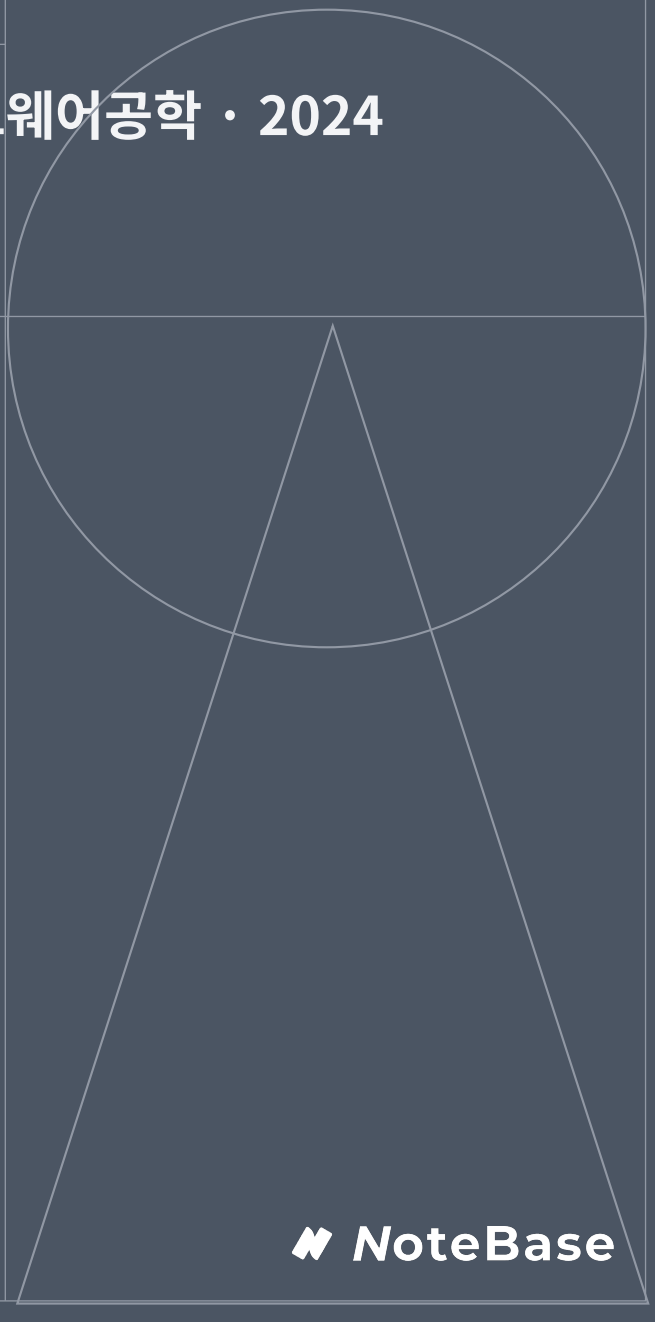
기출OX 해설집

공무원/서울시/7급 · 소프트웨어공학 · 2024



NoteBase

기출OX 해설집

공무원/서울시/7급 · 소프트웨어공학 · 2024

공무원/서울시/7급 · 소프트웨어공학 · 2024

문 1. 소프트웨어 개발 프로세스 모델에 해당하지 않는 것은?

① 폭포수 모델(waterfall model)

② 요구사항 엔지니어링(requirements engineering)

③ 프로토타이핑(prototyping)

④ 증분형 개발 모델(incremental model)

정답 2번

출제 포인트

이 문제는 소프트웨어 개발 프로세스 모델과 소프트웨어 공학 활동을 구별할 수 있는지를 묻는다.

핵심 개념

폭포수 모델은 개발 단계를 순차적으로 진행하는 프로세스 모델이다. 프로토타이핑은 시제품을 만들어 요구를 구체화하고 반복 보완하는 프로세스 모델이다. 증분형 개발 모델은 기능을 나누어 여러 번에 걸쳐 완성하는 프로세스 모델이다. 요구사항 엔지니어링은 사용자 요구를 수집·분석·명세·검증하는 활동 영역이므로 프로세스 모델 자체는 아니다.

