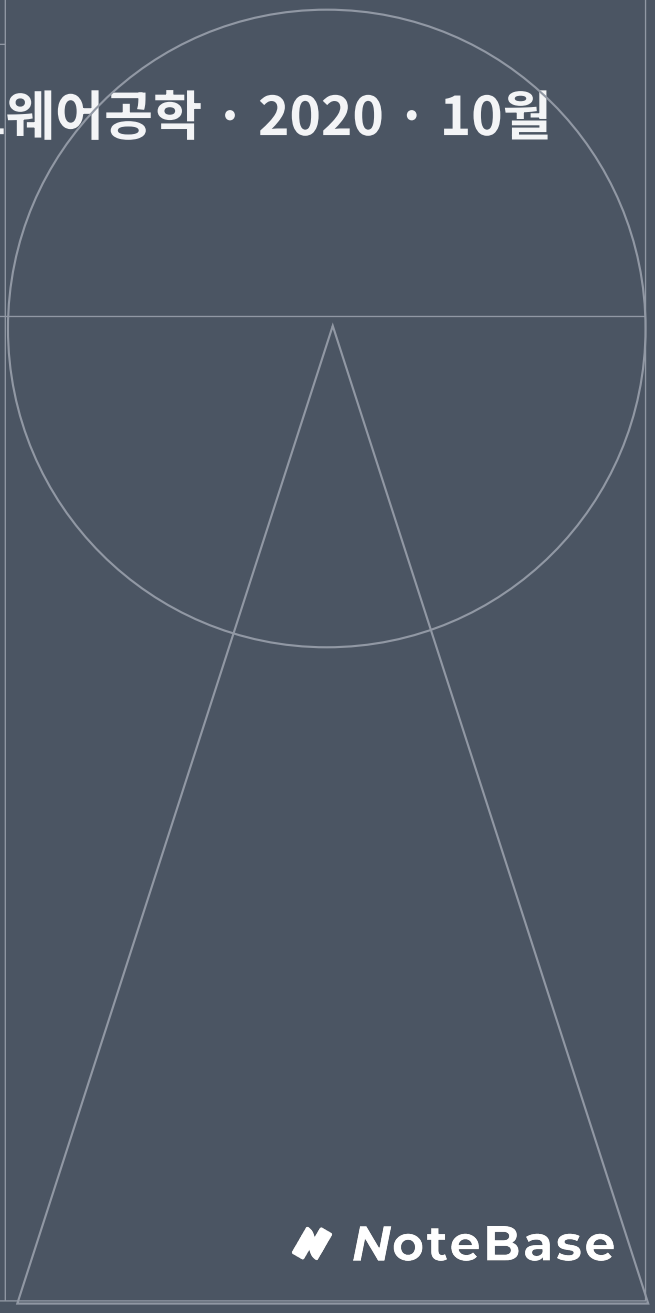
기출OX 해설집

공무원/서울시/7급 · 소프트웨어공학 · 2020 · 10월



NoteBase

기출OX 해설집

공무원/서울시/7급 · 소프트웨어공학 · 2020 · 10월

공무원/서울시/7급 · 소프트웨어공학 · 2020 · 10월

문 1. 폭포수 모델에 대한 설명으로 가장 옳지 않은 것은?

① 각 단계가 순차적으로 진행되며, 병행되거나 거슬러 반복 진행되는 경우가 거의 없다.

② 작업 단계가 단순하고, 다음 단계를 수행하기 위한 중간 산출물이 명확하다.

③ 문서작성에 비교적 시간이 적게 소요되어 코딩과 테스팅을 빠르게 수행할 수 있다.

④ 각 단계와 일정이 엄격하여 요구사항 변경을 수용하기 어렵다.

정답 3번

출제 포인트

이 문제는 폭포수 모델의 핵심 특징인 순차 진행, 산출물 명확성, 변경 대응의 경직성, 문서화 부담을 구별할 수 있는지를 묻는다.

핵심 개념

폭포수 모델은 요구분석, 설계, 구현, 시험, 유지보수가 단계적으로 내려가듯 진행되는 전통적 개발 모델이다. 각 단계의 결과물이 문서로 명확히 남지만, 앞단계로 되돌아가거나 요구사항을 바꾸는 데에는 불리하다.

선지별 해설

① O: 폭포수 모델은 각 단계가 순차적으로 진행되는 선형 모델이므로 단계 간 병행이나 반복이 거의 없다는 설명이 맞다.
② O: 단계가 비교적 분명하게 구분되고 각 단계의 문서와 산출물이 다음 단계의 입력이 되므로 중간 산출물이 명확하다는 설명이 맞다.
③ X: 틀린 부분은 "문서작성에 비교적 시간이 적게 소요되어 코딩과 테스팅을 빠르게 수행할 수 있다"는 표현이다. 폭포수 모델은 문서화 비중이 큰 편이어서 초기 분석과 설계 문서 작성에 시간이 들고, 그만큼 코딩과 테스팅을 바로 빠르게 시작하기 어렵다. 올바르게는 "문서화가 많아 관리와 추적에는 유리하지만 코딩과 테스팅 착수는 상대적으로 늦어질 수 있다"이다.
④ O: 폭포수 모델은 단계와 일정이 엄격하게 고정되는 경향이 있어 개발 도중 요구사항 변경을 수용하기 어렵다는 설명이 맞다.

NoteBase

오답 포인트

③은 폭포수 모델의 강한 문서화를 오히려 개발 속도를 높이는 장점처럼 뒤집어 놓은 선지이다. 폭포수는 문서화와 절차 통제에는 강하지만, 빠른 구현과 잦은 수정에는 불리하다는 점을 바꿔 읽으면 틀린다.

다시 나오면 이렇게 판단

선지가 순차 진행, 단계별 산출물 명확성, 변경 대응의 어려움을 말하면 폭포수 모델의 특징으로 본다. 반대로 반복 개발, 유연한 변경 수용, 빠른 시제품 제작을 말하면 폭포수보다 점증형이나 애자일 계열의 특징으로 판단한다.

암기 포인트

폭포수 모델은 순차 진행, 문서화 강함, 변경 수용 어려움으로 정리된다.

NoteBase

문 2. 응집도(cohesion)는 모듈 내의 구성요소들 간의 연관 정도를 나타낸다. <보기>는 각 구성요소가 응집된 형태에 따라 분류한 것인데, 이 중 응집도가 가장 낮은 것과 가장 높은 것을 순서대로 옳게 짝지은 것은?

<보기>
ㄱ. 우연적(coincidental) 응집도
ㄴ. 절차적(procedural) 응집도
ㄷ. 순차적(sequential) 응집도
ㄹ. 교환적(communicational) 응집도
ㅁ. 기능적(functional) 응집도
ㅂ. 논리적(logical) 응집도
ㅅ. 시간적(temporal) 응집도

① ㄱ - ㄷ
② ㄱ - ㅁ
③ ㄹ - ㅅ
④ ㅂ - ㄴ

정답 2번

출제 포인트

이 문제는 응집도의 강약 순서를 알고 최하위와 최상위를 구별하는지를 묻는 문제이다.

핵심 개념

우연적 응집도는 서로 관련 없는 요소가 한 모듈에 우연히 모여 있는 상태라서 가장 약하다.
기능적 응집도는 모듈 전체가 하나의 명확한 기능만 수행하도록 결합된 상태라서 가장 강하다.
응집도는 일반적으로 우연적 < 논리적 < 시간적 < 절차적 < 교환적 < 순차적 < 기능적 순으로 강해진다.

NoteBase

선지별 해설

① X: ㄱ은 우연적 응집도로 최하위이므로 맞지만, ㄷ은 순차적 응집도라서 최상위가 아니다. 틀린 부분은 '가장 높은 것 = ㄷ'이라는 대응이다. 올바르게는 '가장 높은 것 = ㅁ 기능적 응집도'이다.
② O: ㄱ은 우연적 응집도로 가장 낮고, ㅁ은 기능적 응집도로 가장 높다. 최하위와 최상위를 순서대로 정확히 짝지은 선택지이다.
③ X: ㄹ은 교환적 응집도로 동일한 데이터나 출력에 관련된 처리들이 묶인 형태이므로 최하위가 아니다. ㅅ은 시간적 응집도로 같은 시점에 수행된다는 이유로 묶인 형태이므로 최상위도 아니다. 올바르게는 '가장 낮은 것 = ㄱ 우연적 응집도, 가장 높은 것 = ㅁ 기능적 응집도'이다.
④ X: ㅂ은 논리적 응집도로 비슷한 종류의 기능을 한 모듈에 넣은 형태라서 최하위가 아니고, ㄴ은 절차적 응집도로 수행 순서에 따라 묶인 형태라서 최상위가 아니다. 올바르게는 '가장 낮은 것 = ㄱ 우연적 응집도, 가장 높은 것 = ㅁ 기능적 응집도'이다.

오답 포인트

순차적 응집도와 기능적 응집도를 모두 높은 응집도로 기억해 두고 최상위를 혼동하기 쉽다. 순차적은 앞 단계의 출력이 다음 단계 입력으로 이어지는 관계이고, 기능적은 모듈 전체가 오직 하나의 기능만 수행한다는 점에서 더 강하다.

다시 나오면 이렇게 판단

응집도 문제는 먼저 양 끝을 잡으면 된다. 관련이 거의 없이 모아 놓았으면 우연적, 하나의 목적 기능만 수행하면 기능적이다. 그 다음 중간 단계는 논리적, 시간적, 절차적, 교환적, 순차적 순으로 정리하면 된다.

암기 포인트

응집도는 우논시절교순기 순으로 강해지고, 최저는 우연적, 최고는 기능적이다.

NoteBase

문 3. <보기>의 프로젝트 산정 기법으로 가장 옳은 것은?

<보기>
유사한 프로젝트 경험을 가진 전문가 집단을 구성하고, 이 전문가들에게 규모, 공수, 비용의 산정 의견을 구하되, 의견 일치가 이루어지지 않을 경우 의견의 근거를 익명으로 집단 내에 배포하고 자신들의 산정을 수정할 수 있도록 한다. 이 과정을 필요한 만큼 반복함으로써 의견 일치를 이루어 최종적인 산정값을 구한다.

