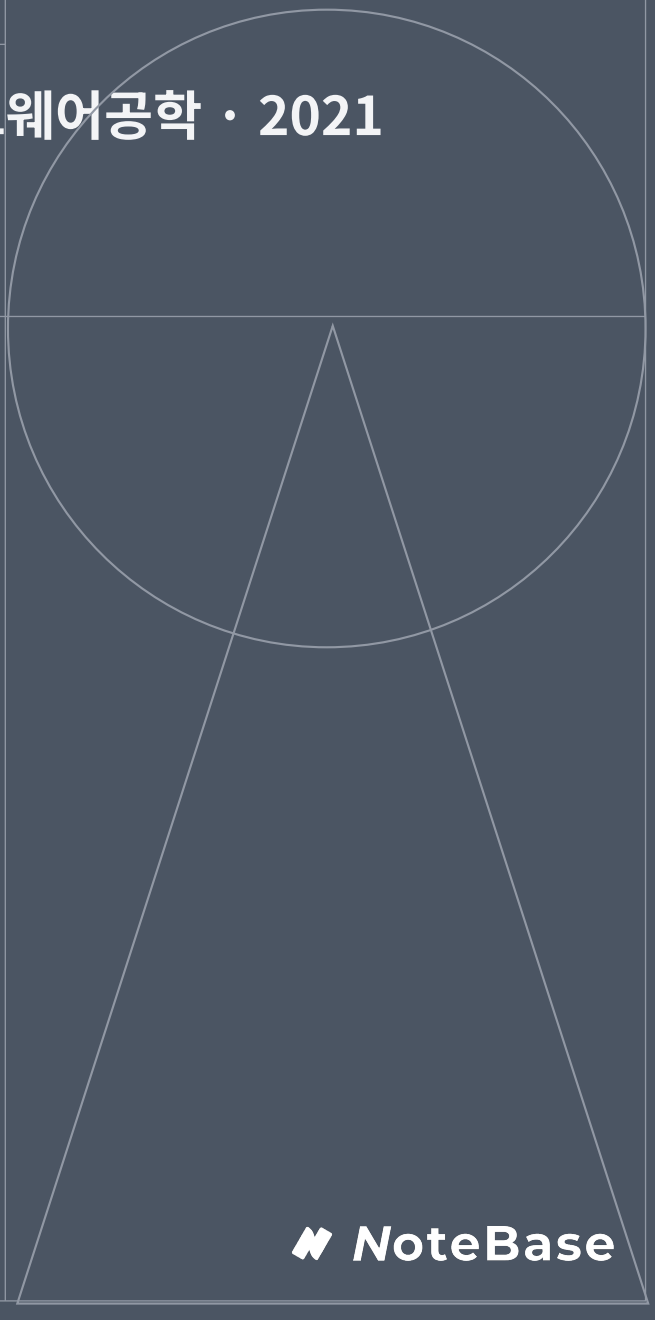
기출OX 해설집

공무원/서울시/7급 · 소프트웨어공학 · 2021



NoteBase

기출OX 해설집

공무원/서울시/7급 · 소프트웨어공학 · 2021

공무원/서울시/7급 · 소프트웨어공학 · 2021

문 1. 형상 관리(configuration management)의 활동 중 형상 식별(configuration identification) 단계에서 수행되는 활동을 <보기>에서 모두 고른 것은?

<보기>
ㄱ. 형상 관리 대상을 선정
ㄴ. 변경사항 요청 및 심사
ㄷ. 형상 관리를 위한 식별자 규칙 선정
ㄹ. 변경을 완료하고 새로운 버전 번호 부여
ㅁ. 형상 관리가 계획서대로 관리되고 있는지 검사

① ㄱ, ㄷ
② ㄴ, ㄷ
③ ㄷ, ㄹ
④ ㄹ, ㅁ

정답 1번

출제 포인트

이 문제는 소프트웨어 형상 관리 활동 중 형상 식별 단계에 해당하는 활동과 변경 통제·감사 활동을 구분하는지를 묻는 문제이다.

핵심 개념

형상 식별은 무엇을 형상 항목으로 관리할지 정하고, 각 항목을 어떻게 구분하고 이름 붙일지 정하는 단계이다.
변경 통제는 변경 요청을 접수하고 심사하여 승인 여부를 결정하는 단계이다.
형상 감사는 형상 관리가 계획과 절차에 맞게 수행되었는지 확인하는 단계이다.
이 문제의 구별축은 관리 대상을 정하고 식별 체계를 세우는가, 아니면 변경을 심사·확인하는가이다.

NoteBase

선지별 해설

① O
ㄱ의 형상 관리 대상 선정은 어떤 산출물을 형상 항목으로 둘지 정하는 일이므로 형상 식별에 해당한다.
ㄷ의 식별자 규칙 선정도 각 형상 항목을 일관되게 구분하기 위한 명명·번호 체계를 정하는 일이므로 형상 식별에 해당한다.

② X
ㄴ의 변경사항 요청 및 심사는 형상 식별이 아니라 변경 통제 활동이다.
ㄷ만 형상 식별에 해당하므로 조합이 틀리다.
틀린 표현은 'ㄴ도 형상 식별 단계의 활동'이라는 판단이다. 올바르게는 ㄴ은 변경 통제 활동이다.

③ X
ㄷ은 형상 식별이 맞지만 ㄹ의 변경 완료 후 새로운 버전 번호 부여는 변경 결과를 반영하는 버전 관리 성격의 활동으로, 형상 식별 단계의 핵심 활동으로 보지 않는다.
틀린 표현은 'ㄹ도 형상 식별 단계의 활동'이라는 판단이다. 올바르게는 ㄹ은 변경 반영 이후의 버전 관리 측면 활동이다.

④ X
ㄹ은 형상 식별 단계가 아니고, ㅁ의 형상 관리가 계획서대로 관리되고 있는지 검사하는 일은 형상 감사에 해당한다.
틀린 표현은 'ㄹ, ㅁ이 형상 식별 단계의 활동'이라는 판단이다. 올바르게는 ㅁ은 형상 감사 활동이다.

오답 포인트

식별자 규칙 선정과 버전 번호 부여를 모두 '번호를 붙이는 일'로 묶어 같은 단계로 착각하기 쉽다. 그러나 형상 식별은 관리 대상과 식별 체계를 정하는 단계이고, 변경 요청 심사나 계획 준수 검사는 각각 변경 통제와 형상 감사에 속한다.

다시 나오면 이렇게 판단

선지에 대상 선정, 형상 항목 정의, 명명 규칙, 기준선 설정처럼 '무엇을 어떻게 식별할지'가 나오면 형상 식별로 본다. 반대로 변경 요청 심사, 변경 반영, 감사·검사는 식별 단계 밖의 활동으로 구분한다.

암기 포인트

형상 식별은 '대상 선정 + 식별 규칙', 변경 통제는 '변경 심사', 형상 감사는 '계획 준수 검사'이다.

NoteBase

문 2. 컴포넌트 기반 소프트웨어 개발(CBSD)에 대한 설명으로 가장 옳지 않은 것은?

① 고객이 원하는 요구들을 지속적으로 피드백한다.

② 이미 개발된 컴포넌트를 조립하여 시스템을 개발하는 과정이다.

③ 도메인 영역에서 재사용이 가능한 기능적 요구사항을 명확하게 정의하는 것이 필요하다.

④ 기능의 추가나 변경이 필요한 경우 요구에 따라 커스터마이징이 가능하다.

정답 1번

출제 포인트

이 문제는 CBSD의 핵심이 고객과의 지속적 피드백이 아니라 재사용 가능한 컴포넌트의 식별, 조립, 커스터마이징에 있음을 구별할 수 있는지를 묻는다.

핵심 개념

CBSD는 이미 개발되고 검증된 컴포넌트를 재사용하여 시스템을 조립하는 개발 방식이다. 핵심 키워드는 재사용, 조립, 도메인 분석, 컴포넌트 선정, 커스터마이징이다.

선지별 해설

1번 X: "고객이 원하는 요구들을 지속적으로 피드백한다"는 표현은 애자일 개발의 반복적 고객 피드백 특징에 가깝다. CBSD의 핵심은 고객 피드백 자체보다 재사용 가능한 컴포넌트를 식별하고 조립하는 데 있다. 틀린 표현은 "지속적으로 피드백한다"이고, 올바르게는 "재사용 가능한 컴포넌트를 선정·조립하여 요구를 구현한다"이다.
2번 O: CBSD는 이미 개발된 컴포넌트를 조립하여 시스템을 구축하는 개발 방식이므로 옳다. 구현의 중심이 새 기능의 전면적 재개발이 아니라 기존 컴포넌트의 재사용에 있다.
3번 O: CBSD에서는 도메인 분석을 통해 재사용 가능한 기능 요구를 명확히 정의해야 적합한 컴포넌트를 찾고 조합할 수 있으므로 옳다. 요구가 불명확하면 재사용성과 적합성 판단이 어려워진다.
4번 O: 기존 컴포넌트가 요구에 완전히 맞지 않을 때 설정, 인터페이스 조정, 확장 등의 방식으로 커스터마이징할 수 있으므로 옳다. CBSD는 조립과 함께 필요한 범위의 맞춤 조정을 전제로 한다.