선지별 해설

1번 O: 폭포수 모델은 요구분석, 설계, 구현, 시험을 순차적으로 수행하는 대표적인 개발 프로세스 모델이므로 맞는 보기이다.
2번 X: 요구사항 엔지니어링은 요구를 도출하고 분석하며 명세하고 검증하는 소프트웨어 공학 활동이다. 틀린 표현은 '개발 프로세스 모델'이다. 올바르게는 '소프트웨어 개발 초기에 수행하는 핵심 활동 영역'이다.
3번 O: 프로토타이핑은 초기 시제품을 바탕으로 사용자 요구를 확인하고 개선하는 개발 방식이므로 프로세스 모델에 해당한다.
4번 O: 증분형 개발 모델은 시스템을 여러 증분으로 나누어 단계적으로 제공하는 개발 프로세스 모델이므로 맞는 보기이다.

NoteBase

오답 포인트

요구사항 엔지니어링도 개발 초기에 수행되므로 모델처럼 보이기 쉽다. 그러나 모델은 개발 전체 흐름이나 반복 방식을 규정하고, 요구사항 엔지니어링은 그중 한 활동 영역이라는 점이 핵심 함정이다.

다시 나오면 이렇게 판단

보기가 개발의 전체 진행 방식이나 반복 구조를 설명하면 프로세스 모델로 본다. 반대로 요구분석, 설계, 테스트처럼 특정 업무나 공정을 가리키면 활동 영역으로 판단한다.

암기 포인트

폭포수·프로토타이핑·증분형은 모델이고, 요구사항 엔지니어링은 요구를 다루는 활동이다.

NoteBase

문 2. 소프트웨어 아키텍처 평가 방법 중, 여러 가지 품질 속성에 초점을 맞추어 평가하여 아키텍처의 설계 타협점을 찾아내고, 아키텍처 내부에 숨어 있는 리스크를 발견하는 데 사용되는 것은?

① SAAM(Software Architecture Analysis Method)

② SAEM(Software Architecture Evaluation Method)

③ ATAM(Architecture Trade-off Analysis Method)

④ AQEM(Architecture Quality Evaluation Method)

정답 3번

출제 포인트

이 문제는 소프트웨어 아키텍처 평가 방법 중 여러 품질 속성을 함께 보면서 설계 타협점과 리스크를 찾는 방법이 무엇인지를 구별하는 문제이다.

핵심 개념

SAAM은 시나리오를 바탕으로 아키텍처를 분석하는 방법으로 변경 용이성 같은 품질 속성 분석에 자주 쓰인다. ATAM은 여러 품질 속성을 동시에 놓고 상충 관계를 분석하여 설계 타협점과 민감점, 숨은 리스크를 찾는 방법이다. 이 문제의 결정 키워드는 trade-off와 risk이다.

선지별 해설

1번 X: SAAM은 Software Architecture Analysis Method로, 시나리오 기반 분석을 통해 아키텍처의 변경 용이성 등을 평가하는 데 초점이 있다. 틀린 부분은 SAAM을 여러 품질 속성의 설계 타협점과 숨은 리스크를 찾는 대표 방법으로 본 점이다. 올바르게는 그런 설명에 정확히 맞는 방법은 ATAM이다.
2번 X: SAEM이라는 명칭은 소프트웨어 아키텍처 평가를 넓게 떠올리게 하지만, 문제의 핵심인 여러 품질 속성 간 trade-off 분석과 리스크 발견을 대표하는 정식 방법명은 아니다. 틀린 부분은 일반적 평가 명칭을 trade-off 분석 기법으로 본 점이다. 올바르게는 설계 타협점과 리스크 식별을 강조하는 방법은 ATAM이다.
3번 O: ATAM은 Architecture Trade-off Analysis Method의 약자로, 이름 자체에 trade-off 분석이 들어 있다. 여러 품질 속성을 함께 평가하여 설계 타협점을 찾고 아키텍처 내부의 위험 요소를 드러내는 대표적인 평가 방법이므로 문제의 설명과 일치한다.
4번 X: AQEM은 품질 평가라는 뜻을 연상시키지만, 문제에서 묘사한 핵심 기능은 품질 속성 간 상충 관계를 분석하여 리스크를 찾는 것이다. 틀린 부분은 품질 평가라는 일반 표현을 trade-off 분석 방법으로 본 점이다. 올바르게는 여러 품질 속성의 균형과 위험을 함께 보는 방법은 ATAM이다.