① COCOMO
② 기능 점수
③ WBS
④ 델파이 기법

정답 4번

출제 포인트

이 문제는 전문가 판단 기반 프로젝트 산정 기법인 델파이 기법의 절차를 다른 산정 관련 개념과 구별할 수 있는지를 묻는다.

핵심 개념

델파이 기법은 유사 프로젝트 경험이 있는 전문가들의 추정치를 익명으로 반복 수렴시켜 합의된 산정값을 얻는 기법이다.
COCOMO는 소프트웨어 규모를 바탕으로 공수와 비용을 계산하는 모형이다.
기능 점수는 사용자 관점의 기능 규모를 계량하는 기법이다.
WBS는 프로젝트 작업을 계층적으로 분해한 구조이다.

NoteBase

선지별 해설

1번: X. COCOMO는 KLOC 등 규모 지표를 입력으로 노력과 비용을 산정하는 알고리즘 모형이다. 보기처럼 전문가 집단이 익명으로 의견을 교환하며 합의하는 절차와는 다르다. 틀린 표현은 COCOMO를 전문가 합의 기법으로 본 점이다. 올바르게는 규모 지표와 산정식을 이용하는 비용 산정 모형이다.
2번: X. 기능 점수는 입력, 출력, 질의, 내부논리파일, 외부인터페이스파일 등을 기준으로 기능 규모를 점수화하는 기법이다. 보기의 핵심은 익명 의견 수렴과 반복적 재산정이므로 기능 점수와 맞지 않는다. 틀린 표현은 기능 점수를 전문가 의견 수렴 기법으로 본 점이다. 올바르게는 사용자 기능을 계량해 규모를 산정하는 기법이다.
3번: X. WBS는 프로젝트 범위를 작업 단위로 계층 분해한 구조이다. 일정 수립과 책임 배분의 기준은 되지만, 익명 피드백을 반복하여 산정값을 합의하는 절차는 아니다. 틀린 표현은 WBS를 산정 합의 기법으로 본 점이다. 올바르게는 작업 범위를 구조화하는 분해 체계이다.
4번: O. 델파이 기법은 전문가 집단의 의견을 익명으로 수집하고, 그 근거를 공유한 뒤 재산정을 반복하여 최종 합의치를 얻는 방식이다. 보기의 익명 배포, 산정 수정, 반복, 의견 일치가 모두 델파이 기법의 핵심 특징이다.

오답 포인트

산정이라는 단어만 보고 COCOMO나 기능 점수로 가면 틀리기 쉽다. 이 문항은 계산식이나 규모 측정이 아니라 전문가 의견을 어떤 방식으로 수렴하는지를 묻는다.

다시 나오면 이렇게 판단

전문가 집단, 익명, 반복 피드백, 합의 도출이라는 단서가 보이면 델파이 기법으로 판단한다. 규모 산정식이면 COCOMO, 기능 수 계량이면 기능 점수, 작업 분해이면 WBS로 구분한다.

암기 포인트

델파이=전문가 익명 반복 합의, COCOMO=비용 산정 모형, 기능 점수=기능 규모 측정, WBS=작업 분해 구조이다.

NoteBase

문 4. UML 다이어그램 중 행위(behavior) 다이어그램에 해당하지 않는 것은?

① 활동(activity) 다이어그램

② 상태(state) 다이어그램

③ 유스케이스(use case) 다이어그램

④ 컴포넌트(component) 다이어그램

정답 4번

출제 포인트

이 문제는 UML 다이어그램을 행위 다이어그램과 구조 다이어그램으로 구분할 수 있는지를 묻는 문제이다.

핵심 개념

행위 다이어그램은 시스템의 동작, 흐름, 상태 변화, 상호작용을 표현하는 다이어그램이다. 활동 다이어그램은 작업 흐름을, 상태 다이어그램은 상태 변화를, 유스케이스 다이어그램은 액터와 시스템 기능의 관계를 나타낸다. 컴포넌트 다이어그램은 소프트웨어 구성요소와 그 의존관계를 표현하는 구조 다이어그램이다.

NoteBase

선지별 해설

① X
활동(activity) 다이어그램은 업무 처리 순서와 제어 흐름을 표현하는 UML 행위 다이어그램이므로 '행위 다이어그램에 해당하지 않는 것'으로 고를 수 없다.
올바르게는 활동 다이어그램은 행위 다이어그램에 해당한다.

② X
상태(state) 다이어그램은 객체나 시스템이 이벤트에 따라 어떤 상태로 바뀌는지를 나타내므로 행위 다이어그램이다.
올바르게는 상태 다이어그램은 행위 다이어그램에 해당한다.

③ X
유스케이스(use case) 다이어그램은 액터와 유스케이스의 관계를 통해 시스템이 제공하는 기능과 사용 시나리오를 나타내므로 행위 다이어그램으로 분류한다.
올바르게는 유스케이스 다이어그램은 행위 다이어그램에 해당한다.

④ O
컴포넌트(component) 다이어그램은 컴포넌트, 인터페이스, 의존관계처럼 시스템의 정적 구성을 표현하므로 구조 다이어그램이다.
따라서 행위 다이어그램에 해당하지 않는 것으로 판단하는 것이 맞다.

오답 포인트

유스케이스 다이어그램을 단순한 기능 목록처럼 보고 구조 쪽으로 착각하기 쉽다. 그러나 UML 분류에서는 사용자와 시스템의 상호작용을 다루므로 행위 다이어그램이다. 반대로 컴포넌트 다이어그램은 실행 흐름이 아니라 구성요소의 정적 관계를 보여 준다.

다시 나오면 이렇게 판단

다이어그램이 흐름, 상태 변화, 상호작용처럼 시스템의 동작을 설명하면 행위 다이어그램으로 본다. 클래스, 컴포넌트, 배치처럼 정적 구성과 연결 관계를 보여 주면 구조 다이어그램으로 분류한다.

암기 포인트

흐름·상태·유스케이스는 행위, 컴포넌트는 구조이다.

NoteBase

문 5. <보기>의 항목들이 해당되는 소프트웨어 유지보수의 종류로 가장 옳은 것은?

<보기>
• 리눅스(Linux)에서 동작하는 프로그램을 다른 운영체제에서도 동작하도록 함
• 인터넷 익스플로러(Internet Explorer)에 최적화된 웹 프로그램을 크롬(Chrome)에서도 동작하도록 함
• 특정 버전의 라이브러리(Library)를 사용하여 개발된 프로그램을 좀 더 최신 버전의 라이브러리도 사용이 가능하도록 개선함

① 수정 유지보수(corrective maintenance)
② 적응 유지보수(adaptive maintenance)
③ 완전 유지보수(perfective maintenance)
④ 예방 유지보수(preventive maintenance)

정답 2번

출제 포인트

이 문제는 수정 유지보수, 적응 유지보수, 완전 유지보수, 예방 유지보수의 구분 기준을 묻는 문제이다.

핵심 개념

수정 유지보수는 이미 발생한 오류를 찾아 고치는 작업이다.
적응 유지보수는 운영체제, 브라우저, 라이브러리, 하드웨어처럼 소프트웨어를 둘러싼 실행 환경의 변화에 맞추는 작업이다.
완전 유지보수는 기능, 성능, 사용성처럼 소프트웨어의 품질이나 만족도를 더 높이는 작업이다.
예방 유지보수는 잠재 결함과 구조적 취약점을 미리 줄여 미래의 장애 가능성을 낮추는 작업이다.

NoteBase

선지별 해설

① X. 보기의 사례들은 고장 난 기능을 복구한 것이 아니라 다른 운영체제, 브라우저, 라이브러리 환경에서도 동작하게 바꾼 것이다. 틀린 표현은 "수정 유지보수"이다. 올바르게는 "환경 변화에 대응하는 적응 유지보수"이다.
② O. 리눅스용 프로그램을 다른 운영체제에서도 실행되게 하거나, 인터넷 익스플로러에 맞춘 웹 프로그램을 크롬에서도 동작하게 하거나, 특정 라이브러리 버전에 묶인 프로그램을 최신 라이브러리에서도 쓰이게 하는 일은 모두 실행 환경 변화에 맞춘 적응 유지보수이다.
③ X. 완전 유지보수는 성능 향상, 기능 추가, 사용성 개선처럼 소프트웨어를 더 좋게 만드는 작업이 중심이다. 보기의 핵심은 품질 고도화가 아니라 환경 적합성 확보이다. 틀린 표현은 "완전 유지보수"이다. 올바르게는 "운영 환경 변화에 맞추는 적응 유지보수"이다.
④ X. 예방 유지보수는 미래의 장애를 막기 위해 코드 구조를 정리하거나 잠재 결함을 사전에 제거하는 작업이다. 보기는 미래 장애 예방보다 운영 환경 변화에 대한 대응이 중심이다. 틀린 표현은 "예방 유지보수"이다. 올바르게는 "환경 변화 수용을 위한 적응 유지보수"이다.

오답 포인트

라이브러리 버전 업그레이드나 브라우저 호환성 개선을 보면 막연히 성능 개선으로 생각해 완전 유지보수로 오해하기 쉽다. 그러나 기준은 결과가 아니라 원인이다. 외부 환경이 바뀌어서 맞춰 주는 작업이면 적응 유지보수이다.

다시 나오면 이렇게 판단

먼저 변경 이유를 본다. 오류를 고치면 수정 유지보수, 운영체제나 브라우저 같은 외부 환경 변화에 맞추면 적응 유지보수, 기능이나 품질을 더 좋게 만들면 완전 유지보수, 장래 장애를 미리 줄이면 예방 유지보수로 판단된다.

암기 포인트

고장 나서 고치면 수정, 환경 바뀌면 적응, 더 좋게 만들면 완전, 미리 막으면 예방이다.

