、計画・リスク分析、分析・設計、実装・テスト、展開という一連の流れを一つの区画として、渦巻き状に区画の実行を繰り返すモデルはスパイラルモデルと呼ばれている。

多品種の製品開発の場合、製品群を横断したソフトウェアの再利用がしやすいように共通のコア資産を構築して、そのコア資産から各製品を導出する方法が品質やコストなどの面で有効である。このような製品系列のためのソフトウェア開発技法はプロダクトライン開発と呼ばれている。

開発済みのソフトウェアへの機能の追加や既存機能の改善を行う場合、既存機能への影響を注意深く確認しながら開発作業を進める必要がある。このような開発は新たなソフトウェアを開発する新規開発とは異なることを明示するために派生開発と呼ばれている。したがって、選択肢イが適切である。

【解説】

プロセスモデルは、ライフサイクルモデルが定義する活動のうち開発部分について、開発作業の手順をプロセスとして構成し、開発工程と手順を抽象化したモデルとして表現したものである。

代表的なプロセスモデルは古典的モデルであるウォーターフォールモデルである。これ以外にもさまざまなプロセスモデルが提案され、さらに派生モデルも考案されている。

よく知られているプロセスモデルとしては、【選択肢の解説】で説明した、ウォーターフォールモデル、スパイラルモデル、プロダクトライン開発、派生開発があり、これら以外に反復型開発プロセス、プロトタイピング、アジャイル開発がある。

反復型開発プロセスは、小さい機能単位に動作可能なソフトウェアの開発を反復することにより、ソフトウェアを完成させる方法である。

プロトタイピングは、最終的に求めるソフトウェアのプロトタイプを早期段階で作成して確認することにより、要求定義や設計において合目的性を高めていく開発方法である。

アジャイル開発は、ウォーターフォールモデルに代表される逐次型のアプローチに対極する形のものであり、動くソフトウェアを継続的に提供することにより変化に素早く対応できるプロセスモデルとして脚光を浴びている。アジャイルプロセスモデルとして、XP（エクストリーム・プログラミング）、スクラム、FDD（機能駆動型開発）、クリスタルなどがある。