오답 포인트

ATAM의 핵심 단어인 trade-off를 놓치면 SAAM처럼 익숙한 약어로 오인하기 쉽다. 이 문항은 방법론 이름 전체보다 설명 속 기능어인 여러 품질 속성, 설계 타협점, 리스크 발견을 읽어 내는지를 노린 문제이다.

NoteBase

다시 나오면 이렇게 판단

아키텍처 평가 방법 문제가 나오면 먼저 단일 품질 속성 중심인지, 여러 품질 속성 간 상충 관계까지 보는지부터 구분한다. 설계 타협점과 리스크 식별이 함께 나오면 ATAM으로 판단할 수 있다.

암기 포인트

ATAM은 Trade-off와 Risk, SAAM은 Scenario 기반 분석이라는 축으로 기억한다.

NoteBase

문 3. 정적 분석 도구에 대한 설명으로 가장 옳지 않은 것은?

① 원시 코드가 실행되는 동안의 동작이나 반응을 추적하고 보고한다.

② 원시 코드의 문법적 적합성을 자동으로 평가하여 잘못된 문장을 표기한다.

③ 원시 코드의 그래프를 생성하여 논리 흐름을 보여 주고 구조적 결함이 있는지 체크한다.

④ 원시 코드에 정의된 데이터 구조 및 선언, 컴포넌트 인터페이스를 검사하여 잘못된 링크나 데이터 정의의 충돌, 잘못된 데이터의 사용 등을 발견한다.

정답 1번

출제 포인트

이 문항은 정적 분석 도구와 동적 분석 도구의 차이, 그리고 정적 분석 도구가 발견할 수 있는 결함의 유형을 구분하는 문제이다.

핵심 개념

정적 분석은 프로그램을 실행하지 않은 상태에서 원시 코드, 데이터 구조, 인터페이스, 제어 흐름을 분석해 결함을 찾는 기법이다. 동적 분석은 프로그램을 실제로 실행하면서 동작, 반응, 실행 경로를 관찰하는 기법이다. 정적 분석의 키워드는 문법 검사, 인터페이스 일관성, 데이터 정의 충돌, 제어 흐름 분석이고, 동적 분석의 키워드는 실행 중 추적과 런타임 행위 관찰이다.

선지별 해설

① X: '원시 코드가 실행되는 동안의 동작이나 반응을 추적'한다는 표현이 틀렸다. 이는 정적 분석이 아니라 동적 분석의 설명이다. 정적 분석 도구는 실행 없이 원시 코드 자체를 분석한다.
② O: 원시 코드의 문법적 적합성을 자동으로 평가하여 잘못된 문장을 표시하는 것은 대표적인 정적 분석 기능이다.
③ O: 원시 코드의 그래프를 생성해 논리 흐름을 확인하고 구조적 결함을 점검하는 것은 정적 분석에 해당한다. 제어 흐름도나 호출 관계 분석은 실행 없이 수행할 수 있다.
④ O: 데이터 구조, 선언, 컴포넌트 인터페이스를 검사하여 잘못된 링크, 데이터 정의 충돌, 부적절한 데이터 사용을 찾는 것도 정적 분석의 전형적인 기능이다.