NoteBase

문 6. <보기>에서 비기능 요구사항(non-functional requirement)을 모두 고른 것은?

<보기>
ㄱ. 시스템의 반응시간(response time)
ㄴ. 시스템이 제공해야 하는 가용성(availability)
ㄷ. 시스템이 동작해야 하는 운영체제
ㄹ. 개발에 사용될 프로그래밍 언어

① ㄱ, ㄴ
② ㄷ, ㄹ
③ ㄱ, ㄴ, ㄹ
④ ㄱ, ㄴ, ㄷ, ㄹ

정답 4번

출제 포인트

이 문제는 기능 요구사항과 비기능 요구사항을 구분하면서, 성능·가용성뿐 아니라 운영 환경과 개발 제약도 비기능 요구사항에 포함되는지를 묻는 문제이다.

핵심 개념

기능 요구사항은 시스템이 무엇을 해야 하는지를 정의한 것이다.
비기능 요구사항은 시스템이 어떤 품질 수준으로 동작해야 하는지, 어떤 환경과 제약 아래에서 개발·운영되어야 하는지를 정의한 것이다.
반응시간과 가용성은 성능·신뢰성 같은 품질 요구사항이고, 운영체제와 프로그래밍 언어는 구현 및 운영 환경에 대한 제약사항이므로 모두 비기능 요구사항에 속한다.

NoteBase

기출OX 해설집 · 공무원/서울시/7급 · 소프트웨어공학 · 2020 · 10월

선지별 해설

① X
- ①은 ㄱ, ㄴ만 골라서 ㄷ, ㄹ을 제외한 점이 틀리다.
- ㄱ의 반응시간은 성능 요구사항이므로 비기능 요구사항이다.
- ㄴ의 가용성은 시스템이 얼마나 지속적으로 서비스를 제공할 수 있는지를 나타내는 품질 요구사항이므로 비기능 요구사항이다.
- ㄷ의 운영체제는 시스템이 동작해야 하는 환경 제약이므로 비기능 요구사항이다.
- ㄹ의 프로그래밍 언어도 개발에 사용될 기술 제약이므로 비기능 요구사항이다.
- 틀린 표현은 'ㄷ, ㄹ 제외'이다. 올바르게는 'ㄱ, ㄴ, ㄷ, ㄹ 모두 포함'이다.

② X
- ②는 ㄷ, ㄹ만 골라서 성능과 가용성을 빠뜨린 점이 틀리다.
- ㄷ의 운영체제와 ㄹ의 프로그래밍 언어는 환경 및 개발 제약이라는 점에서 비기능 요구사항이 맞다.
- 그러나 ㄱ의 반응시간과 ㄴ의 가용성 역시 시스템의 품질 특성을 규정하므로 비기능 요구사항이다.
- 틀린 표현은 'ㄱ, ㄴ 제외'이다. 올바르게는 '성능·가용성도 비기능 요구사항에 포함'이다.

③ X
- ③은 ㄱ, ㄴ, ㄹ은 포함했지만 ㄷ을 제외한 점이 틀리다.
- 운영체제는 시스템이 어떤 플랫폼에서 동작해야 하는지를 제한하는 요구이므로 기능이 아니라 비기능 요구사항이다.
- 따라서 ㄷ을 뺀 구성이 성립하지 않는다.
- 틀린 표현은 'ㄷ 제외'이다. 올바르게는 '운영체제도 비기능 요구사항에 포함'이다.

④ O
- ㄱ의 반응시간, ㄴ의 가용성은 대표적인 품질 속성이다.
- ㄷ의 운영체제, ㄹ의 프로그래밍 언어는 시스템의 구현·운영 환경과 기술 제약을 나타낸다.
- 네 항목 모두 시스템의 기능 자체보다는 품질, 환경, 제약을 설명하므로 비기능 요구사항이다.

오답 포인트

수험생은 비기능 요구사항을 성능·가용성 같은 품질 속성으로만 좁게 외우는 경우가 많다. 이 때문에 ㄷ의 '운영체제'와 ㄹ의 '프로그래밍 언어'를 기능 요구사항이나 단순 설계 사항으로 오인하기 쉽다. 이 문항은 제약사항도 비기능 요구사항에 포함된다는 점을 노린 문제이다.

다시 나오면 이렇게 판단

선지가 시스템이 제공할 서비스 자체를 말하면 기능 요구사항으로 본다. 반대로 성능, 신뢰성, 가용성, 사용 환경, 개발 언어, 운영 플랫폼처럼 서비스가 구현·운영되는 품질이나 제약을 말하면 비기능 요구사항으로 판단한다.

NoteBase

암기 포인트

무엇을 한다는 것은 기능, 얼마나 잘 하고 어떤 환경에서 해야 하는지는 비기능이다.

NoteBase

문 7. V 모델은 폭포수 모델에 품질 보증을 위한 테스팅 과정이 보완된 모델이다. 이 모델에서 테스트가 수행되는 순서대로 바르게 나열한 것은?

① 단위 테스트 - 통합 테스트 - 시스템 테스트 - 인수 테스트

② 단위 테스트 - 시스템 테스트 - 통합 테스트 - 인수 테스트

③ 단위 테스트 - 통합 테스트 - 인수 테스트 - 시스템 테스트

④ 단위 테스트 - 인수 테스트 - 통합 테스트 - 시스템 테스트

정답 1번

출제 포인트

이 문제는 V 모델에서 테스트가 단위 수준에서 전체 사용자 수용 수준으로 확대되는 순서를 정확히 구분할 수 있는지를 묻는다.

올바른 순서

1. 단위 테스트 - 개별 모듈이나 함수가 설계대로 동작하는지 확인하는 단계이다.
2. 통합 테스트 - 단위 테스트를 통과한 모듈들을 결합하여 인터페이스와 상호작용을 검증하는 단계이다.
3. 시스템 테스트 - 통합된 전체 시스템이 기능 및 비기능 요구사항을 만족하는지 확인하는 단계이다.
4. 인수 테스트 - 사용자의 요구와 업무 관점에서 최종 수용 가능 여부를 확인하는 단계이다.

왜 이 순서: 통합 테스트는 단위가 결합되어야 가능하고, 시스템 테스트는 통합된 전체가 준비되어야 가능하며, 인수 테스트는 시스템 수준 검증이 끝난 뒤 최종 승인 단계로 수행된다.

선지별 해설

① O: 단위 테스트 → 통합 테스트 → 시스템 테스트 → 인수 테스트의 순서가 맞다. V 모델은 작은 구성요소 검증에서 시작해 점차 전체 시스템 검증과 사용자 수용 검증으로 올라간다.

② X: 시스템 테스트를 통합 테스트보다 먼저 둔 점이 틀리다. 전체 시스템 테스트는 모듈 간 통합이 끝난 뒤에야 가능하다. 틀린 표현은 "시스템 테스트 - 통합 테스트"의 순서이고, 올바르게는 "통합 테스트 - 시스템 테스트"이다.

③ X: 인수 테스트를 시스템 테스트보다 먼저 둔 점이 틀리다. 인수 테스트는 완성된 시스템을 사용자 관점에서 최종 확인하는 단계이므로 시스템 테스트 다음에 와야 한다. 틀린 표현은 "인수 테스트 - 시스템 테스트"의 순서이고, 올바르게는 "시스템 테스트 - 인수 테스트"이다.

④ X: 인수 테스트를 단위 테스트 직후에 둔 점이 틀리다. 인수 테스트는 개별 모듈 단계가 아니라 통합과 시스템 검증이 끝난 뒤 수행된다. 틀린 표현은 "단위 테스트 - 인수 테스트"의 순서이고, 올바르게는 "단위 테스트 - 통합 테스트 - 시스템 테스트 - 인수 테스트"이다.

NoteBase

오답 포인트

통합 테스트와 시스템 테스트를 비슷한 단계로 보고 서로 바꾸거나, 인수 테스트를 단순한 최종 확인으로만 이해해 시스템 테스트보다 앞에 두면 틀리기 쉽다.

다시 나오면 이렇게 판단

테스트 순서 문제는 개별 모듈 확인인지, 모듈 결합 확인인지, 전체 시스템 확인인지, 사용자 수용 확인인지를 차례로 대입하면 판단할 수 있다.

암기 포인트

단위에서 시작해 통합으로 묶고, 시스템으로 넓힌 뒤 인수로 마무리하는 흐름이다.

NoteBase

문 8. <보기>는 위험 관리 활동을 구성하는 작업들이다. 이들을 실행 순서대로 바르게 나열한 것은?

<보기>
ㄱ. 위험 식별(risk identification)
ㄴ. 위험 관찰(risk monitoring)
ㄷ. 위험 분석(risk analysis)
ㄹ. 위험 계획(risk planning)

① ㄱ - ㄹ - ㄷ - ㄴ
② ㄱ - ㄷ - ㄹ - ㄴ
③ ㄴ - ㄱ - ㄹ - ㄷ
④ ㄹ - ㄴ - ㄱ - ㄷ

정답 2번

출제 포인트

이 문제는 소프트웨어 위험 관리 활동의 표준 수행 순서를 묻는 문제이다.

올바른 순서

1. 위험 식별(ㄱ) - 프로젝트에 영향을 줄 수 있는 위험 요소를 찾아 목록화하는 단계이다.
2. 위험 분석(ㄷ) - 식별한 위험의 발생 가능성과 영향도를 평가하는 단계이다.
3. 위험 계획(ㄹ) - 분석 결과를 바탕으로 대응 전략과 회피·완화 방안을 세우는 단계이다.
4. 위험 관찰(ㄴ) - 계획한 대응을 추적하면서 위험 상태 변화를 지속적으로 점검하는 단계이다.

왜 이 순서: 분석은 식별된 위험이 있어야 가능하고, 계획은 분석 결과가 있어야 세울 수 있으며, 관찰은 실행된 계획과 위험 상태를 계속 추적하는 후속 관리 단계이기 때문이다.