오답 포인트

1번의 "지속적으로 피드백"이라는 문구 때문에 반복적 개발방법론과 CBSD를 혼동하기 쉽다. 이 문항은 개발 절차의 분위기를 묻는 것이 아니라 CBSD의 중심 메커니즘이 재사용과 조립인지 여부를 묻는다.

다시 나오면 이렇게 판단

선지에 고객 피드백, 반복 개발, 스프린트 같은 표현이 나오면 먼저 애자일류 특징인지 확인한다. 반대로 재사용, 컴포넌트 선정, 조립, 도메인 분석, 커스터마이징이 나오면 CBSD 특징으로 판단할 수 있다.

NoteBase

암기 포인트

CBSD는 새로 만드는 개발이 아니라 재사용 가능한 컴포넌트를 찾아 조립하고 필요하면 커스터마이징하는 개발 방식이다.

NoteBase

문 3. 시퀀스(sequence) 다이어그램에 대한 설명으로 가장 옳지 않은 것은?

① 각각의 유스케이스(use case)에 대해서 작성하는 것이 일반적이다.

② 유스케이스 시나리오를 확장하여 시스템 내부의 객체들이 상호작용하는 과정까지는 나타낼 수 없다.

③ 데이터의 송수신은 매개변수와 반환값으로 표현된다.

④ 상호작용 과정에서 어떠한 데이터를 주고받았는지 밝힐 수 있다.

정답 2번

출제 포인트

이 문제는 시퀀스 다이어그램이 유스케이스 시나리오를 시간 순서의 객체 상호작용으로 구체화하는 도구인지, 그리고 메시지에 매개변수와 반환값을 실어 표현하는지 묻는 문제이다.

핵심 개념

시퀀스 다이어그램은 특정 유스케이스가 수행될 때 객체들이 시간 순서대로 어떤 메시지를 주고받는지 나타내는 상호작용 다이어그램이다.
유스케이스 다이어그램은 기능의 범위와 행위자를 개괄하고, 시퀀스 다이어그램은 그 유스케이스 내부 절차를 객체 수준으로 풀어낸다.
메시지에는 호출 시 전달되는 매개변수와 처리 결과인 반환값을 함께 표시할 수 있다.

NoteBase

선지별 해설

① O
각 유스케이스의 흐름을 상세화할 때 시퀀스 다이어그램을 별도로 작성하는 경우가 일반적이다. 유스케이스별 대표 시나리오를 객체 간 메시지 흐름으로 풀어내기 때문이다.

② X
틀린 부분은 "시스템 내부의 객체들이 상호작용하는 과정까지는 나타낼 수 없다"이다. 시퀀스 다이어그램은 바로 그 내부 객체들의 상호작용 과정을 시간 순서대로 나타내는 UML 다이어그램이다. 올바르게는 "유스케이스 시나리오를 확장하여 시스템 내부 객체들의 상호작용 과정까지 나타낼 수 있다"이다.

③ O
시퀀스 다이어그램의 메시지에는 메서드 호출 시 넘기는 매개변수와 실행 후 돌려주는 반환값을 표기할 수 있다. 따라서 데이터 송수신을 매개변수와 반환값으로 표현한다는 설명은 맞다.

④ O
메시지 이름과 전달 인자를 보면 상호작용 과정에서 어떤 데이터를 주고받는지 파악할 수 있다. 시퀀스 다이어그램은 단순 순서뿐 아니라 전달 정보까지 드러낼 수 있다.

오답 포인트

②는 시퀀스 다이어그램과 유스케이스 다이어그램의 역할을 뒤바꾸는 표현이 함정이다. 유스케이스 다이어그램은 외부 기능 관점, 시퀀스 다이어그램은 내부 객체 상호작용 관점이라는 축을 놓치면 ②를 맞는 설명처럼 보기 쉽다. 또한 발문이 "가장 옳지 않은 것"이므로 틀린 설명 하나를 골라야 한다는 점도 함께 확인해야 한다.

다시 나오면 이렇게 판단

시퀀스 다이어그램 문제가 나오면 먼저 "객체 간 메시지를 시간 순서대로 보여 주는가"를 기준으로 판단한다. 유스케이스를 객체 상호작용 수준으로 상세화한다면 시퀀스 다이어그램에 해당하고, 메시지의 매개변수·반환값까지 표현할 수 있으면 옳은 설명으로 본다.

암기 포인트

시퀀스 다이어그램은 유스케이스의 내부 절차를 객체 메시지 흐름으로 펼친 그림이다.

NoteBase

문 4. 설계 단계에서 고려할 사항으로 가장 옳지 않은 것은?

① 설계 과정에서 추상화 메커니즘을 적절히 이용한다면 구조적이고 단계적인 설계를 할 수 있다.

② 정보은닉은 필요하지 않은 정보는 접근할 수 없도록 하여, 한 모듈 또는 하부 시스템이 다른 모듈의 구현에 영향을 받지 않게 설계되는 것을 의미한다.

③ 단계적 정제는 기본 설계 단계에서 나타나는 프로그램의 구조에서 점차 모듈에 대한 세부 사항으로 내려가면서 구체화된다.

④ 모듈화는 시스템을 지능적으로 관리할 수 있지만 복잡도의 문제를 해결할 수는 없다.

정답 4번

출제 포인트

이 문제는 소프트웨어 설계 원리인 추상화, 정보은닉, 단계적 정제, 모듈화의 의미와 효과를 구분할 수 있는지를 묻는 문제이다.

핵심 개념

추상화는 복잡한 대상을 핵심 특성만 남겨 상위 수준에서 표현하는 설계 방법이다. 키워드는 본질만 표현, 구조화이다.
정보은닉은 모듈 내부의 구현 세부를 외부에 감추어 다른 모듈이 내부 구현에 직접 의존하지 않게 하는 원리이다. 키워드는 내부 감춤, 독립성이다.
단계적 정제는 상위 수준의 개략적 설계에서 시작해 하위 수준의 구체적 절차와 모듈 세부로 점차 내려가는 방법이다. 키워드는 상위에서 하위로 구체화이다.
모듈화는 시스템을 기능 단위로 분해해 복잡성을 낮추고 유지보수와 이해를 쉽게 만드는 원리이다. 키워드는 분할, 복잡성 관리이다.

NoteBase

선지별 해설

① O
추상화는 세부를 모두 다루지 않고 핵심 구조와 기능만 먼저 잡게 하므로 구조적이고 단계적인 설계에 도움이 된다. 설계 초기에 큰 틀을 세우는 설명으로 타당하다.

② O
정보은닉은 필요한 인터페이스만 드러내고 불필요한 구현 정보는 숨기는 원리이다. 그래서 한 모듈이나 하부 시스템의 내부 구현 변경이 다른 모듈에 미치는 영향을 줄일 수 있다.

③ O
단계적 정제는 기본 설계에서 잡은 프로그램 구조를 점차 세분화하여 각 모듈의 절차와 세부 사항으로 구체화하는 방식이다. 상위 설계에서 하위 설계로 내려간다는 설명이 맞다.

④ X
틀린 부분은 "복잡도의 문제를 해결할 수는 없다"는 표현이다. 모듈화의 핵심 목적 자체가 시스템을 여러 부분으로 나누어 복잡성을 관리 가능하게 만드는 데 있다. 올바르게는 "모듈화는 시스템을 여러 모듈로 분해하여 복잡도를 낮추고 이해와 유지보수를 쉽게 한다"가 된다.

오답 포인트

④는 모듈화의 효과를 반대로 뒤집은 선지이다. 수험생이 모듈화와 정보은닉을 모두 "좋은 설계 원리" 정도로만 외우면 ②와 ④를 함께 맞는 말처럼 볼 수 있다. 그러나 정보은닉은 내부 구현 노출 차단이 핵심이고, 모듈화는 복잡성 분해와 관리가 핵심이라는 차이를 잡아야 한다.

다시 나오면 이렇게 판단

설계 원리 문제는 각 용어의 대표 효과를 바로 연결해 판단하면 된다. 추상화는 핵심만 표현, 정보은닉은 내부 구현 감춤, 단계적 정제는 상위에서 하위로 구체화, 모듈화는 복잡성 분해로 연결되면 정답을 빠르게 고를 수 있다.

암기 포인트

추상화는 큰 틀, 정보은닉은 내부 감춤, 단계적 정제는 점진적 구체화, 모듈화는 복잡성 분해이다.

NoteBase

문 5. 상향식(bottom-up) 통합테스트에 대한 설명으로 가장 옳지 않은 것은?

① 프로그램 구조에서 가장 하위레벨에 있는 컴포넌트부터 구축하고 테스트한다.

② 하위레벨의 컴포넌트를 결합하여 특정 소프트웨어 하위기능을 수행하는 클러스터(cluster)를 만든다.

③ 테스트 케이스 입력과 출력을 조정하기 위한 드라이버(driver)를 작성한다.

④ 테스트된 컴포넌트는 아직 테스트되지 않은 컴포넌트의 호출을 위하여 스텁(stub)을 작성한다.

정답 4번

출제 포인트

이 문제는 상향식 통합테스트의 진행 방향과 보조 테스트 도구인 드라이버, 스텁의 역할 차이를 묻는 문제이다.