오답 포인트

정적 분석도 '추적'이나 '그래프 생성' 같은 표현이 나오면 실행 중 분석으로 오해하기 쉽다. 이 문항의 함정은 ①의 '실행되는 동안'이라는 문구이다. 정적 분석은 실행 전 코드 자체를 보고, 동적 분석은 실행 중 행위를 본다는 기준으로 구분하면 흔들리지 않는다.

NoteBase

다시 나오면 이렇게 판단

선지에 실행 중, 반응 추적, 런타임 동작 관찰이 나오면 먼저 동적 분석 여부를 본다. 반대로 문법 검사, 인터페이스 점검, 데이터 정의 충돌, 제어 흐름 분석처럼 코드를 실행하지 않고 가능한 작업이면 정적 분석으로 판단한다.

암기 포인트

정적 분석은 실행 전 코드 점검, 동적 분석은 실행 중 동작 추적이다.

NoteBase

문 4. 애자일 방법론과 연관성이 가장 적은 것은?

① 사용자 스토리

② 분산 시스템 모델링

③ 스크럼

④ 짝 프로그래밍

정답 2번

출제 포인트

이 문제는 애자일 방법론의 대표적 프레임워크·실천·요구사항 표현 방식과 일반 소프트웨어 공학 기법을 구별하는 문제이다.

핵심 개념

사용자 스토리는 애자일에서 요구사항을 사용자 가치 중심으로 짧게 표현하는 방식이다.
스크럼은 스프린트와 백로그를 중심으로 일하는 대표적 애자일 프레임워크이다.
짝 프로그래밍은 XP에서 대표적으로 쓰이는 애자일 실천이다.
분산 시스템 모델링은 분산 환경의 구조와 상호작용을 설계·분석하는 일반 소프트웨어 공학 기법이다.

선지별 해설

① X: 사용자 스토리는 제품 백로그 항목을 사용자 관점에서 정리할 때 쓰는 전형적인 애자일 요구사항 표현이다. 틀린 판단은 '관련성이 가장 적다'라는 부분이다. 올바르게는 애자일과 직접 관련된 개념이다.
② O: 분산 시스템 모델링은 시스템 구조를 분석하고 표현하는 일반 설계 기법이다. 애자일의 대표 프레임워크나 실천으로 보기는 어려우므로 애자일 방법론과의 관련성이 가장 적다.
③ X: 스크럼은 대표적인 애자일 프레임워크이다. 틀린 판단은 '관련성이 가장 적다'라는 부분이다. 올바르게는 애자일 방법론 그 자체를 대표하는 개념 중 하나이다.
④ X: 짝 프로그래밍은 두 개발자가 함께 코드를 작성하는 XP의 대표 실천이다. 틀린 판단은 '관련성이 가장 적다'라는 부분이다. 올바르게는 애자일 실천 기법 중 하나이다.

오답 포인트

분산 시스템 모델링은 소프트웨어 공학 용어라서 애자일과도 관련 있어 보일 수 있으나, 이 문항은 '애자일의 대표 개념인가'를 묻는다. 사용자 스토리, 스크럼, 짝 프로그래밍은 애자일의 핵심 용어이고, 분산 시스템 모델링은 일반 설계·분석 기법이라는 점에서 갈린다.

NoteBase

다시 나오면 이렇게 판단

선지가 애자일 프레임워크인지, 애자일 실천인지, 애자일 요구사항 표현 방식인지부터 본다. 여기에 직접 들어가지 않고 일반적인 설계·분석 기술에 머물면 관련성이 가장 적은 선택지로 판단한다.

암기 포인트

스토리·스크럼·짝 프로그래밍은 애자일이고, 분산 시스템 모델링은 일반 설계 기법이다.

NoteBase

문 5. <보기>는 모듈 간의 결합 형태를 보여주고 있다. 결합도가 약한 것부터 강한 것 순서대로 바르게 나열한 것은?