NoteBase

선지별 해설

① X
- 틀린 부분: 'ㄱ-ㄹ-ㄷ-ㄴ'에서 위험 계획을 위험 분석보다 앞에 둔 점이 틀렸다.
- 이유: 대응 계획은 위험의 우선순위와 영향도를 분석한 뒤에 세운다.
- 올바르게는: ㄱ-ㄷ-ㄹ-ㄴ이다.

② O
- 이유: 위험 식별 후 위험 분석으로 심각도를 판단하고, 그 결과로 대응 계획을 수립한 뒤 지속적으로 위험을 관찰하는 순서가 맞다.

③ X
- 틀린 부분: 'ㄴ-ㄱ-ㄹ-ㄷ'에서 위험 관찰을 가장 먼저 둔 점이 틀렸다.
- 이유: 관찰은 이미 식별되고 계획된 위험을 추적하는 활동이므로 시작 단계가 될 수 없다.
- 올바르게는: ㄱ-ㄷ-ㄹ-ㄴ이다.

④ X
- 틀린 부분: 'ㄹ-ㄴ-ㄱ-ㄷ'에서 위험 계획과 위험 관찰을 위험 식별보다 앞에 둔 점이 틀렸다.
- 이유: 계획은 분석 이전에 세울 수 없고, 관찰은 식별조차 되지 않은 위험을 대상으로 수행할 수 없다.
- 올바르게는: ㄱ-ㄷ-ㄹ-ㄴ이다.

오답 포인트

위험 계획과 위험 관찰을 모두 관리 활동으로만 묶어 중간 순서를 혼동하기 쉽다. 그러나 계획은 분석 결과를 입력으로 받고, 관찰은 대응 이후 계속 이어지는 추적 단계라는 점이 핵심이다.

다시 나오면 이렇게 판단

위험 관리 순서 문제는 '찾는다 -> 평가한다 -> 대응을 정한다 -> 계속 추적한다'의 흐름으로 판단하면 된다.

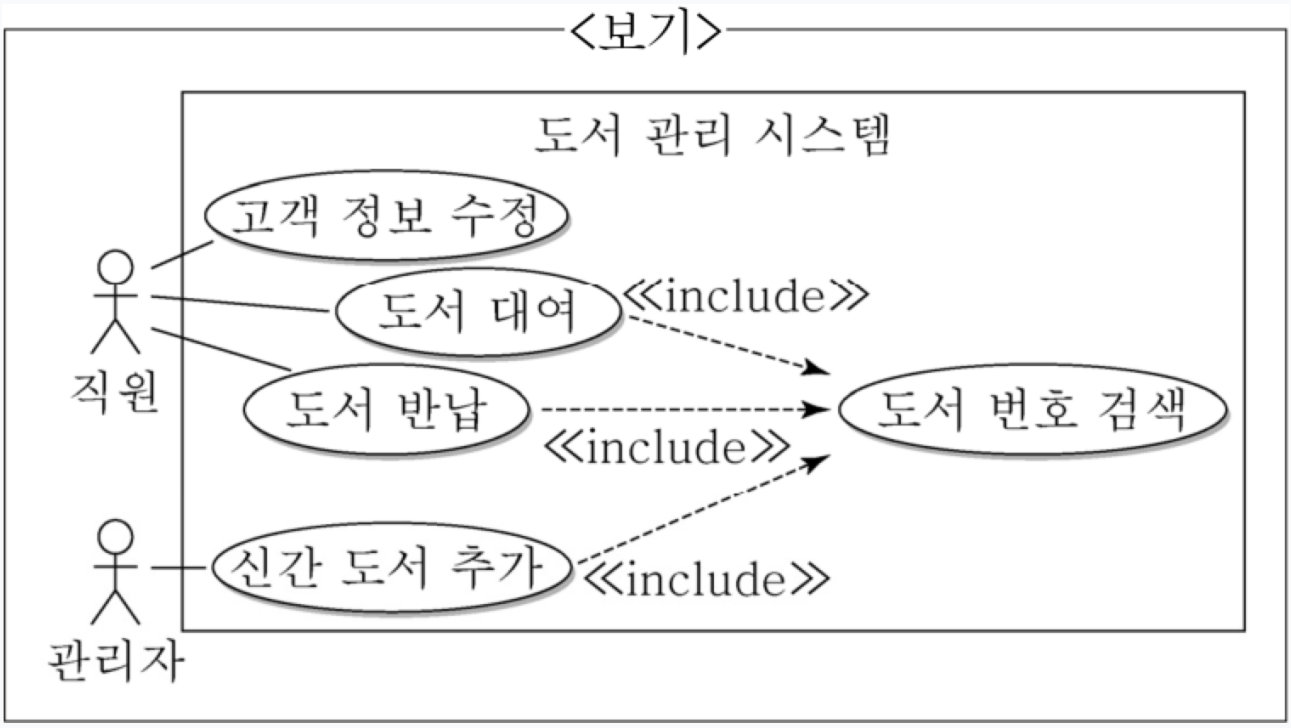
암기 포인트

위험 관리는 식별 -> 분석 -> 계획 -> 관찰의 흐름이다.

NoteBase

문 9. <보기>는 도서 관리 시스템을 표현한 유스케이스(use case) 다이어그램이다. 이에 대한 설명으로 가장 옳은 것은?

<보기>



① 모든 액터(actor)는 도서 관리 시스템의 모든 기능을 사용할 수 있다.

② 도서 관리 시스템에는 두 액터(직원, 관리자) 이외에 추가적인 사용자가 있다.

③ '도서 대여', '도서 반납', '신간 도서 추가'의 유스케이스에서 중복된 부분이 '도서 번호 검색' 유스케이스로 추출되었다.

④ '도서 번호 검색'은 액터와 상호작용하지 않는다.

정답 3번

출제 포인트

이 문제는 유스케이스 다이어그램에서 액터와 유스케이스의 연관, 시스템 경계, `<<include>>` 관계의 의미를 정확히 읽는지를 묻는 문제이다.

NoteBase

선지별 해설

① X. 액터는 자신과 연관선으로 연결된 유스케이스만 수행할 수 있다. 그림에서 직원은 '고객 정보 수정', '도서 대여', '도서 반납'과 연결되고 관리자는 '신간 도서 추가'와만 연결된다. 틀린 표현은 '모든 액터'와 '모든 기능'이며, 올바르게는 각 액터는 자신에게 연결된 기능만 사용할 수 있다.
② X. 다이어그램에 표현된 외부 액터는 직원과 관리자 두 명뿐이다. 시스템 경계 밖에 다른 액터 기호나 명칭이 없다. 틀린 표현은 '추가적인 사용자가 있다'이며, 올바르게는 추가 액터가 표현되어 있지 않다.
③ O. '도서 대여', '도서 반납', '신간 도서 추가'에서 공통으로 필요한 기능이 '도서 번호 검색'으로 분리되어 있고, 각 유스케이스에서 `<<include>>` 관계로 이를 포함한다. 이는 여러 유스케이스의 공통 동작을 별도 유스케이스로 추출한 형태이다.
④ X. '도서 번호 검색'은 액터와 직접 연관선이 없을 뿐, 액터가 수행하는 상위 유스케이스에 포함되어 실행된다. 틀린 표현은 '상호작용하지 않는다'이며, 올바르게는 직접 연결은 없지만 포함 관계를 통해 간접적으로 사용된다.

오답 포인트

액터와 직접 연결선이 없다는 이유만으로 해당 유스케이스가 완전히 독립적이라고 보면 ④에 흔들리기 쉽다. 또한 `<<include>>`를 상속이나 액터 추가 의미로 오해하면 ②, ④를 잘못 고르기 쉽다.

다시 나오면 이렇게 판단

유스케이스 다이어그램은 먼저 액터와 직접 연결된 기능을 확인하고, 그다음 점선 화살표와 `<<include>>` 표기가 공통 기능의 포함 관계인지 본다. 직접 연관과 포함 관계를 구분하면 정답을 안정적으로 고를 수 있다.

암기 포인트

유스케이스 다이어그램에서 `<<include>>`는 여러 유스케이스가 공통으로 사용하는 기능의 분리이다.

NoteBase

문 10. 재공학(reengineering)에 대한 설명으로 가장 옳지 않은 것은?

① 소프트웨어의 품질 등을 개선하기 위해 시스템 또는 컴포넌트를 재구조화하는 과정을 의미한다.

② 역공학(reverse engineering)과 새로운 요구에 맞추어 구현하는 정공학(forward engineering)을 포함하는 작업이다.

③ 재공학의 작업과정에서는 개선이 필요한 위치 파악을 가장 먼저 수행한다.

④ 현재 시스템의 아키텍처를 수정하지 않는 선에서 새로운 요구사항을 반영하도록 한다.

정답 4번

출제 포인트

이 문제는 재공학의 목적과 작업 흐름을 이해하고, 역공학·정공학·유지보수를 서로 구별할 수 있는지를 묻는 문제이다.

핵심 개념

재공학은 기존 소프트웨어의 외부 기능은 가능한 한 유지하면서 내부 구조, 코드, 데이터, 문서를 다시 정비해 품질과 유지보수성을 높이는 활동이다.
역공학은 기존 시스템을 분석하여 설계나 명세를 추출하는 활동이다.
정공학은 분석 결과를 바탕으로 더 나은 구조로 다시 구현하는 활동이다.

NoteBase

기출OX 해설집 · 공무원/서울시/7급 · 소프트웨어공학 · 2020 · 10월

선지별 해설

① O
소프트웨어 품질 향상을 위해 시스템이나 컴포넌트를 재구조화한다는 설명은 재공학의 핵심과 맞는다. 재공학은 성능, 유지보수성, 이해 가능성 같은 품질 속성을 높이기 위해 내부 구조를 손보는 활동이다.

② O
재공학은 기존 시스템을 이해하는 역공학과, 필요한 경우 이를 바탕으로 다시 구성하는 정공학을 포함한다. 즉 기존 것을 해석하고 더 나은 형태로 재구성하는 흐름으로 이해하면 된다.