핵심 개념

상향식 통합테스트는 가장 하위 모듈부터 결합하여 위로 올라가며 테스트하는 방식이다.
드라이버는 아직 준비되지 않은 상위 모듈을 대신하여 하위 모듈을 호출하는 임시 프로그램이다.
스텁은 아직 구현되지 않은 하위 모듈을 대신하는 임시 모듈이므로 주로 하향식 통합테스트에서 사용된다.

선지별 해설

① O
프로그램 구조에서 가장 하위레벨에 있는 컴포넌트부터 구축하고 테스트하는 것은 상향식 통합테스트의 기본 절차에 해당한다.

② O
하위레벨 컴포넌트를 결합하여 특정 기능을 수행하는 클러스터를 만드는 방식은 상향식 통합테스트의 전형적인 통합 방법이다.

③ O
상향식 통합테스트에서는 상위 모듈이 아직 없으므로 테스트 케이스의 입력과 출력을 조정하며 하위 모듈을 호출할 드라이버를 작성한다.

④ X
틀린 부분은 "스텁(stub)"이다.
상향식 통합테스트에서는 아직 구현되지 않은 상위 컴포넌트를 대신하여 테스트된 하위 컴포넌트를 호출하기 위해 드라이버(driver)를 작성한다.
스텁은 하위 모듈의 기능을 흉내 내는 대체 모듈이므로 하향식 통합테스트와 연결된다.

NoteBase

오답 포인트

드라이버와 스텁을 모두 임시 테스트 모듈로만 외우면 방향이 뒤섞인다. 상향식은 아래에서 위로 올라가므로 상위를 대신해 하위를 호출하는 드라이버가 필요하고, 하향식은 위에서 아래로 내려가므로 하위를 대신하는 스텁이 필요하다.

다시 나오면 이렇게 판단

통합 방향을 먼저 본다. 아래에서 위로 올리면 상위 호출자가 없으므로 드라이버, 위에서 아래로 내리면 하위 피호출 모듈이 없으므로 스텁으로 판단한다.

암기 포인트

상향식은 드라이버, 하향식은 스텁이다.

NoteBase

문 6. 시스템을 릴리스하고 사용자에게 배포, 설치한 후 사용하다 보면 변경이 필요하게 된다. 변경의 이유로 가장 옳지 않은 것은?

① 운영 환경의 변화 - 하드웨어, 플랫폼, 시스템 형상의 변화로 소프트웨어 수정이 필요하다.

② 비즈니스 절차의 변화 - 비즈니스 운영 절차가 바뀌면 소프트웨어 시스템도 그에 따라서 변경하여야 한다.

③ 미래의 문제를 배제하기 위하여 변경 - 미래에 일어날 수 있는 문제를 피하기 위하여 소프트웨어 시스템에 대한 변경이 필요하다.

④ 버그 제거 - 신뢰성을 개선하기 위하여 복잡한 컴포넌트를 다시 설계하고 구현한다.

정답 4번

출제 포인트

소프트웨어 변경 이유를 수정형, 적응형, 완전형, 예방형 관점에서 구분할 수 있는지를 묻는 문제이다.

핵심 개념

수정형 변경은 이미 발견된 결함을 고치는 변경이다.
적응형 변경은 하드웨어, 운영체제, 플랫폼, 시스템 형상 등 운영 환경 변화에 맞추는 변경이다.
완전형 변경은 업무 절차나 사용자 요구 변화에 맞추어 기능이나 동작을 개선하는 변경이다.
예방형 변경은 미래 장애나 유지보수 어려움을 줄이기 위해 구조를 정리하고 복잡한 부분을 재설계하는 변경이다.

선지별 해설

① O: 하드웨어나 플랫폼이 바뀌면 기존 소프트웨어를 그 환경에 맞게 수정해야 하므로 적응형 변경 설명으로 맞다.
② O: 비즈니스 운영 절차가 바뀌면 시스템 기능이나 처리 흐름도 그 요구에 맞게 바뀌어야 하므로 업무 변화에 따른 변경 설명으로 맞다.
③ O: 미래에 발생할 수 있는 문제를 미리 줄이기 위한 변경은 예방형 변경에 해당하므로 맞다.
④ X: 틀린 부분은 '버그 제거'라는 명칭과 뒤의 설명이 서로 맞지 않는 점이다. 버그 제거는 발견된 오류를 수정하는 변경이어야 한다. 복잡한 컴포넌트를 다시 설계하고 구현하여 신뢰성이나 유지보수성을 높이는 일은 예방형 변경에 가깝다.

오답 포인트

④는 '신뢰성 개선', '복잡한 컴포넌트 재설계'라는 설명만 보면 품질 향상처럼 자연스럽게 보이지만, 핵심 함정은 앞의 명칭이 '버그 제거'라는 점이다. 결함 수정과 미래 문제 예방을 같은 범주로 섞으면 오답이 된다.

NoteBase

다시 나오면 이렇게 판단

선지의 명칭과 설명을 각각 분리해 본다. 이미 발생한 결함을 고치면 수정형, 환경 변화에 맞추면 적응형, 업무 요구 변화에 맞추면 완전형, 미래 장애를 줄이기 위한 구조 개선이면 예방형이다. 명칭과 설명의 범주가 엇갈리면 오답이다.

암기 포인트

버그는 수정형, 환경 변화는 적응형, 요구 변화는 완전형, 미래 문제 예방은 예방형이다.

NoteBase

문 7. 요구사항 타당성 검증 시의 내용 중 가장 옳지 않은 것은?

① 명확성 - 요구분석의 내용이 모호함 없이 모든 참여자들에 의해 명확하게 이해될 수 있는가?

② 일관성 - 사용자의 요구를 에러 없이 완전하게 반영하고 있는가?

③ 검증 가능성 - 요구사항 명세서에 기술된 내용이 사용자의 요구를 만족하는가?

④ 추적 가능성 - 시스템 요구사항과 시스템 설계 문서를 추적할 수 있는가?

정답 2번

출제 포인트

이 문제는 요구사항 타당성 검증 항목 중 명확성, 일관성, 검증 가능성, 추적 가능성의 의미를 구분하는 문제이다.

핵심 개념

명확성은 요구사항이 모호하지 않아 이해관계자가 같은 뜻으로 이해할 수 있는 성질이다.
일관성은 요구사항들 사이에 서로 충돌하거나 모순되는 내용이 없는 성질이다.
검증 가능성은 요구사항 명세의 내용이 실제 사용자 요구를 만족하는지 확인할 수 있는 성질이다.
추적 가능성은 요구사항이 설계, 구현, 시험 산출물과 연결되어 앞뒤로 추적될 수 있는 성질이다.

선지별 해설

① O: 명확성은 요구분석 결과가 모호함 없이 이해되어야 한다는 뜻이므로, 모든 참여자가 명확하게 이해될 수 있는가를 묻는 설명이 맞다.
② X: 틀린 부분은 "일관성"이다. "사용자의 요구를 에러 없이 완전하게 반영하고 있는가"는 완전성이나 정확성에 가까운 설명이다. 일관성은 요구사항 상호 간에 충돌이나 모순이 없는가로 고쳐야 한다.
③ O: 검증 가능성은 요구사항 명세의 내용이 실제 사용자 요구를 만족하는지 확인할 수 있어야 한다는 설명으로 볼 수 있으므로 타당하다.
④ O: 추적 가능성은 시스템 요구사항이 설계 문서 등 후속 산출물과 연결되어 추적되는 성질이므로 설명이 맞다.

오답 포인트

일관성과 완전성을 서로 바꿔 외우면 ②를 고르지 못한다. 일관성은 요구사항끼리 서로 모순이 없는가를 보는 기준이고, 완전성은 사용자 요구가 빠짐없이 반영되었는가를 보는 기준이다.

다시 나오면 이렇게 판단

요구사항 검증 용어가 나오면 먼저 대상이 이해 가능성인지, 상호 모순 여부인지, 충족 여부 확인인지, 산출물 간 연결인지 본다. 그다음 명확성, 일관성, 검증 가능성, 추적 가능성으로 대응시키면 된다.

NoteBase

암기 포인트

명확성은 이해, 일관성은 무모순, 검증 가능성은 확인 가능, 추적 가능성은 연결 추적이다.

NoteBase

문 8. 설계품질의 결합도 중 가장 바람직한 결합도에 해당하는 것은?

① 내용결합도(content coupling)

② 제어결합도(control coupling)

③ 데이터결합도(data coupling)

④ 외부결합도(external coupling)

정답 3번

출제 포인트

이 문제는 모듈 결합도의 강도 비교와 데이터결합도의 의미를 묻는 문제이다.

핵심 개념

결합도는 모듈 사이의 의존 정도이며 낮을수록 바람직하다.
내용결합도는 한 모듈이 다른 모듈의 내부 데이터나 내부 로직을 직접 참조하거나 수정하는 형태로 결합도가 매우 높다.
제어결합도는 제어 플래그나 분기 조건을 넘겨 다른 모듈의 처리 흐름에 영향을 주는 형태이다.
데이터결합도는 필요한 데이터만 매개변수로 주고받는 형태로 가장 바람직하다.
외부결합도는 장치, 통신 규약, 파일 형식 같은 외부 환경을 함께 의존하는 형태이다.