<보기>
ㄱ. 공통 결합(common coupling)
ㄴ. 내용 결합(content coupling)
ㄷ. 스탬프 결합(stamp coupling)
ㄹ. 자료 결합(data coupling)
ㅁ. 제어 결합(control coupling)

① ㄱ - ㄴ - ㄹ - ㄷ - ㅁ

② ㄴ - ㄹ - ㄷ - ㅁ - ㄱ

③ ㄹ - ㄴ - ㄷ - ㄱ - ㅁ

④ ㄹ - ㄷ - ㅁ - ㄱ - ㄴ

정답 4번

출제 포인트

이 문제는 모듈 결합도의 약한 순서부터 강한 순서까지를 구별할 수 있는지를 묻는 문제이다.

올바른 순서

약한 것부터 강한 것은 자료 결합 → 스탬프 결합 → 제어 결합 → 공통 결합 → 내용 결합이다.
자료 결합은 필요한 데이터만 매개변수로 주고받는 형태라 가장 약하다.
스탬프 결합은 레코드나 구조체 같은 자료구조 전체를 넘기지만 그중 일부만 사용하므로 자료 결합보다 강하다.
제어 결합은 제어용 플래그나 분기 정보를 넘겨 상대 모듈의 처리 흐름에 영향을 주므로 스탬프 결합보다 강하다.
공통 결합은 여러 모듈이 전역 데이터나 공용 영역을 함께 사용하므로 제어 결합보다 강하다.
내용 결합은 한 모듈이 다른 모듈의 내부 데이터나 내부 처리에 직접 접근하므로 가장 강하다.

NoteBase

선지별 해설

1번(X): ㄱ-ㄴ-ㄹ-ㄷ-ㅁ은 공통 결합과 내용 결합을 앞에 두어 순서가 거꾸로 되어 있다. 틀린 표현은 약한 것부터의 배열이라는 점이다. 올바르게는 ㄹ-ㄷ-ㅁ-ㄱ-ㄴ이다.
2번(X): ㄴ-ㄹ-ㄷ-ㅁ-ㄱ은 내용 결합을 맨 앞에 두었는데 내용 결합은 가장 강한 결합이다. 틀린 표현은 ㄴ을 시작점으로 둔 부분이다. 올바르게는 가장 약한 ㄹ부터 시작해 ㄹ-ㄷ-ㅁ-ㄱ-ㄴ으로 배열한다.
3번(X): ㄹ-ㄴ-ㄷ-ㄱ-ㅁ은 첫 단계만 자료 결합으로 맞고 그다음에 내용 결합을 배치한 것이 틀리다. 내용 결합은 최후에 와야 하고 제어 결합은 공통 결합보다 앞선다. 올바르게는 ㄹ-ㄷ-ㅁ-ㄱ-ㄴ이다.
4번(O): ㄹ-ㄷ-ㅁ-ㄱ-ㄴ은 자료 결합, 스탬프 결합, 제어 결합, 공통 결합, 내용 결합 순서이므로 결합도가 약한 것부터 강한 것까지의 표준 배열과 일치한다.

오답 포인트

스탬프 결합과 자료 결합, 공통 결합과 내용 결합의 위치를 자주 뒤섞는다. 특히 공통 결합을 가장 강한 것으로 착각하기 쉬운데, 다른 모듈의 내부를 직접 건드리는 내용 결합이 공통 결합보다 더 강하다.

다시 나오면 이렇게 판단

결합도 문제는 필요한 값만 주고받는가, 자료구조 전체를 넘기는가, 제어 흐름까지 건드리는가, 전역 데이터를 공유하는가, 다른 모듈 내부를 직접 건드리는가의 순서로 판단하면 된다. 이 기준을 적용하면 자료 → 스탬프 → 제어 → 공통 → 내용으로 정리된다.

암기 포인트

결합도는 자료, 스탬프, 제어, 공통, 내용 순으로 강해진다. 내부를 직접 건드리는 내용 결합이 최강 결합이다.