③ O
재공학은 먼저 어떤 시스템이나 모듈이 개선 대상인지 파악하는 단계가 선행된다. 개선이 필요한 위치를 식별해야 이후의 역공학, 재구조화, 재구현 범위를 정할 수 있다.

④ X
틀린 부분은 "현재 시스템의 아키텍처를 수정하지 않는 선에서 새로운 요구사항을 반영"이라는 표현이다.
재공학의 주목적은 새로운 요구사항 반영 자체가 아니라 기존 시스템의 구조와 품질을 개선하는 데 있다.
실제로는 필요하면 아키텍처나 설계를 재구성할 수 있으며, 새로운 요구사항 반영 중심의 작업은 유지보수나 재개발에 더 가깝다.

오답 포인트

재공학을 기존 시스템에 새 기능을 덧붙이는 유지보수와 같은 뜻으로 보면 ④에 끌리기 쉽다. 이 문항은 "기존 기능 유지 + 내부 구조 개선"이 재공학의 중심이라는 점을 묻는다.

다시 나오면 이렇게 판단

선지에 기존 시스템을 분석한다는 말이 나오면 역공학, 더 나은 구조로 다시 구현한다는 말이 나오면 정공학, 기존 시스템의 품질과 유지보수성을 높이기 위해 내부를 손본다는 말이 나오면 재공학으로 판단한다. 반대로 새 요구사항 반영 자체를 중심에 두면 유지보수나 재개발 쪽 설명일 가능성이 크다.

암기 포인트

재공학은 기존 시스템 개선, 역공학은 기존 시스템 분석, 정공학은 개선 결과의 재구현이다.

NoteBase

문 11. <보기>의 설명에 해당하는 GoF (Gang of Four) 설계 패턴은?

<보기>
어떤 객체에 책임(responsibility)을 동적으로 추가할 수 있도록 한다. 기능 확장이 필요할 때 서브 클래싱(subclassing) 대신 쓸 수 있는 유연한 대안을 제공한다.

① 장식자(decorator) 패턴
② 싱글톤(singleton) 패턴
③ 반복자(iterator) 패턴
④ 상태(state) 패턴

정답 1번

출제 포인트

이 문제는 객체에 기능과 책임을 동적으로 덧붙이는 장식자 패턴의 정의를 다른 GoF 패턴과 구별할 수 있는지를 묻는 문제이다.

핵심 개념

장식자 패턴은 기존 객체를 감싸서 같은 인터페이스를 유지한 채 기능과 책임을 동적으로 추가하는 패턴이다. 결정 차이축은 상속 대신 합성으로 확장하느냐이다. 키워드는 동적 기능 추가, 래핑, 상속의 대안이다.
싱글톤 패턴은 어떤 클래스의 인스턴스를 하나만 생성하도록 보장하고 그 인스턴스에 전역 접근점을 제공하는 패턴이다. 키워드는 유일 인스턴스이다.
반복자 패턴은 집합 객체의 내부 표현을 노출하지 않고 원소를 순차적으로 접근하게 하는 패턴이다. 키워드는 순회이다.
상태 패턴은 객체의 내부 상태가 바뀔 때 행위가 달라지도록 상태를 객체로 분리하는 패턴이다. 키워드는 상태에 따른 행위 변화이다.

NoteBase

선지별 해설

① O
- <보기>의 '책임을 동적으로 추가'와 '서브 클래싱 대신 유연한 대안'은 장식자 패턴의 대표 정의와 정확히 일치한다.
- 장식자 패턴은 객체를 감싸는 방식으로 기능을 확장하므로 상속 계층을 늘리지 않고도 책임을 조합할 수 있다.

② X
- 틀린 부분은 싱글톤 패턴이라고 본 점이다.
- 싱글톤은 인스턴스 수를 하나로 제한하는 패턴이지, 객체에 책임을 동적으로 추가하는 패턴이 아니다.
- 올바르게는 '하나의 인스턴스만 유지'가 싱글톤의 핵심이다.

③ X
- 틀린 부분은 반복자 패턴이라고 본 점이다.
- 반복자는 컬렉션의 내부 구조를 드러내지 않고 순차 접근을 제공하는 패턴이지, 기능 확장을 위한 패턴이 아니다.
- 올바르게는 '집합 객체 순회 방법 제공'이 반복자 패턴의 핵심이다.

④ X
- 틀린 부분은 상태 패턴이라고 본 점이다.
- 상태 패턴은 내부 상태 변화에 따라 객체의 행위가 달라지게 만드는 패턴이며, 책임을 덧붙이는 확장 방식과는 다르다.
- 올바르게는 '상태를 객체로 분리하여 상태별 행동을 바꾼다'가 상태 패턴의 핵심이다.

오답 포인트

출제자는 '동적으로 추가'와 '서브 클래싱 대신'이라는 표현을 통해 확장 방법을 묻고 있다. 수험생은 '동적'이라는 단어만 보고 상태 변화나 반복 처리로 연결하기 쉽지만, 이 문항의 핵심은 행위 변화가 아니라 상속 대신 객체를 감싸서 기능을 늘리는 구조라는 점이다.

다시 나오면 이렇게 판단

설명에 '기능 추가', '책임 추가', '상속 대신 유연한 확장', '객체를 감싸서 확장'이 나오면 장식자 패턴을 먼저 판단한다. 반대로 인스턴스 하나면 싱글톤, 순차 접근이면 반복자, 상태에 따른 행위 변화면 상태 패턴으로 구분한다.

암기 포인트

Decorator는 덧입혀 확장, Singleton은 하나만 유지, Iterator는 순서대로 접근, State는 상태가 바뀌면 행동이 바뀐다.

NoteBase

문 12. 애자일(Agile) 개발 방법론에 대한 설명으로 가장 옳지 않은 것은?

① 애자일 방법론의 전제는 고객이 계속 새로운 요구사항을 제시하고 기존 요구사항을 변경한다는 것이다.

② 익스트림 프로그래밍(XP, eXtreme Programming)에서는 사용자 스토리 카드를 사용하여 요구사항을 수집한다.

③ 익스트림 프로그래밍에서는 초기에 미래에 대해 고려하여 시스템 설계를 완성하며 리팩토링(refactoring)을 통하여 내부의 구조를 변화시킨다.

④ 스크럼(Scrum) 방법론에서는 백로그(backlog)를 통하여 프로젝트 요구사항 및 이에 대한 변경관리를 수행한다.

정답 3번

출제 포인트

이 문제는 애자일, XP, 스크럼의 요구사항 처리 방식과 설계 철학을 구분하는 문제이다.

핵심 개념

애자일: 요구사항을 초기에 완전히 고정하기보다 변화에 유연하게 대응하는 반복·점진 개발 방식이다. 키워드: 변화 수용, 고객 협업.
XP: 사용자 스토리, 단순한 설계, 테스트, 리팩토링으로 빠르게 피드백을 반영하는 개발 방식이다. 키워드: user story, simple design, refactoring.
스크럼: 제품 백로그를 중심으로 스프린트 단위로 일을 관리하는 애자일 프레임워크이다. 키워드: backlog, sprint, 우선순위 관리.

NoteBase

선지별 해설

① O
애자일은 요구사항이 개발 도중 계속 추가되거나 변경될 수 있음을 전제로 하고, 그 변화에 유연하게 대응하는 방식이다.

② O
XP에서는 사용자 스토리 카드를 사용해 고객 요구를 짧고 이해하기 쉬운 형태로 정리하며, 이를 바탕으로 개발 우선순위를 정한다.

③ X
틀린 부분은 "초기에 미래에 대해 고려하여 시스템 설계를 완성"이다.
XP는 초기에 설계를 완성해 두는 방식이 아니라 단순한 설계로 시작해 반복적으로 개선하는 방식을 취한다.
올바르게는 '초기에는 단순한 설계로 시작하고, 리팩토링을 통해 외부 동작은 유지한 채 내부 구조를 지속적으로 개선한다'가 맞다.

④ O
스크럼은 제품 백로그를 통해 요구사항을 정리하고 우선순위를 관리하며, 변경 사항도 백로그에 반영해 관리한다.

오답 포인트

XP의 리팩토링을 '초기 대설계 후 구조 변경'으로 오해하면 ③을 맞는 설명처럼 보기 쉽다. XP는 Big Design Up Front보다 지속적인 설계 개선에 가깝다.

다시 나오면 이렇게 판단

애자일 계열 문항은 '요구사항 변화 수용', '초기 설계 완성 여부', '백로그 사용'을 먼저 본다. XP에 초기 완성 설계가 나오면 틀린 선지일 가능성이 높다.

암기 포인트

애자일은 변화 수용, XP는 사용자 스토리·단순설계·리팩토링, 스크럼은 백로그 중심 관리이다.

NoteBase

문 13. CMMI 모델 레벨 2(관리 단계)의 주요 프로세스 영역에 해당하지 않는 것은?

① 요구사항 관리(requirement management)

② 제품 품질 보증(product quality assurance)

③ 프로젝트 계획(project planning)

④ 위험 관리(risk management)

정답 4번

출제 포인트

이 문제는 CMMI 성숙도 레벨 2의 대표 프로세스 영역과 레벨 3 영역을 구분하는 문제이다.

핵심 개념

CMMI 레벨 2는 프로젝트 단위에서 기본 관리가 되는 단계로 요구사항 관리, 프로젝트 계획, 프로젝트 모니터링 및 통제, 형상 관리, 측정 및 분석, 공급자 계약 관리, 프로세스 및 제품 품질 보증 같은 관리 영역이 포함된다.
위험 관리는 조직적이고 선제적인 리스크 식별·분석·대응 체계를 다루는 영역으로 레벨 3에 속한다.
따라서 레벨 2는 기본적인 프로젝트 관리와 통제, 레벨 3은 이를 조직 차원에서 표준화·고도화하는 단계로 구분된다.

선지별 해설

① O
요구사항 관리는 CMMI 레벨 2의 REQM에 해당한다. 개발 범위와 요구 변경을 통제해 프로젝트를 관리 가능한 상태로 두는 기본 영역이므로 맞는 선지이다.