선지별 해설

① X: 내용결합도는 다른 모듈의 내부를 직접 건드리는 결합이므로 가장 바람직하지 않다. 틀린 표현은 '가장 바람직한'이다. 올바르게는 '결합도가 매우 높아 지양되는 결합도'이다.
② X: 제어결합도는 제어정보를 전달해 상대 모듈의 내부 처리 흐름을 좌우하므로 데이터결합도보다 바람직하지 않다. 틀린 표현은 '가장 바람직한'이다. 올바르게는 '제어정보 전달로 의존성이 남아 있는 결합도'이다.
③ O: 데이터결합도는 필요한 데이터만 인터페이스로 주고받고 내부 구현에는 간섭하지 않으므로 보기 중 가장 바람직하다.
④ X: 외부결합도는 공통 파일 형식, 통신 인터페이스, 장치 규격 등 외부 요소를 함께 의존하므로 순수한 데이터전달만 하는 데이터결합도보다 덜 바람직하다. 틀린 표현은 '가장 바람직한'이다. 올바르게는 '외부 환경에 함께 의존하는 결합도'이다.

오답 포인트

외부결합도를 데이터결합도와 비슷하게 보는 실수가 잦다. 그러나 외부결합도는 필요한 데이터만 주고받는 것이 아니라 외부 형식이나 장치라는 공통 환경에 묶인다는 점에서 의존성이 더 크다.

NoteBase

다시 나오면 이렇게 판단

결합도 문제는 상대 모듈의 내부를 건드리는지, 제어흐름을 좌우하는지, 외부 환경을 함께 의존하는지, 필요한 데이터만 주고받는지를 기준으로 판단한다. 필요한 데이터만 인터페이스로 전달하면 데이터결합도이고 가장 바람직하다.

암기 포인트

결합도는 낮을수록 좋고, 필요한 데이터만 주고받는 데이터결합도가 대표적인 바람직한 결합도이다.

NoteBase

문 9. Lehman의 소프트웨어 진화 법칙에 대한 설명으로 가장 옳지 않은 것은?

① 시스템이 릴리스된 후 그 시스템이 다른 시스템으로 대체될 때까지 변경된다.

② 진화하는 시스템의 유지보수 프로세스의 평균 작업량은 시스템이 소멸될 때까지 일정하다.

③ 소프트웨어 시스템의 구조는 변경될수록 복잡도가 낮아지는 경향이 있다.

④ 시스템의 평균 성장률은 소멸될 때까지 일정하다. 즉 시스템 개발의 전 단계에 걸쳐 각 버전의 변화는 거의 일정하다.

정답 3번

출제 포인트

이 문제는 Lehman의 소프트웨어 진화 법칙 중 계속적 변화, 조직 안정성 보존, 복잡도 증가, 친숙성 보존을 구별할 수 있는지를 묻는 문제이다.

핵심 개념

계속적 변화 법칙은 릴리스된 시스템이 환경에 맞추기 위해 지속적으로 수정된다는 내용이다.
조직 안정성 보존 법칙은 진화 과정의 평균 작업량이 시스템 수명 동안 거의 일정하다는 내용이다.
복잡도 증가 법칙은 시스템이 진화할수록 구조 복잡도가 커지며, 이를 낮추려면 별도의 정리와 재구성이 필요하다는 내용이다.
친숙성 보존 법칙은 개발 조직이 이해 가능한 범위 때문에 각 릴리스의 평균 성장량이 크게 변하지 않고 거의 일정하다는 내용이다.

선지별 해설

① O
시스템이 릴리스된 뒤에도 다른 시스템으로 대체될 때까지 계속 변경된다는 설명은 계속적 변화 법칙에 맞는다. 운영 환경에 적응하지 않으면 시스템의 만족도가 떨어진다고 본다.
② O
유지보수 프로세스의 평균 작업량이 시스템이 소멸될 때까지 일정하다는 설명은 조직 안정성 보존 법칙에 맞는다. 진화 시스템에 투입되는 평균 활동량은 장기간 거의 일정하게 유지된다고 본다.
③ X
틀린 표현은 "변경될수록 복잡도가 낮아지는"이다. Lehman의 복잡도 증가 법칙은 변경이 누적될수록 소프트웨어 구조와 상호의존성이 더 복잡해지는 경향이 있다고 본다. 올바르게는 "변경될수록 복잡도가 증가하는 경향이 있으며, 이를 줄이려면 의도적인 정리 작업이 필요하다"이다.
④ O
시스템의 평균 성장률이 수명 동안 일정하다는 설명은 친숙성 보존 법칙의 취지에 맞는다. 각 버전의 변화량은 개발 조직이 이해하고 통제할 수 있는 범위 때문에 거의 일정하게 유지된다.

NoteBase

오답 포인트

③의 "복잡도가 낮아지는"과 ④의 "성장률은 일정하다"를 모두 변화가 안정된다는 말로 뭉뚱그리면 오답이 된다. Lehman 문제는 무엇이 증가하는지와 무엇이 일정한지를 분리해서 읽어야 한다.

다시 나오면 이렇게 판단

Lehman 법칙 문제는 먼저 대상 축을 나누면 된다. 시스템은 릴리스 후에도 계속 변하고, 평균 작업량과 평균 성장량은 거의 일정하며, 복잡도는 관리하지 않으면 증가한다고 판단하면 된다.

암기 포인트

릴리스 후 계속 변화, 일량은 일정, 복잡도는 증가, 성장량은 일정이다.

NoteBase

문 10. <보기>는 A, B회사의 월 평균 생산성과 1인당 월 평균 임금을 보여준다. 두 회사가 각각 49,300 LOC(Line of Code) 시스템 개발에 참여할 경우에 대한 설명으로 가장 옳지 않은 것은? (단, LOC 외의 다른 비용은 고려하지 않는다.)

<보기>

회사명 / 회사 정보	A	B
월 평균 생산성 LOC/MM(man month)	725	580
1인당 월 평균 임금	410만 원	320만 원

① A 회사의 노력(man month)은 68MM이다.

② B 회사의 노력(man month)은 85MM이다.

③ A 회사가 B 회사보다 프로젝트 총 비용이 적게 든다.

④ B 회사의 프로젝트 총 비용은 27,200만 원이다.

정답 3번

출제 포인트

이 문제는 생산성 *LOC/MM*으로 노력 *MM*을 구하고, 노력에 1인당 월 평균 임금을 곱해 총 비용을 비교하는 문제이다.

조건 정리

주어진 값:
- 총 개발량: 49,300 LOC
- A 회사 생산성: 725 LOC/MM, 1인당 월 평균 임금: 410만 원
- B 회사 생산성: 580 LOC/MM, 1인당 월 평균 임금: 320만 원

구해야 할 값:
- 각 회사의 노력 *MM*
- 각 회사의 프로젝트 총 비용

주의할 조건:
- LOC 외의 다른 비용은 고려하지 않는다.

NoteBase

사용할 공식

노력 *MM* = 총 LOC / 생산성 *LOC/MM*
총 비용 = 노력 *MM* × 1인당 월 평균 임금

이 공식을 쓰는 이유:
생산성이 한 사람의 한 달 작업량 *LOC/MM*으로 주어졌으므로 전체 LOC를 완성하는 데 필요한 총 인월 *MM*을 먼저 구해야 하고, 여기에 월 평균 임금을 곱하면 총 인건비를 계산할 수 있다.

계산 과정

1. A 회사의 노력은 49, 300/725 = 68이므로 68MM이다.
2. B 회사의 노력은 49, 300/580 = 85이므로 85MM이다.
3. A 회사의 총 비용은 68 × 410 = 27, 880이므로 27,880만 원이다.
4. B 회사의 총 비용은 85 × 320 = 27, 200이므로 27,200만 원이다.

선지별 해설

① O
A 회사의 노력은 49, 300/725 = 68이므로 맞는 설명이다.

② O
B 회사의 노력은 49, 300/580 = 85이므로 맞는 설명이다.

③ X
A 회사 총 비용은 27,880만 원이고 B 회사 총 비용은 27,200만 원이므로 A 회사가 더 적게 든다는 설명은 틀리다. 틀린 표현은 "A 회사가 B 회사보다 프로젝트 총 비용이 적게 든다"이고, 올바르게는 "A 회사가 B 회사보다 프로젝트 총 비용이 더 많이 든다"이다.

④ O
B 회사의 총 비용은 85 × 320 = 27, 200만 원이므로 맞는 설명이다.

오답 포인트

생산성이 높은 A 회사가 무조건 총비용도 더 낮을 것이라고 단정하면 ③에 속기 쉽다. 이 문항은 생산성과 임금을 함께 봐야 하며, 노력이 적더라도 월 임금이 더 높으면 총비용이 커질 수 있다는 점을 묻는다.

다시 나오면 이렇게 판단

계산형 비용 비교 문제는 총 LOC에서 바로 비용을 비교하지 않고, 먼저 생산성으로 노력 *MM*을 구한 뒤 그 노력에 월 평균 임금을 곱해 총비용을 비교하는 순서로 판단한다.

NoteBase

암기 포인트

생산성으로 인월을 구하고, 인월에 월 임금을 곱하면 총비용이 된다.