NoteBase

문 6. '문서편집기 소프트웨어'의 비기능적 요구사항으로 가장 옳은 것은?

① 텍스트 형식(TXT)으로 작성된 문서를 불러들일 수 있어야 한다.

② HTML 형식으로 문서를 저장할 수 있어야 한다.

③ 문서 저장은 3초 이내에 완수되어야 한다.

④ 문서 저장 중 사용자가 단축키로 취소할 수 있어야 한다.

정답 3번

출제 포인트

이 문제는 기능적 요구사항과 비기능적 요구사항의 차이를 구별하는 문제이다.

핵심 개념

기능적 요구사항은 소프트웨어가 무엇을 해야 하는지를 나타내는 기능·동작 요구이다. 비기능적 요구사항은 그 기능이 어느 수준의 성능·품질·제약을 만족해야 하는지를 나타내는 요구이다.

선지별 해설

1번 X: TXT 형식 문서를 불러들이는 것은 문서편집기가 수행해야 할 구체적 기능이다. 틀린 표현은 비기능적 요구사항으로 본 점이다. 올바르게는 기능적 요구사항이다.
2번 X: HTML 형식으로 문서를 저장하는 것도 저장 기능의 한 종류이므로 기능적 요구사항이다. 틀린 표현은 비기능적 요구사항으로 본 점이다. 올바르게는 기능적 요구사항이다.
3번 O: 문서 저장이 3초 이내에 끝나야 한다는 것은 저장 기능 자체가 아니라 수행 시간이라는 성능 기준을 제시한 것이다. 따라서 비기능적 요구사항에 해당한다.
4번 X: 저장 중 단축키로 취소할 수 있어야 한다는 것은 사용자가 사용할 수 있는 동작을 추가로 요구한 것이다. 틀린 표현은 품질 제약처럼 본 점이다. 올바르게는 취소 기능 제공에 관한 기능적 요구사항이다.

오답 포인트

저장, 불러오기, 취소처럼 동사가 보이면 기능적 요구사항인 경우가 많고, 시간·속도·용량·신뢰도처럼 기준값이 붙으면 비기능적 요구사항인 경우가 많다. 4번은 사용성처럼 보일 수 있지만 핵심은 취소 기능의 존재 여부이므로 기능적 요구사항이다.

다시 나오면 이렇게 판단

선지가 시스템의 행위를 새로 요구하면 기능적 요구사항이고, 이미 있는 기능의 성능·품질·제약 수준을 수치나 기준으로 요구하면 비기능적 요구사항이다.

NoteBase

암기 포인트

무엇을 한다는 기능, 얼마나 잘해야 한다는 비기능이다.

NoteBase

문 7. RUP(Rational Unified Process)의 4+1 뷰(view)에 대한 <보기>의 설명과 용어를 옳게 짝지은 것은?

<보기>
(가) 주로 설계자와 개발자를 위한 것으로 클래스, 객체, 관계 등과 같은 정적인 구조와 객체가 다른 객체에 메시지를 전달할 때 발생하는 동적인 협동을 설명한다.
(나) 자원의 효율적인 사용, 병행 실행 및 비동기 이벤트의 처리뿐 아니라 병행 실행되는 스레드 간의 통신과 동기화 등을 다룬다.

① (가): 구현(implementation) 뷰, (나): 프로세스(process) 뷰
② (가): 논리(logical) 뷰, (나): 구현(implementation) 뷰
③ (가): 논리(logical) 뷰, (나): 프로세스(process) 뷰
④ (가): 프로세스(process) 뷰, (나): 구현(implementation) 뷰

정답 3번

출제 포인트

이 문제는 RUP 4+1 뷰에서 논리 뷰, 프로세스 뷰, 구현 뷰의 핵심 키워드를 구분할 수 있는지를 묻는다.