② O
제품 품질 보증은 통상 CMMI의 PPQA를 가리키며 정식 명칭은 프로세스 및 제품 품질 보증이다. 레벨 2의 주요 프로세스 영역에 포함되므로 취지는 맞는 선지이다.

③ O
프로젝트 계획은 CMMI 레벨 2의 PP에 해당한다. 일정, 자원, 예산, 작업 범위를 계획하는 핵심 관리 영역이므로 맞는 선지이다.

④ X
틀린 부분은 위험 관리를 레벨 2의 주요 프로세스 영역이라고 본 점이다.
올바르게는 위험 관리는 CMMI 레벨 3의 프로세스 영역이다.

NoteBase

오답 포인트

레벨 2와 레벨 3을 모두 관리 활동으로 묶어 외우면 위험 관리도 레벨 2처럼 보이기 쉽다. 그러나 레벨 2는 프로젝트 수행을 반복 가능하게 만드는 기본 관리 단계이고, 위험 관리는 한 단계 올라간 레벨 3의 조직적 관리 영역이라는 점이 함정이다.

다시 나오면 이렇게 판단

CMMI 문항에서는 먼저 레벨 2의 대표 묶음인 요구사항 관리, 프로젝트 계획, 품질 보증, 형상 관리를 떠올리면 된다. 선지에 위험 관리, 의사결정 분석, 조직 프로세스 초점처럼 고도화된 관리 항목이 나오면 레벨 3 가능성을 먼저 본다.

암기 포인트

레벨 2는 기본 프로젝트 관리, 위험 관리는 레벨 3이다.

NoteBase

문 14. 소스 코드를 테스트할 때 소스 코드 내의 분기를 결정하는 개별 조건식이 다른 개별 조건식에 영향을 받지 않고 전체 조건식의 결과에 독립적으로 영향을 주는 테스트 케이스를 선정하는 방법을 MCDC (Modified Condition Decision Coverage)라고 한다. <보기 1>에 주어진 조건식에서 MCDC를 만족하는 최소의 테스트 케이스 집합을 <보기 2>에서 고른 것은?

<보기 1>

조건식: (A and B) or C

<보기 2>

	A	B	C	결과
ㄱ	True	True	True	True
ㄴ	True	True	False	True
ㄷ	True	False	True	True
ㄹ	True	False	False	False
ㅁ	False	True	True	True
ㅂ	False	True	False	False
ㅅ	False	False	True	True
ㅇ	False	False	False	False

① ㄴ, ㄹ, ㅁ, ㅂ

② ㄷ, ㄹ, ㅁ, ㅂ

③ ㄴ, ㄷ, ㄹ, ㅁ, ㅂ

④ ㄴ, ㄹ, ㅁ, ㅅ, ㅇ

정답 1번

출제 포인트

이 문제는 조건식 (A B) C에서 MCDC를 만족하도록 각 조건 A, B, C가 결과를 독립적으로 바꾸는 테스트 쌍을 최소 개수로 구성하는지를 묻는 문제이다.

NoteBase

조건 정리

주어진 조건식은 (A B) C이다.
MCDC는 한 조건만 바꾸고 나머지 조건은 고정했을 때 전체 결과가 바뀌는 쌍이 각 조건마다 있어야 한다.
이 문항에서는 A, B, C 세 조건 각각에 대해 독립 영향 쌍을 만들어야 하므로, 서로 다른 쌍이 케이스를 공유하는지가 최소 여부를 결정한다.

사용할 공식

MCDC 판단 기준은 각 조건마다 다음을 찾는 것이다.
한 쌍의 테스트 케이스에서
1. 확인하려는 조건만 값이 달라진다.
2. 다른 조건들은 모두 동일하다.
3. 전체 결과가 True와 False로 바뀐다.
이 기준을 쓰는 이유는 MCDC가 개별 조건의 독립적인 결정 영향력을 증명하는 커버리지이기 때문이다.

계산 과정

1. A의 독립 영향은 $B = True, C = False$를 고정한 ㄴ과 ㅂ으로 확인할 수 있다. ㄴ은 (T T) F=True, ㅂ은 (F T) F=False이므로 결과가 바뀐다.
2. B의 독립 영향은 $A = True, C = False$를 고정한 ㄴ과 ㄹ로 확인할 수 있다. ㄴ은 True, ㄹ은 False이므로 결과가 바뀐다.
3. C의 독립 영향은 $A = False, B = True$를 고정한 ㅁ과 ㅂ으로 확인할 수 있다. ㅁ은 True, ㅂ은 False이므로 결과가 바뀐다.
4. 따라서 ㄴ, ㄹ, ㅁ, ㅂ 네 개만으로 A, B, C를 모두 증명할 수 있다. 이보다 적은 3개 케이스로는 세 조건의 독립 쌍을 모두 만들기 어렵기 때문에 최소 집합은 4개 구성이다.

NoteBase

기출OX 해설집 · 공무원/서울시/7급 · 소프트웨어공학 · 2020 · 10월

선지별 해설

① O

ㄴ, ㄹ, ㅁ, ㅂ은 MCDC를 만족하는 최소 집합이다. ㄴ-ㅂ은 *A*의 독립 영향, ㄴ-ㄹ은 *B*의 독립 영향, ㅁ-ㅂ은 *C*의 독립 영향을 보여 준다. 네 케이스가 서로 재사용되므로 최소 구성이 된다.

② X

ㄷ, ㄹ, ㅁ, ㅂ에는 *A*가 독립적으로 결과를 바꾸는 쌍이 없다. 예를 들어 ㄷ과 ㅁ은 *A*뿐 아니라 *B*도 함께 바뀌고, ㄹ과 ㅂ도 *A*와 *B*가 동시에 바뀐다. 틀린 표현은 'MCDC를 만족한다'는 부분이고, 올바르게는 *A*를 확인하려면 ㄴ-ㅂ처럼 *B*, *C*를 고정한 쌍이 포함되어야 한다.

③ X

ㄴ, ㄷ, ㄹ, ㅁ, ㅂ은 MCDC를 만족하기는 하지만 최소 집합은 아니다. 이미 ㄴ, ㄹ, ㅁ, ㅂ만으로 세 조건의 독립 영향이 모두 증명되므로 ㄷ은 불필요한 추가 케이스이다. 틀린 표현은 '최소 테스트 케이스 집합'이라는 부분이고, 올바르게는 'MCDC를 만족하지만 최소는 아닌 집합'이다.

④ X

ㄴ, ㄹ, ㅁ, ㅅ, ㅇ은 *A*의 독립 영향을 증명하지 못한다. ㄹ-ㅇ은 *A*만 다르지만 결과가 둘 다 False이고, ㅁ-ㅅ은 *A*만 다르지만 결과가 둘 다 True라서 결과 변화가 없다. 틀린 표현은 'MCDC를 만족한다'는 부분이고, 올바르게는 *A* 검증용으로 ㄴ-ㅂ 같은 쌍이 들어가야 한다.

오답 포인트

이 문항의 함정은 단순히 True와 False가 섞여 있는 케이스를 많이 고르는 것이 아니라, 한 조건만 바뀌고 결과까지 바뀌는 쌍이 있는지를 따져야 한다는 점이다. 특히 ③은 조건은 만족하지만 '최소'를 놓치기 쉬운 선택지이고, ④는 *A*만 바뀌는 행이 있어 보여도 결과가 바뀌지 않아 MCDC 조건을 충족하지 못한다.

다시 나오면 이렇게 판단

MCDC 문제는 먼저 조건 하나를 정한 뒤 나머지 조건을 고정한 두 행을 찾고, 그 두 행에서 결과가 뒤집히는지를 본다. 그다음 같은 행을 다른 조건 검증에도 재사용할 수 있는지 확인하면 최소 집합 여부까지 판단할 수 있다.

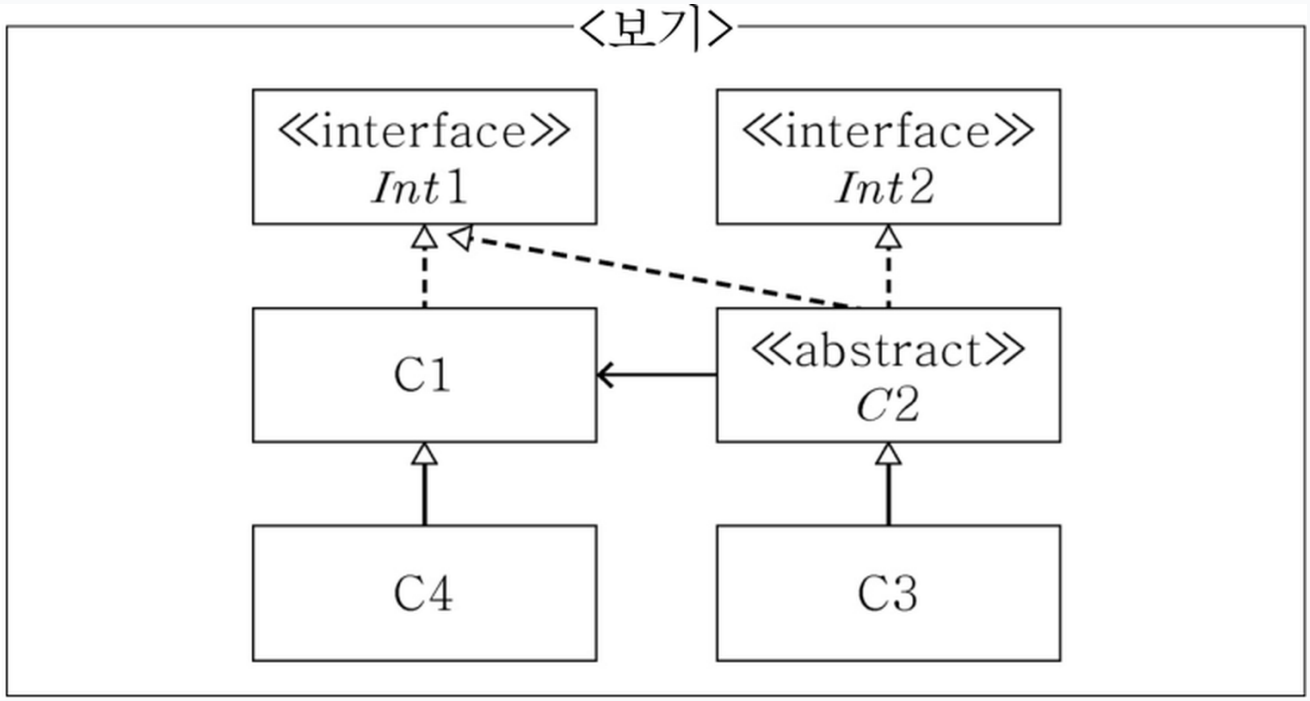
암기 포인트

MCDC는 '한 조건만 변경, 나머지는 고정, 결과는 반전'이 핵심이며, 최소 집합은 같은 행을 여러 조건 검증에 공유할수록 유리하다.