NoteBase

문 11. 나선형(spiral) 모델의 특징으로 가장 옳지 않은 것은?

① 반복 주기가 시작될 때 소프트웨어의 목표와 제약조건을 결정한다.

② 한 번의 개발 주기를 거치면서 시스템이 완성된다.

③ 프로토타입을 만들면서 위험을 분석한다.

④ 개발을 위한 계획 및 요구분석 후에 위험 요소에 대하여 검토한다.

정답 2번

출제 포인트

나선형 모델의 핵심인 반복적 개발, 위험 분석, 프로토타입 활용, 그리고 한 번에 완성되지 않고 점진적으로 발전한다는 점을 구별하는 문제이다.

핵심 개념

나선형 모델은 위험 분석을 중심에 두고 여러 번의 반복 주기를 돌면서 시스템을 점진적으로 완성하는 개발 모형이다.
폭포수 모델은 단계를 순차적으로 한 번씩 크게 진행하는 성격이 강하고, 프로토타입 모델은 시제품으로 요구사항을 구체화하는 데 초점이 있다.
나선형 모델은 이 둘의 장점을 일부 활용하되, 가장 두드러진 구별점은 반복 주기마다 위험을 식별하고 줄인다는 점이다.

NoteBase

선지별 해설

① O
반복 주기의 시작에서 목표, 대안, 제약 조건을 정하는 것은 나선형 모델의 첫 단계에 해당한다. 각 사이클은 무엇을 만들 것인지와 어떤 제약 아래 진행할 것인지를 먼저 정리한다.

② X
틀린 부분은 "한 번의 개발 주기"이다.
나선형 모델은 한 번의 주기로 시스템이 완성되는 모형이 아니라 여러 번의 반복 주기를 거치며 점차 완성도를 높이는 모형이다. 올바르게는 "여러 번의 개발 주기를 거치면서 시스템이 점진적으로 완성된다"이다.

③ O
나선형 모델은 위험 분석을 매우 중시하며, 위험을 줄이기 위한 수단으로 프로토타입을 활용할 수 있다. 따라서 프로토타입을 만들면서 위험을 분석한다는 설명은 맞다.

④ O
나선형 모델에서는 계획 수립과 요구의 확인 뒤에 위험 요소를 검토하고 분석하는 단계가 이어진다. 위험 분석이 각 반복 주기의 핵심 활동이라는 점에서 옳은 설명이다.

오답 포인트

②의 "한 번의 개발 주기"라는 표현이 함정이다. 나선형 모델은 반복과 점진적 완성이 본질인데, 이를 폭포수 모델처럼 한 번의 흐름으로 끝나는 것으로 바꾸면 바로 틀린 선지가 된다.

다시 나오면 이렇게 판단

선지에 위험 분석, 반복 주기, 프로토타입 활용이 함께 나오면 나선형 모델의 특징으로 본다. 반대로 한 번에 완성된다거나 단일 순차 흐름만 강조하면 나선형 모델의 설명으로 보기 어렵다.

암기 포인트

나선형 모델은 반복 + 위험 분석 + 프로토타입이며, 1회 주기로 완성되는 모형은 아니다.

NoteBase

문 12. 스크럼(scrum) 방법론에서 SRS(Software Requirement Specification)나 TRS(Technical Requirement Specification)에 해당하는 목록은?

① 제품 백로그(product backlog)

② 스프린트(Sprint)

③ 스크럼 마스터(Scrum master)

④ 스프린트 트래킹(Sprint tracking)

정답 1번

출제 포인트

이 문제는 스크럼에서 전통적 요구사항 명세서 역할을 어떤 산출물이 대신하는지를 묻는다.

핵심 개념

제품 백로그는 제품에 필요한 기능과 요구사항을 우선순위대로 정리한 목록이다. 스크럼은 SRS나 TRS 같은 상세 명세서를 고정 문서로 두기보다 제품 백로그를 계속 갱신하며 요구사항을 관리한다.
스프린트는 고정된 기간의 개발 반복이다. 스크럼 마스터는 프로세스를 촉진하는 역할이다. 스프린트 트래킹은 진행 상황을 추적하는 활동이지 요구사항 명세 산출물이 아니다.

선지별 해설

1번 O: 제품 백로그는 구현할 요구사항을 우선순위와 함께 정리한 목록이므로 스크럼에서 SRS/TRS에 가장 가까운 요구사항 관리 산출물이다.
2번 X: 틀린 부분은 '스프린트'이다. 스프린트는 작업 기간 단위이지 요구사항 명세 목록이 아니다. 올바르게는 요구사항을 담는 산출물은 제품 백로그이다.
3번 X: 틀린 부분은 '스크럼 마스터'이다. 스크럼 마스터는 팀이 스크럼을 제대로 수행하도록 돕는 역할이다. 올바르게는 요구사항 명세 역할에 가까운 것은 제품 백로그이다.
4번 X: 틀린 부분은 '스프린트 트래킹'이다. 이는 진행 상황을 점검하는 관리 활동 또는 추적 방식이지 요구사항을 정의한 명세 산출물이 아니다. 올바르게는 요구사항 목록은 제품 백로그가 담당한다.

오답 포인트

스프린트, 스크럼 마스터, 트래킹처럼 스크럼에서 자주 보이는 용어를 모두 산출물로 착각하면 틀린다. 이 문항은 역할, 이벤트, 관리활동, 요구사항 산출물을 구분하는지를 노린다.

다시 나오면 이렇게 판단

선지가 문서인지 역할인지 이벤트인지부터 구분한 뒤, 요구사항을 목록으로 관리하는 산출물이 무엇인지 연결하면 된다. 요구사항의 저장소는 제품 백로그, 기간 단위는 스프린트, 역할은 스크럼 마스터이다.

NoteBase

암기 포인트

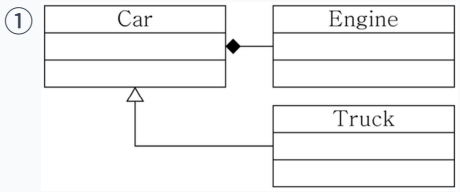
스크럼의 요구사항 목록은 제품 백로그, 개발 기간은 스프린트, 운영 지원 역할은 스크럼 마스터이다.

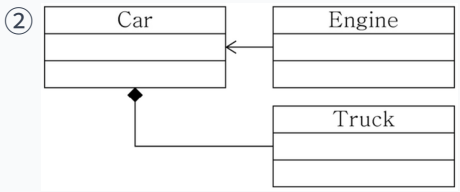
NoteBase

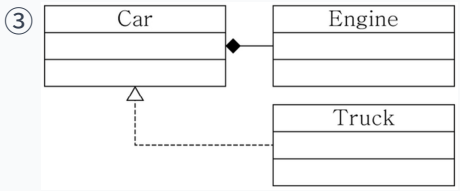
문 13. <보기>의 자바 소스코드를 UML로 표현했을 때 가장 옳은 것은?

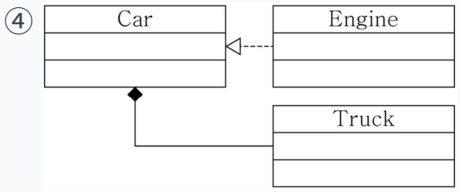
<보기>

```
Class Engine {
    …
}
Class Car {
    Engine carEngine = new Engine();
    …
}
Class Truck extends Car{
    …
}
```

① 

② 

③ 

④ 

정답 1번

NoteBase

출제 포인트

이 문제는 자바 코드의 `has-a` 관계와 `extends` 관계를 UML에서 각각 컴포지션과 일반화로 정확히 읽어낼 수 있는지를 묻는다.

선지별 해설

1번: O. `Car` 클래스 안에 `Engine carEngine = new Engine();`가 있으므로 `Car`가 `Engine` 객체를 직접 생성하고 강하게 소유하는 컴포지션 관계로 보는 것이 맞다. 따라서 `Car` 쪽의 검은 마름모가 맞고, `Truck extends Car`는 `Truck`이 `Car`를 상속하므로 `Truck`에서 `Car`로 향하는 실선과 빈 삼각형의 일반화 표기도 맞다.
2번: X. `Engine`에서 `Car`로 향하는 화살표는 방향성 연관처럼 보일 뿐 `Car`가 `Engine`을 포함해 생성하는 관계를 나타내지 못한다. 또 `Car`와 `Truck` 사이의 검은 마름모는 상속이 아니라 합성 관계가 된다. 올바르게는 `Car`와 `Engine` 사이를 `Car` 쪽 검은 마름모의 컴포지션으로, `Truck`과 `Car` 사이는 실선 빈 삼각형의 일반화로 표현한다.
3번: X. `Car`와 `Engine`의 컴포지션 표시는 맞지만 `Truck`에서 `Car`로 가는 점선과 빈 삼각형은 클래스 상속이 아니라 인터페이스 실체화에 쓰는 표기이다. 올바르게는 점선이 아니라 실선 일반화여야 한다.
4번: X. `Engine`에서 `Car`로 향하는 점선과 빈 삼각형은 `Car`가 `Engine`을 포함한다는 뜻이 아니며 실체화 또는 잘못된 의존 표기에 가깝다. 또한 `Car`와 `Truck` 사이의 검은 마름모도 상속을 합성으로 잘못 그린 것이다. 올바르게는 `Car`-`Engine`은 `Car` 쪽 검은 마름모의 컴포지션, `Truck`-`Car`는 실선과 빈 삼각형의 일반화이다.