핵심 개념

논리 뷰: 클래스, 객체, 관계와 객체 간 메시지 교환으로 시스템의 기능 구조를 설명하는 뷰이다. / 결정 차이축: 무엇이 어떤 관계로 기능을 이루는가를 본다. / 키워드: 클래스, 객체, 관계, 협력
프로세스 뷰: 병행 실행되는 프로세스와 스레드의 통신, 동기화, 성능을 설명하는 뷰이다. / 결정 차이축: 실행 중 동시성과 자원 사용을 어떻게 다루는가를 본다. / 키워드: 병행성, 스레드, 통신, 동기화
구현 뷰: 컴포넌트, 모듈, 패키지처럼 소스 코드를 어떻게 조직하는지를 설명하는 뷰이다. / 결정 차이축: 개발 산출물이 코드 단위로 어떻게 묶이는가를 본다. / 키워드: 컴포넌트, 모듈, 패키지

NoteBase

선지별 해설

1번: X. (가)는 클래스, 객체, 관계, 객체 간 메시지 전달에 따른 협력을 설명하므로 논리 뷰이다. 틀린 표현은 '(가) 구현 뷰'이다. 올바르게는 '(가) 논리 뷰, (나) 프로세스 뷰'이다.
2번: X. (가)를 논리 뷰로 본 것은 맞지만, (나)를 구현 뷰로 본 것이 틀리다. 구현 뷰는 모듈과 컴포넌트의 정적 코드 구성을 다루고, 병렬 실행되는 스레드 간 통신과 동기화는 프로세스 뷰가 다룬다. 틀린 표현은 '(나) 구현 뷰'이다. 올바르게는 '(나) 프로세스 뷰'이다.
3번: O. (가)는 설계자와 개발자를 위한 관점에서 클래스, 객체, 관계, 객체 협력을 설명하므로 논리 뷰에 해당한다. (나)는 자원 사용, 병행 실행, 비동기 이벤트 처리, 스레드 간 통신과 동기화를 다루므로 프로세스 뷰에 해당한다.
4번: X. (가)를 프로세스 뷰로 보면 객체 구조와 객체 협력 설명을 동시성 관점과 혼동한 것이다. 또 (나)를 구현 뷰로 보면 스레드 통신과 동기화를 코드 조직 관점과 혼동한 것이다. 틀린 표현은 '(가) 프로세스 뷰'와 '(나) 구현 뷰'이다. 올바르게는 '(가) 논리 뷰, (나) 프로세스 뷰'이다.

오답 포인트

'동적 협력'이라는 표현 때문에 (가)를 프로세스 뷰로 착각하기 쉽지만, 여기서의 동적 협력은 객체 간 메시지 상호작용이라는 기능 설계 관점이다. 반대로 스레드, 통신, 동기화가 나오면 실행 중 병행성을 다루는 프로세스 뷰로 판단해야 한다.

다시 나오면 이렇게 판단

클래스·객체·관계·메시지가 보이면 논리 뷰인지 먼저 본다. 프로세스·스레드·통신·동기화가 보이면 프로세스 뷰로 본다. 모듈·컴포넌트·패키지·소스 코드 구성이 보이면 구현 뷰로 판단한다.

암기 포인트

클래스·객체는 논리 뷰, 스레드·동기화는 프로세스 뷰, 모듈·패키지는 구현 뷰이다.

NoteBase

문 8. 소프트웨어 형상 변경 제어 활동에서, 형상 항목을 변경하는 이유를 <보기>에서 모두 고른 것은?

<보기>
ㄱ. 소프트웨어의 결함
ㄴ. 하드웨어의 변경
ㄷ. 운영 요구의 변경
ㄹ. 고객이나 사용자로부터의 개선 요구
ㅁ. 예산, 프로젝트 일정, 기간의 변경

① ㄱ, ㄴ, ㅁ
② ㄱ, ㄷ, ㄹ
③ ㄴ, ㄷ, ㄹ, ㅁ
④ ㄱ, ㄴ, ㄷ, ㄹ, ㅁ

정답 4번

출제 포인트

이 문제는 소프트웨어 형상 변경 제어에서 형상 항목을 변경하게 만드는 사유를 폭넓게 구분할 수 있는지를 묻는다.