NoteBase

문 15. <보기>의 클래스 다이어그램에 대한 자바 코드를 컴파일할 때 오류가 발생하지 않는 것은?

<보기>



① Int1 a = new C1();
Int2 b = new C1();

② Int1 c = new C2();
C3 d = new C2();

③ Int1 e = new C3();
C2 f = new C3();

④ Int1 g = new C4();
Int2 h = new C4();

정답 3번

출제 포인트

이 문제는 클래스 다이어그램에서 실체화와 상속 관계를 읽고, 인터페이스 대입 가능성과 추상 클래스 인스턴스화 가능 여부를 함께 판단하는 문제이다.

NoteBase

선지별 해설

1번: X. Int1 a = new C1();는 C1이 Int1을 실체화하므로 가능하다. 그러나 Int2 b = new C1();는 C1이 Int2를 실체화하지 않으므로 컴파일 오류가 난다. 틀린 표현은 Int2 b = new C1();이며, 올바르게는 Int2 b = new C2(); 또는 Int2 b = new C3();이다.
2번: X. Int1 c = new C2();는 C2가 Int1 계통이더라도 C2가 추상 클래스이므로 new C2() 자체가 불가능하다. C3 d = new C2();도 추상 클래스 인스턴스화가 불가능하고, 상위 타입인 C2를 하위 타입 변수 C3에 그대로 대입할 수 없으므로 오류이다. 틀린 표현은 두 줄 모두의 new C2()이며, 올바르게는 Int1 c = new C3();, C3 d = new C3();처럼 구체 클래스를 생성해야 한다.
3번: O. C3는 C2를 상속하고, C2는 C1을 상속하면서 Int1과 Int2를 실체화하므로 C3 객체는 Int1 타입으로 참조할 수 있다. 또한 C2 f = new C3();는 하위 클래스 객체를 상위 클래스 참조변수에 대입하는 업캐스팅이므로 정상이다.
4번: X. Int1 g = new C4();는 C4가 C1을 상속하므로 가능하다. 그러나 C4는 Int2를 실체화한 계통이 아니므로 Int2 h = new C4();는 컴파일 오류가 난다. 틀린 표현은 Int2 h = new C4();이며, 올바르게는 Int2 h = new C3();이다.

오답 포인트

추상 클래스라도 상위 타입으로는 참조할 수 있다는 사실과 추상 클래스는 직접 생성할 수 없다는 사실을 섞어 버리면 ②와 ③을 혼동하게 된다. 또 상속으로 물려받는 인터페이스와 직접 실체화한 인터페이스를 구분하지 않으면 ①과 ④의 두 번째 줄을 놓치기 쉽다.

다시 나오면 이렇게 판단

다이어그램 문제는 먼저 각 클래스가 최종적으로 어떤 인터페이스를 구현하는지 상속 사슬 끝까지 따라간 뒤, new 뒤의 클래스가 추상인지 확인하고, 마지막에 좌변 타입으로 업캐스팅이 가능한지만 보면 된다.

암기 포인트

인터페이스 구현은 하위 클래스로 전파되고, 추상 클래스는 상위 타입으로 참조할 수 있어도 new로 직접 생성할 수 없다.

NoteBase

문 16. <보기>에서 설명하는 소프트웨어 품질 요소를 옳게 짝지은 것은?

<보기>
(가) 사용자의 요구사항을 만족시키는 정도
(나) 프로그램 수행을 위한 자원의 최적화 정도
(다) 다른 시스템에 결합시키는 데 요구되는 노력의 정도

① (가): 유지보수성 (maintainability), (나): 상호운영성 (interoperability), (다): 정확성 (correctness)
② (가): 정확성 (correctness), (나): 상호운영성 (interoperability), (다): 효율성 (efficiency)
③ (가): 정확성 (correctness), (나): 효율성 (efficiency), (다): 상호운영성 (interoperability)
④ (가): 상호운영성 (interoperability), (나): 효율성 (efficiency), (다): 유지보수성 (maintainability)

정답 3번

출제 포인트

이 문제는 소프트웨어 품질 요소의 정의를 정확성, 효율성, 상호운영성, 유지보수성으로 구별할 수 있는지를 묻는다.

핵심 개념

정확성: 사용자의 요구사항과 명세를 만족시키는 정도이다.
효율성: 기능 수행에 필요한 시간과 자원 사용을 얼마나 최적화했는지를 나타내는 정도이다.
상호운영성: 다른 시스템과 결합하거나 정보를 주고받는 데 필요한 노력의 정도이다.
유지보수성: 오류 수정이나 기능 변경을 쉽고 안정적으로 수행할 수 있는 정도이다.

선지별 해설

1번 X: (가)를 유지보수성으로 본 점이 틀리다. (가)는 사용자의 요구사항을 만족시키는 정도이므로 정확성이다. 또한 (다)를 정확성으로 본 점도 틀리다. (다)는 다른 시스템에 결합시키는 데 요구되는 노력의 정도이므로 상호운영성이다.
2번 X: (가)는 정확성으로 맞지만 (나)를 상호운영성으로 본 점이 틀리다. (나)는 프로그램 수행을 위한 자원의 최적화 정도이므로 효율성이다.
3번 O: (가)는 정확성, (나)는 효율성, (다)는 상호운영성으로 모두 올바르게 대응된다.
4번 X: (가)를 상호운영성으로 본 점이 틀리다. (가)는 정확성이다. (다)를 유지보수성으로 본 점도 틀리다. (다)는 상호운영성이다.

NoteBase

오답 포인트

정확성을 결과의 세밀함 정도로 오해하거나, 상호운영성과 유지보수성을 모두 시스템 외부와 관련된 성질로 뭉뚱그리면 틀리기 쉽다. 이 문항은 정의의 핵심 명사를 바꿔 끼우는 방식으로 혼동을 유도한다.

다시 나오면 이렇게 판단

요구사항 충족이면 정확성, 수행 자원 최적화이면 효율성, 다른 시스템과의 연동 노력이면 상호운영성, 수정과 개선의 용이성이면 유지보수성으로 바로 대응하면 된다.

암기 포인트

정확성은 요구 충족, 효율성은 자원 절약, 상호운영성은 연동 용이, 유지보수성은 수정 용이이다.

NoteBase

문 17. 로버트 마틴이 주장한 다섯 가지 설계 원칙(SOLID) 중, 기존의 코드를 변경하지 않으면서 새로운 기능을 추가할 수 있도록 설계하는 원칙은?

① 단일 책임 원칙(SRP, Single Responsibility Principle)

② 개방 폐쇄 원칙(OCP, Open Closed Principle)

③ 인터페이스 분리 원칙(ISP, Interface Segregation Principle)

④ 리스코프의 교체 원칙(LSP, Liskov Substitution Principle)

정답 2번

출제 포인트

이 문제는 SOLID 원칙 중 기존 코드를 수정하지 않고 기능 확장을 가능하게 하는 개방 폐쇄 원칙의 의미를 묻는 문제이다.

핵심 개념

SRP는 클래스나 모듈이 하나의 책임만 가지도록 나누는 원칙이다. OCP는 확장에는 열려 있고 수정에는 닫혀 있도록 설계하는 원칙이다. ISP는 클라이언트가 사용하지 않는 기능까지 강요하지 않도록 인터페이스를 작게 분리하는 원칙이다. LSP는 하위 타입이 상위 타입을 대체해도 프로그램의 의미가 깨지지 않아야 한다는 원칙이다.

선지별 해설

① X: 단일 책임 원칙은 하나의 클래스나 모듈이 하나의 책임만 가지게 하는 원칙이다. 기능 추가를 위해 기존 코드를 수정하지 않도록 하는 확장성 원칙을 직접 가리키지는 않는다. 틀린 표현은 발문의 '새로운 기능을 추가'에 ①을 대응시킨 점이며, 올바르게는 '하나의 책임만 맡도록 분리하는 원칙'이다.
② O: 개방 폐쇄 원칙은 새로운 기능이 필요할 때 기존 코드를 크게 고치기보다 확장 지점을 추가해 기능을 늘리도록 설계하는 원칙이다. 발문의 '기존의 코드를 변경하지 않으면서 새로운 기능을 추가'와 정확히 일치한다.
③ X: 인터페이스 분리 원칙은 클라이언트가 사용하지 않는 메서드에 의존하지 않게 인터페이스를 작게 나누는 원칙이다. 확장 시 기존 구현 수정 여부를 핵심으로 삼는 원칙은 아니다. 틀린 표현은 ③을 확장성 원칙으로 본 점이며, 올바르게는 '불필요하게 큰 인터페이스를 분리하는 원칙'이다.
④ X: 리스코프의 교체 원칙은 자식 클래스가 부모 클래스를 대체해도 동작의 일관성이 유지되어야 한다는 원칙이다. 이는 상속 관계의 치환 가능성을 다루며, 기존 코드 수정 없이 기능을 추가하는 원칙과는 다르다. 올바르게는 '상위 타입 자리에 하위 타입을 안전하게 넣을 수 있어야 한다는 원칙'이다.