오답 포인트

검은 마름모만 보면 무조건 포함 관계라고 외우고, 빈 삼각형만 보면 실선인지 점선인지 구분하지 않으면 3번에 쉽게 끌린다. UML에서는 검은 마름모는 컴포지션, 실선 빈 삼각형은 클래스 상속, 점선 빈 삼각형은 인터페이스 실체화라는 구분이 핵심이다.

다시 나오면 이렇게 판단

코드에 `extends`가 보이면 먼저 실선 빈 삼각형의 일반화를 찾고, 멤버 필드로 다른 객체를 직접 생성·보유하면 그다음 검은 마름모의 컴포지션인지 확인하면 된다.

암기 포인트

`extends`는 실선 빈 삼각형, 내부에서 `new`로 강하게 포함하면 검은 마름모 컴포지션이다.

NoteBase

문 14. ISO/IEC 9126의 6가지 품질 특성 중 신뢰성(reliability)을 구성하는 하위 특성에 대한 설명으로 가장 옳은 것은?

① 적응성(adaptability)은 소프트웨어가 설치되어 있는 운영체제나 미들웨어 또는 하드웨어 환경에서 정상적으로 잘 작동할 수 있는 능력을 말한다.

② 안정성(stability)은 소프트웨어 변경으로 인한 예상치 못한 결과를 최소화하는 능력을 말한다.

③ 정확성(accuracy)은 사용자가 요구하는 정밀도를 유지하거나 허용 범위 내의 결과값을 제공할 수 있는 능력을 말한다.

④ 결함 수용성(fault tolerance)은 소프트웨어 일부에서 고장이 발생해도 요구되는 기능을 유지할 수 있는지를 나타낸다.

정답 4번

출제 포인트

이 문제는 ISO/IEC 9126의 품질 특성과 하위 특성을 짝지어 구분할 수 있는지를 묻는 문제이다.

핵심 개념

신뢰성은 주어진 조건에서 일정 시간 동안 요구 기능을 안정적으로 수행하는 품질 특성이다. 대표 하위 특성은 성숙성, 결함 수용성, 회복성이다.
적응성은 다른 운영환경에 맞게 소프트웨어를 조정하여 사용할 수 있는 성질로 이식성의 하위 특성이다.
안정성은 수정 후 예상치 못한 부작용을 적게 일으키는 성질로 유지보수성의 하위 특성이다.
정확성은 요구된 정밀도와 올바른 결과를 제공하는 성질로 기능성의 하위 특성이다.

NoteBase

선지별 해설

① X
적응성은 신뢰성이 아니라 이식성의 하위 특성이다. 선지의 설명도 특정 운영체제나 하드웨어 환경에 맞게 동작하도록 조정될 수 있는 성질 쪽에 가깝다. 틀린 표현은 "신뢰성을 구성하는 하위 특성"에 해당하는 것처럼 제시된 부분이다. 올바르게는 "적응성은 이식성의 하위 특성이다"이다.

② X
안정성의 설명 자체는 소프트웨어 변경으로 인한 예상치 못한 결과를 최소화하는 능력이라는 점에서 대체로 맞다. 그러나 안정성은 신뢰성의 하위 특성이 아니라 유지보수성의 하위 특성이다. 틀린 표현은 "신뢰성의 하위 특성"에 포함되는 것처럼 제시된 점이다. 올바르게는 "안정성은 유지보수성의 하위 특성이다"이다.

③ X
정확성의 설명은 사용자가 요구하는 정밀도를 유지하거나 허용 범위 내의 결과값을 제공하는 능력이라는 점에서 맞다. 그러나 정확성은 신뢰성의 하위 특성이 아니라 기능성의 하위 특성이다. 틀린 표현은 "신뢰성의 하위 특성"에 해당하는 것처럼 제시된 점이다. 올바르게는 "정확성은 기능성의 하위 특성이다"이다.

④ O
결함 수용성은 신뢰성의 하위 특성이다. 시스템 일부에 결함이 발생해도 전체가 즉시 중단되지 않고 요구 기능을 계속 수행할 수 있는 능력을 뜻하므로 선지의 설명이 맞다.

오답 포인트

이 문제의 함정은 각 선지의 정의 자체는 상당 부분 맞게 써 두고, 그것이 어느 상위 품질 특성에 속하는지만 바꾸어 놓은 점이다. 특히 ②의 안정성과 ③의 정확성은 정의가 익숙해서 그대로 정답처럼 보기 쉽지만, 상위 분류가 각각 유지보수성과 기능성이라는 점이 핵심이다.

다시 나오면 이렇게 판단

ISO/IEC 9126 문제는 선지의 정의가 맞는지부터 보지 말고 먼저 상위 품질 특성과 하위 특성의 소속을 짝으로 확인하면 판단이 빨라진다. 신뢰성은 성숙성, 결함 수용성, 회복성처럼 장애 상황에서도 기능을 유지하거나 복구하는 개념인지로 판별할 수 있다.

암기 포인트

신뢰성은 장애 대응, 유지보수성은 변경 안정, 기능성은 결과 정확, 이식성은 환경 적응으로 구분된다.

NoteBase

문 15. <보기>는 CMMI의 성숙도(maturity) 5단계의 내용을 요약하고 있다. 이들을 초기 단계부터 최적화 단계까지 순서대로 바르게 나열한 것은?

<보기>
(가) 기본적인 프로젝트 관리 체계 수립
(나) 프로세스 최적화
(다) 조직 차원의 표준 프로세스를 통한 프로젝트 지원
(라) 정량적으로 프로세스가 측정/통제됨
(마) 예측/통제 불가능

① (라) - (다) - (가) - (마) - (나)
② (라) - (가) - (다) - (마) - (나)
③ (마) - (다) - (가) - (라) - (나)
④ (마) - (가) - (다) - (라) - (나)

정답 4번

출제 포인트

CMMI 성숙도 5단계의 핵심 특징을 초기 단계부터 최적화 단계까지 올바른 순서로 연결하는지를 묻는 문제이다.

올바른 순서

1. (마) 예측/통제 불가능: 초기(Initial) 단계로 프로세스가 불안정하고 예측이 어렵다.
2. (가) 기본적인 프로젝트 관리 체계 수립: 관리(Managed) 단계로 일정, 비용, 요구사항 같은 기본 관리가 자리 잡는다.
3. (다) 조직 차원의 표준 프로세스를 통한 프로젝트 지원: 정의(Defined) 단계로 조직 표준 프로세스가 정착한다.
4. (라) 정량적으로 프로세스가 측정/통제됨: 정량적 관리(Quantitatively Managed) 단계로 수치 기반 관리가 이루어진다.
5. (나) 프로세스 최적화: 최적화(Optimizing) 단계로 지속적 개선이 이루어진다.

왜 이 순서인가: 기본 프로젝트 관리 체계가 먼저 잡혀야 조직 표준 프로세스로 확장할 수 있고, 표준화가 끝나야 그 프로세스를 정량적으로 측정·통제한 뒤 최적화할 수 있다.

NoteBase

선지별 해설

① X
(라)를 첫 단계에 둔 점이 틀렸다. 정량적으로 측정·통제되는 상태는 4단계의 특징이므로 초기 단계에 올 수 없다.
틀린 표현 → 올바르게는 (마) - (가) - (다) - (라) - (나)이다.

② X
①과 마찬가지로 (라)를 맨 앞에 둔 점이 틀렸다. 기본적인 프로젝트 관리 체계 수립도 정량 관리보다 앞서지만, 그보다 먼저 초기 단계의 불안정 상태인 (마)가 와야 한다.
틀린 표현 → 올바르게는 (마) - (가) - (다) - (라) - (나)이다.

③ X
(다)와 (가)의 위치가 바뀐 점이 틀렸다. 조직 차원의 표준 프로세스 지원은 기본적인 프로젝트 관리 체계가 수립된 다음 단계이다.
틀린 표현 → 올바르게는 (마) - (가) - (다) - (라) - (나)이다.

④ O
(마) - (가) - (다) - (라) - (나)는 Initial → Managed → Defined → Quantitatively Managed → Optimizing의 순서를 정확히 반영한 배열이다.

오답 포인트

2단계 Managed와 3단계 Defined를 서로 바꾸거나, 4단계 Quantitatively Managed를 단순한 '관리' 단계로 착각하는 함정이 자주 나온다. CMMI는 기본 관리가 먼저이고, 정량 관리는 표준화 이후의 고도화 단계이다.

다시 나오면 이렇게 판단

CMMI는 예측 불가능한 초기 상태에서 출발해 기본 관리, 조직 표준화, 정량 관리, 지속적 최적화로 올라간다고 보면 된다. '기본 관리'는 2단계, '정량적 관리'는 4단계, '최적화'는 5단계로 끊어 판단하면 순서가 정리된다.

암기 포인트

Initial → Managed → Defined → Quantitatively Managed → Optimizing은 예측불가 → 기본관리 → 표준화 → 수치관리 → 개선의 흐름이다.

NoteBase

문 16. <보기>와 같은 특징을 갖는 테스트 기법은?

<보기>
• 프로그램의 소스코드를 분석해 결함을 찾아내는 테스트 기법
• 시스템의 실행을 동반하지 않음
• 무한루프, 초기화되지 않은 변수, 실행 불가능한 코드 등을 발견할 수 있음
• 소스코드의 잠재적 오류를 찾아냄으로써 디버깅에 유용한 정보 생성

① 정적 분석(static analysis)
② 단위 테스트(unit test)
③ 스모크 테스트(smoke test)
④ 코드 스멜(code smell)

정답 1번

출제 포인트

이 문제는 소스코드를 실행하지 않고 코드 구조와 규칙 위반을 분석해 결함을 찾는 정적 분석의 특징을 묻는다.

핵심 개념

정적 분석은 프로그램을 실행하지 않은 상태에서 소스코드나 중간 코드를 검사해 결함 가능성을 찾는 기법이다. 단위 테스트와 스모크 테스트는 실제 코드를 실행해 동작을 확인하는 동적 테스트이다. 코드 스멜은 테스트 기법이 아니라 설계나 구현에 개선 신호가 보이는 상태를 뜻한다.

선지별 해설

1번: O. 정적 분석은 프로그램을 실행하지 않고 소스코드를 분석해 초기화되지 않은 변수, 실행 불가능한 코드, 잠재적 결함을 찾는 기법이므로 보기의 설명과 일치한다.
2번: X. 틀린 표현은 단위 테스트를 실행이 없는 분석 기법처럼 보는 점이다. 단위 테스트는 함수나 모듈을 실제로 실행해 기대 결과가 나오는지 검증하는 동적 테스트이다.
3번: X. 틀린 표현은 스모크 테스트가 소스코드 분석만으로 결함을 찾는다는 점이다. 스모크 테스트는 빌드 후 핵심 기능이 큰 문제 없이 동작하는지 빠르게 확인하는 실행 기반 테스트이다.
4번: X. 틀린 표현은 코드 스멜을 테스트 기법으로 보는 점이다. 코드 스멜은 중복, 긴 메서드, 과도한 결합도처럼 리팩터링이 필요할 가능성을 보여 주는 징후이지 정적 분석 자체를 가리키는 용어는 아니다.

NoteBase

오답 포인트

정적 분석과 코드 스멜은 모두 소스코드 품질과 관련되어 헷갈리기 쉽다. 그러나 정적 분석은 결함을 찾는 기법이고, 코드 스멜은 코드에 나타나는 이상 징후라는 점이 다르다. 또한 단위 테스트와 스모크 테스트는 모두 실제 실행을 동반하므로 보기와 바로 구별된다.

다시 나오면 이렇게 판단

선지를 볼 때 먼저 실행 여부를 확인한다. 실행 없이 코드 규칙, 흐름, 변수 사용을 검사하면 정적 분석이고, 실제로 프로그램이나 모듈을 돌려 보면 동적 테스트이다. 기법이 아니라 상태나 징후를 말하면 테스트 종류가 아닐 가능성이 크다.

암기 포인트

정적 분석은 실행 없이 코드 검사, 단위·스모크 테스트는 실행 기반, 코드 스멜은 결함 징후이다.

NoteBase

문 17. GoF(Gang of Four)가 제시한 디자인 패턴에 대한 설명으로 가장 옳지 않은 것은?

① Factory Method - 객체를 생성하는 처리를 파생 클래스로 분리하여 처리하도록 캡슐화하는 패턴

② Adapter - 한 클래스의 인터페이스를 클라이언트에서 필요로 하는 인터페이스로 변환해주는 패턴

③ State - 객체의 상태에 따라 객체의 행위 내용을 변경해주는 패턴

④ Strategy - 이미 고정된 자료구조에 행위를 쉽게 추가할 수 있도록 해주는 패턴

정답 4번

출제 포인트

이 문제는 GoF 디자인 패턴 중 Factory Method, Adapter, State, Strategy의 핵심 목적을 정확히 구별하는지를 묻는 문제이다.

핵심 개념

Factory Method는 객체 생성 인터페이스는 상위 타입에 두고 실제 생성은 하위 클래스에 위임하는 생성 패턴이다. 키워드는 생성 위임이다.
Adapter는 기존 클래스의 인터페이스를 클라이언트가 원하는 인터페이스로 바꾸는 구조 패턴이다. 키워드는 인터페이스 변환이다.
State는 상태를 객체로 분리하여 내부 상태 변화에 따라 동작이 달라지게 하는 행위 패턴이다. 키워드는 상태별 행위 변화이다.
Strategy는 알고리즘이나 행위 방식을 캡슐화하여 서로 교체할 수 있게 하는 행위 패턴이다. 키워드는 알고리즘 교체이다.

NoteBase

선지별 해설

① O

Factory Method는 객체 생성 로직을 서브클래스가 맡도록 하여 생성 책임을 분리하는 패턴이다. "객체를 생성하는 처리를 파생 클래스로 분리"한다는 설명이 핵심 목적과 맞는다.

② O

Adapter는 기존 클래스의 인터페이스를 클라이언트가 기대하는 인터페이스로 바꾸어 연결해 주는 패턴이다. "필요로 하는 인터페이스로 변환"해 준다는 설명이 정확하다.

③ O

State는 객체의 상태가 바뀌면 같은 요청에 대한 동작도 달라지도록 상태별 행위를 분리하는 패턴이다. "객체의 상태에 따라 객체의 행위 내용을 변경"한다는 설명이 맞다.

④ X

틀린 부분은 "이미 고정된 자료구조에 행위를 쉽게 추가"라는 표현이다. 이는 Strategy가 아니라 Visitor의 설명에 가깝다. Strategy는 여러 알고리즘을 캡슐화하고 필요에 따라 교체할 수 있게 하는 패턴이다.

오답 포인트

Strategy와 Visitor는 모두 행위를 분리한다는 점 때문에 혼동하기 쉽다. 그러나 Strategy는 알고리즘을 바꿔 끼우는 데 초점이 있고, Visitor는 객체 구조는 유지한 채 새로운 연산을 추가하는 데 초점이 있다.

다시 나오면 이렇게 판단

설명에 인터페이스 변환이 나오면 Adapter, 생성 책임 위임이 나오면 Factory Method, 상태 변화에 따른 동작 변화가 나오면 State로 본다. 알고리즘 교체가 나오면 Strategy이고, 구조는 그대로 둔 채 연산 추가가 나오면 Visitor로 판단한다.

암기 포인트

Factory Method는 생성 위임, Adapter는 인터페이스 변환, State는 상태별 동작 변화, Strategy는 알고리즘 교체, Visitor는 연산 추가이다.

NoteBase

문 18. ISO 품질 평가 표준에 대한 설명으로 가장 옳지 않은 것은?

① ISO/IEC 9126 : 소프트웨어 품질의 특성을 정의하고 품질평가 메트릭을 정의

② ISO/IEC 12207 : 설계, 생산, 설치 및 서비스 과정에 대한 품질보증 모델

③ ISO/IEC 14598 : 소프트웨어 제품의 품질을 측정하거나 평가하기 위한 방법과 절차를 규정

④ ISO/IEC 15504 : 소프트웨어 프로세스를 평가하고 개선함으로써 품질 및 생산성을 높이기 위한 표준

정답 2번

출제 포인트

이 문제는 ISO/IEC 9126, 12207, 14598, 15504가 각각 제품 품질 모델, 생명주기 프로세스, 제품 평가 절차, 프로세스 평가 중 무엇에 해당하는지를 구분하는 문제이다.

핵심 개념

ISO/IEC 9126은 소프트웨어 제품의 품질 특성과 품질 모델을 정의한 표준이다.
ISO/IEC 12207은 소프트웨어의 획득, 개발, 운영, 유지보수 등 생명주기 프로세스를 정의한 표준이다.
ISO/IEC 14598은 소프트웨어 제품 품질을 평가하는 절차와 방법을 규정한 표준이다.
ISO/IEC 15504는 소프트웨어 프로세스의 수행능력과 성숙도를 평가하고 개선하는 표준이다.

선지별 해설

① O
ISO/IEC 9126은 소프트웨어 품질 특성을 정의하고 품질평가 메트릭 체계를 제시한 표준이므로 옳다.

② X
틀린 부분은 "설계, 생산, 설치 및 서비스 과정에 대한 품질보증 모델"이다.
ISO/IEC 12207은 품질보증 모델이 아니라 소프트웨어 생명주기 프로세스를 정의한 표준이다.
설계·생산·설치·서비스에 대한 품질보증 모델이라는 설명은 ISO 9001 계열에 해당한다.

③ O
ISO/IEC 14598은 소프트웨어 제품의 품질을 측정하거나 평가하기 위한 방법과 절차를 규정한 표준이므로 옳다.

④ O
ISO/IEC 15504는 소프트웨어 프로세스를 평가하고 개선하기 위한 표준이므로 옳다.
프로세스 개선을 통해 품질과 생산성 향상을 도모한다는 설명도 맞다.