NoteBase

오답 포인트

SOLID는 모두 객체지향 설계 원칙이라 약어만 보고 헷갈리기 쉽다. 이 문항은 '기존 코드를 변경하지 않으면서 기능 추가'라는 문장을 보고 OCP의 핵심 문구인 '확장에는 열려 있고 수정에는 닫혀 있음'을 떠올리면 바로 풀린다.

다시 나오면 이렇게 판단

발문이 기능 추가, 확장, 변경 최소화, 수정 없이 추가를 강조하면 OCP를 먼저 떠올린다. 책임 하나만 강조하면 SRP, 인터페이스를 잘게 나누는 내용이면 ISP, 상속 객체의 대체 가능성을 묻으면 LSP로 구분한다.

암기 포인트

OCP는 Open for Extension, Closed for Modification이다.

NoteBase

문 18. <보기>에 제시된 코드에 대한 순환복잡도를 계산하여 기본 경로 테스팅을 수행하고자 한다. 이 코드의 순환복잡도 수치는?

<보기>

```
while(True):
        code = ''
        for a in range(12):
                code+=str(chr(random.randint(65,90)))
        if len(Model.objects.filter(name__contains=code)) >0:
                continue
        else :
                break
print(code)
```

① 3

② 4

③ 5

④ 6

정답 2번

출제 포인트

이 문제는 코드의 분기 수를 세어 순환복잡도 $V(G) = P + 1$를 구하는 문제이다.

조건 정리

주어진 구조는 `while` 1개, `for` 1개, `if` 1개이다.
구해야 할 값은 기본 경로 수와 같은 순환복잡도이다.
`continue`와 `break`는 흐름 제어문이지만 새로운 판단 조건을 만들지 않는다.

사용할 공식

$V(G) = P + 1$이다.
여기서 P는 predicate node, 즉 판단 노드 수이다.
이 공식을 쓰는 이유는 기본 경로 테스팅에서 독립 경로 수가 순환복잡도와 같기 때문이다.

NoteBase

계산 과정

1. `while(True)`는 반복 여부를 판단하는 루프 조건이므로 판단 노드 1개이다.
2. `for a in range(12)`도 반복 제어 조건이 있으므로 판단 노드 1개이다.
3. `if len(Model.objects.filter(name__contains=code)) > 0`는 참/거짓 분기이므로 판단 노드 1개이다.
4. 따라서 $P = 3$이고, $V(G) = 3 + 1 = 4$이다.

선지별 해설

① X: 3은 `while(True)`를 판단 노드에서 빠뜨릴 때 나오는 값이다. 틀린 표현은 `while`를 단순 반복문으로만 보고 분기를 세지 않는 점이며, 올바르게는 `while`도 순환복잡도 계산에서 1개의 판단 노드로 본다.
② O: `while`, `for`, `if`의 판단 노드가 각각 1개씩이므로 총 3개이고 순환복잡도는 4이다.
③ X: 5는 `else`까지 별도 판단 노드로 중복 계산할 때 나오는 값이다. 틀린 표현은 `if-else`를 두 개의 판단으로 보는 점이며, 올바르게는 `if` 문 전체가 1개의 판단 노드이다.
④ X: 6은 `continue`, `break` 같은 제어문까지 분기 수로 과대 계산할 때 나오는 값이다. 틀린 표현은 흐름 제어문을 predicate로 보는 점이며, 올바르게는 새 조건식을 갖는 구문만 판단 노드로 센다.

오답 포인트

반복문 안의 `continue`, `break`, `else`를 모두 별도 분기로 세면 값이 커진다. 이 문항의 함정은 '판단 조건이 있는가'와 '단순 흐름 전환인가'를 구분하지 못하게 만드는 데 있다.

다시 나오면 이렇게 판단

순환복잡도 문항은 먼저 조건식을 가진 구문만 표시하고, 그 개수를 P로 둔 다음 $V(G) = P + 1$로 계산하면 된다. `while`, `for`, `if`는 세고 `else`, `continue`, `break`는 세지 않는다.

암기 포인트

순환복잡도는 판단 노드 수에 1을 더한 값이며, 조건식 없는 제어문은 세지 않는다.

NoteBase

문 19. 올바른 요구사항 명세서를 작성하기 위한 주의 사항으로 가장 옳지 않은 것은?

① 요구사항 명세서는 소프트웨어 개발의 전 과정을 주도하므로, 사용자가 아니라 개발자 중심으로 이해하기 쉽게 작성되어야 한다.

② 요구사항 명세서에는 원하는 기능을 정확하고 완벽하게 기술해야 한다.

③ 요구사항 명세서에는 2가지 이상의 해석이 발생하지 않도록 모호하지 않은 표현을 써야 한다.

④ 요구사항 명세서는 시스템 인수를 위한 테스트 기준을 제시해야 한다.

정답 1번

출제 포인트

이 문제는 요구사항 명세서의 품질 기준인 사용자 지향성, 완전성, 비모호성, 검증 가능성을 구별할 수 있는지를 묻는다.

핵심 개념

요구사항 명세서는 사용자의 요구를 정확하게 문서화한 기준 문서이다. 좋은 명세서는 사용자 요구를 기준으로 이해 가능해야 하고, 필요한 기능을 빠짐없이 담아야 하며, 해석이 하나로 수렴해야 하고, 나중에 시험과 인수로 검증할 수 있어야 한다.

선지별 해설

① X. 틀린 부분은 "사용자가 아니라 개발자 중심으로"이다. 요구사항 명세서는 개발 전 과정을 이끄는 기준 문서이지만 출발점은 사용자 요구이므로 사용자와 개발자 모두가 이해할 수 있게, 특히 사용자 요구 중심으로 작성되어야 한다.
② O. 요구사항 명세서는 필요한 기능과 제약을 정확하고 완전하게 기술해야 한다. 빠진 기능이나 불명확한 조건이 있으면 이후 설계와 구현에서 오류가 커진다.
③ O. 요구사항 명세서는 한 문장이 둘 이상으로 해석되지 않도록 비모호하게 써야 한다. 모호한 표현은 개발자마다 다른 구현을 낳아 요구 불일치를 만든다.
④ O. 요구사항 명세서는 검증 가능해야 하므로 시스템 인수 시험의 기준이 될 수 있어야 한다. 명세서의 각 요구는 테스트로 충족 여부를 확인할 수 있어야 한다.

오답 포인트

요구사항 명세서가 개발자도 사용하는 문서라는 점 때문에 곧바로 개발자 중심 문서라고 오해하기 쉽다. 이 문서는 개발자가 읽는 문서이기도 하지만, 내용의 기준은 사용자 요구이며 인수 판단의 근거가 되는 문서라는 점이 핵심이다.

NoteBase

다시 나오면 이렇게 판단

요구사항 명세서 관련 선지는 사용자 요구 기준인지, 빠짐없이 완전한지, 문장이 한 뜻만 갖는지, 테스트로 확인 가능한지를 차례로 보면 판단할 수 있다.

암기 포인트

요구사항 명세서는 사용자 요구를 정확·완전·명확·검증 가능하게 적는 문서이다.

NoteBase

문 20. 테스트 오라클(test oracle)에 대한 설명으로 가장 옳은 것은?

① 테스트 케이스(test case)에 대해 프로그램의 실제 실행 결과가 올바른 결과인지를 판단하는 메커니즘(mechanism)이다.

② 테스트하려는 모듈을 호출하는 시스템을 대신하여 간단히 구동하도록 작성한 것이다.

③ 테스트하려는 모듈에 의해 호출되는 가상 모듈이다.

④ 테스트를 수행한 후 관찰된 결과를 저장한 것이다.

정답 1번

출제 포인트

이 문제는 테스트 오라클, 테스트 드라이버, 테스트 스텁, 실행 결과 기록의 역할 차이를 구분하는 문제이다.

핵심 개념

테스트 오라클은 테스트 케이스의 실제 결과가 기대 결과와 일치하는지를 판정하는 기준 또는 메커니즘이다.
테스트 드라이버는 상위 호출 모듈을 대신하여 테스트 대상 모듈을 호출하는 임시 프로그램이다.
테스트 스텁은 테스트 대상 모듈이 호출하는 하위 모듈을 대신하는 가상 모듈이다.
실행 결과 기록은 테스트 후 관찰값을 저장한 결과물일 뿐 정오를 판정하지 않는다.

선지별 해설

① O
테스트 오라클은 실제 실행 결과가 올바른지, 즉 기대 결과와 맞는지를 판단하는 메커니즘이므로 옳다.

② X
틀린 부분은 테스트 오라클의 설명처럼 서술한 점이다.
실제로는 테스트하려는 모듈을 호출하는 상위 시스템을 대신해 구동하는 테스트 드라이버이다.

③ X
틀린 부분은 테스트 오라클의 설명처럼 서술한 점이다.
실제로는 테스트 대상 모듈에 의해 호출되는 하위 모듈을 흉내 내는 테스트 스텁이다.

④ X
틀린 부분은 관찰된 결과를 저장한 것을 테스트 오라클로 본 점이다.
실제로는 실행 결과나 테스트 로그에 가까우며, 테스트 오라클은 그 저장된 결과가 맞는지 판정하는 역할을 한다.

NoteBase

오답 포인트

드라이버는 테스트 대상을 호출하고, 스텁은 테스트 대상에게 호출된다는 방향 차이를 놓치기 쉽다. 또한 결과를 기록하는 기능과 결과의 정오를 판정하는 기능을 혼동하면 ④에 흔들리기 쉽다.

다시 나오면 이렇게 판단

선지를 보면 먼저 역할 동사를 잡으면 된다. 정오를 판정하면 오라클, 테스트 대상을 호출하면 드라이버, 테스트 대상에게 호출되면 스텁, 수행 결과를 남기면 로그나 결과 기록이다.

암기 포인트

오라클은 판정, 드라이버는 호출, 스텁은 응답, 로그는 기록이다.

NoteBase