NoteBase

오답 포인트

12207과 15504는 모두 프로세스와 관련되어 보여 서로 바꾸기 쉽다. 그러나 12207은 프로세스의 종류와 구성 자체를 정의하는 표준이고, 15504는 그 프로세스를 평가하고 개선하는 표준이라는 점이 핵심 함정이다.

다시 나오면 이렇게 판단

표준 번호가 나오면 먼저 대상이 제품 품질 특성인지, 소프트웨어 생명주기인지, 제품 평가 절차인지, 프로세스 평가인지로 나누면 된다. 품질보증 모델이라는 표현이 보이면 12207보다는 ISO 9000 계열 설명인지부터 점검하면 된다.

암기 포인트

9126은 품질특성, 12207은 생명주기, 14598은 제품평가, 15504는 프로세스평가이다.

NoteBase

문 19. 소프트웨어 아키텍처 스타일에 대한 설명으로 가장 옳지 않은 것은?

① 계층형(layering) : 시스템을 기능에 따라 수직적인 계층으로 구분하고, 각 계층은 서비스의 집합을 제공

② MVC(Model-View-Controller) : 사용자 인터페이스와 데이터 처리 로직을 독립적으로 분리하여 변경에 대한 영향을 줄임

③ 파이프 필터(pipe and filter) : 프로그램에서 감지되고 처리될 수 있는 사건(event) 중심의 시스템에 적합한 아키텍처

④ 데이터 중심형(저장소 구조)(repository) : 주요 데이터가 중앙에서 관리되고 서브시스템이 공유데이터에 접근해 정보를 사용

정답 3번

출제 포인트

이 문제는 대표적인 소프트웨어 아키텍처 스타일의 핵심 구조와 적용 대상을 구별하는지를 묻는 문제이다.

핵심 개념

계층형은 기능을 층으로 나누고 상위 계층에 서비스를 제공하는 구조이다.
MVC는 모델, 뷰, 컨트롤러를 분리해 화면과 처리 로직의 결합을 낮추는 구조이다.
파이프-필터형은 데이터가 여러 처리 단계를 순차적으로 흐르며 변환되는 구조이다.
데이터 중심형은 중앙 저장소를 여러 서브시스템이 공유하며 사용하는 구조이다.

NoteBase

선지별 해설

① O
계층형은 시스템을 여러 계층으로 분리하고 각 계층이 인접 계층에 서비스를 제공하는 구조이므로 설명이 타당하다.

② O
MVC는 사용자 인터페이스와 데이터 처리 로직을 분리해 한쪽 변경이 다른 쪽에 미치는 영향을 줄이는 구조이므로 설명이 맞다.

③ X
틀린 부분은 "사건(event) 중심의 시스템에 적합한 아키텍처"라는 표현이다.
파이프-필터형은 이벤트 중심이 아니라 데이터 흐름 중심 구조이다. 독립된 필터가 입력 데이터를 단계적으로 변환하고, 파이프가 그 결과를 다음 필터로 전달한다.
올바르게는 "파이프-필터형은 데이터 스트림을 연속된 처리 단계로 변환하는 시스템에 적합한 아키텍처"이다.
이벤트 중심 시스템에 적합한 것은 이벤트 기반 또는 암시적 호출 구조이다.

④ O
데이터 중심형 저장소 구조는 중앙 저장소에 핵심 데이터를 두고 여러 서브시스템이 공유 데이터에 접근해 사용하는 방식이므로 설명이 옳다.

오답 포인트

파이프-필터형의 "흐름"과 이벤트 기반 구조의 "사건 발생"을 모두 비동기 처리처럼 받아들이면 ③을 맞는 설명으로 착각하기 쉽다. 이 문항은 데이터가 단계적으로 전달되는 구조인지, 사건 발생에 반응하는 구조인지를 구분하는지가 핵심 함정이다.

다시 나오면 이렇게 판단

선지에 데이터가 순차적으로 가공되고 전달된다는 뜻이 있으면 파이프-필터형으로 판단한다. 사건 발생, 알림, 핸들러, 반응 같은 표현이 나오면 이벤트 기반 구조를 먼저 떠올리면 된다.

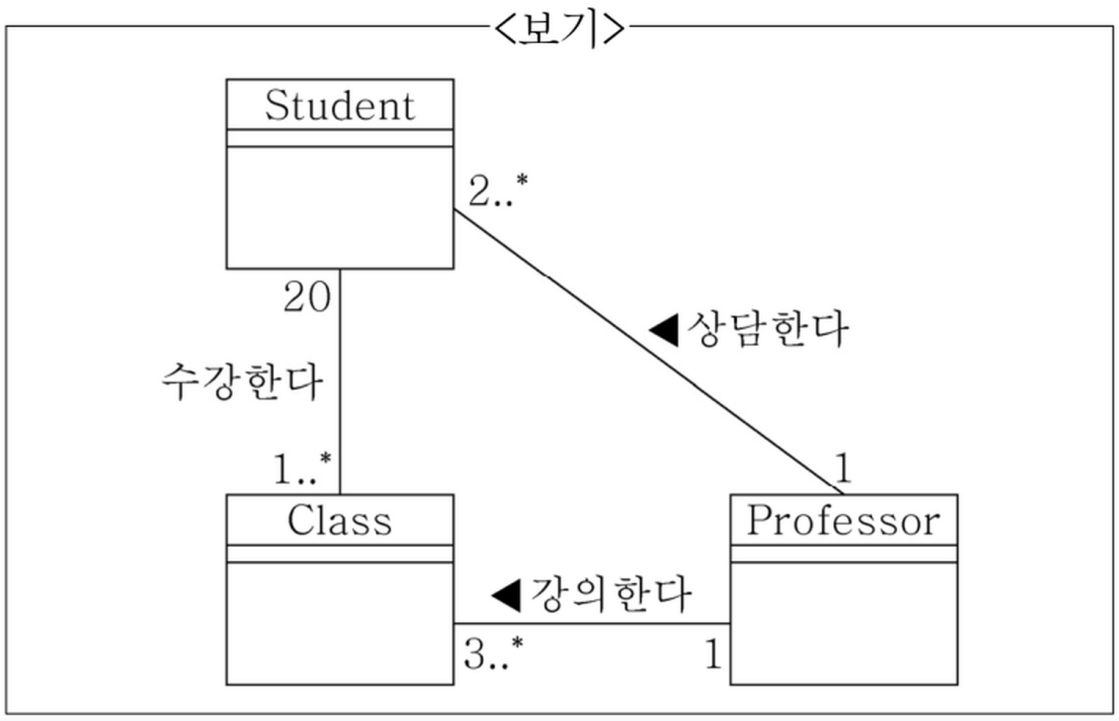
암기 포인트

파이프-필터형은 데이터 흐름, 이벤트 기반은 사건 반응, 저장소 구조는 중앙 데이터 공유이다.

NoteBase

문 20. <보기>의 클래스(class) 다이어그램에 대한 설명으로 가장 옳은 것은?

<보기>



① 'Professor'는 1개 이상의 'Class'를 강의한다.

② 'Class'를 20명 이상의 'Student'가 수강한다.

③ 'Professor'는 2명 이상의 'Student'를 상담한다.

④ 'Student'는 20개의 'Class'를 수강한다.

정답 3번

출제 포인트

이 문제는 클래스 다이어그램의 연관 다중성 표기를 읽어 각 객체가 다른 객체와 몇 개까지 연결되는지를 정확히 해석할 수 있는지를 묻는다.

NoteBase

선지별 해설

① X. Professor-Class 연관에서 Class 쪽 다중성이 3..*이므로 한 Professor는 3개 이상의 Class를 강의한다. 틀린 표현은 '1개 이상'이며, 올바르게는 '3개 이상의 Class를 강의한다'이다.
② X. Student-Class 연관에서 Student 쪽 다중성이 20이므로 한 Class를 수강하는 Student 수는 정확히 20명이다. 틀린 표현은 '20명 이상'이며, 올바르게는 '20명의 Student가 수강한다'이다.
③ O. Professor-Student 연관에서 Student 쪽 다중성이 2..*이고 Professor 쪽 다중성이 1이므로 한 Professor는 2명 이상의 Student를 상담한다. 또한 한 Student는 1명의 Professor와 상담 관계를 가진다.
④ X. Student-Class 연관에서 Class 쪽 다중성이 1..*이므로 한 Student는 1개 이상의 Class를 수강한다. 틀린 표현은 '20개의 Class'이며, 올바르게는 '1개 이상의 Class를 수강한다'이다.

오답 포인트

연결선 끝에 적힌 다중성을 그 끝 클래스의 개수로 반대로 읽거나, 정확한 제약을 '이상' 정도로 뭉뚱그리면 틀린다. 이 문항은 20, 1..*, 2..*, 3..*를 정확한 수와 최소 수로 구분하는지가 핵심이다.

다시 나오면 이렇게 판단

연결선 한쪽 끝의 다중성은 반대편 객체 1개에 대해 그 끝 객체가 몇 개 연결되는지를 뜻한다고 놓고 읽는다. 그다음 단일 숫자는 정확한 개수, a..*는 최소 a개로 해석하면 된다.

암기 포인트

UML 다중성은 반대편 기준으로 읽으며, 숫자 하나는 정확한 개수이고 a..*는 최소 a개이다.

NoteBase