핵심 개념

형상 항목은 소프트웨어 개발과 운영에서 식별하고 통제해야 하는 프로그램, 문서, 데이터, 관련 환경 요소를 말한다.
형상 변경 제어는 형상 항목에 대한 변경 요청을 검토하고 승인하며 반영 결과를 추적하는 활동이다.
변경 사유는 결함 수정 같은 기술적 원인뿐 아니라 하드웨어 환경 변화, 운영 요구 변화, 사용자 개선 요구, 예산과 일정 변화 같은 관리적 원인까지 포함된다.

NoteBase

선지별 해설

1번(X): 틀린 부분은 'ㄱ, ㄴ, ㅁ만'이라는 조합이다. 소프트웨어의 결함, 하드웨어의 변경, 예산·일정의 변경은 모두 형상 항목 변경 사유가 맞지만, 운영 요구의 변경인 ㄷ과 고객이나 사용자로부터의 개선 요구인 ㄹ도 함께 포함되어야 한다. 올바른 조합은 ㄱ, ㄴ, ㄷ, ㄹ, ㅁ이다.
2번(X): 틀린 부분은 'ㄱ, ㄷ, ㄹ만'이라는 조합이다. 결함 수정과 운영 요구 변경, 사용자 개선 요구는 맞는 사유이지만, 하드웨어의 변경인 ㄴ과 예산·프로젝트 일정·기간의 변경인 ㅁ도 형상 변경의 원인이 된다. 올바른 조합은 ㄱ, ㄴ, ㄷ, ㄹ, ㅁ이다.
3번(X): 틀린 부분은 'ㄴ, ㄷ, ㄹ, ㅁ'이라는 조합에서 ㄱ을 뺀 점이다. 하드웨어 변경, 운영 요구 변경, 사용자 개선 요구, 예산·일정 변경은 모두 맞지만, 소프트웨어의 결함인 ㄱ 역시 대표적인 변경 사유이다. 올바른 조합은 ㄱ, ㄴ, ㄷ, ㄹ, ㅁ이다.
4번(O): ㄱ의 결함 수정, ㄴ의 하드웨어 환경 변화, ㄷ의 운영 요구 변화, ㄹ의 고객·사용자 개선 요구, ㅁ의 예산·일정·기간 변화는 모두 형상 항목 변경을 유발할 수 있는 사유이므로 전체를 포함한 조합이 맞다.

오답 포인트

형상 변경 제어를 결함 수정 중심의 기술 활동으로만 좁게 보면 ㄴ이나 ㅁ 같은 환경·관리 요인을 빼기 쉽다. 이 문항은 변경 사유가 기술, 운영, 사용자, 관리 영역 전반에 걸친다는 점을 놓치면 틀리도록 구성되어 있다.

다시 나오면 이렇게 판단

형상 항목 변경 사유를 묻는 문항은 먼저 결함 수정 여부를 보고, 그다음 실행 환경 변화, 요구사항 변화, 사용자 개선 요구, 프로젝트 관리 조건 변화가 포함되는지를 순서대로 확인하면 된다. 어느 하나만 떠올리지 말고 기술적 원인과 관리적 원인을 함께 점검하는 방식으로 판단한다.

암기 포인트

형상 변경 사유는 결함만이 아니라 환경, 요구, 사용자, 일정까지 전부 포함된다고 정리할 수 있다.

NoteBase

문 9. Z명세 기법에 대한 설명으로 가장 옳지 않은 것은?

① 대수학적 방법에 해당한다.

② 정형적 명세 언어를 사용한다.

③ 명세를 작성하는 단위는 스키마이다.

④ 집합과 함수, 술어 논리 등의 수식을 사용한다.

정답 1번

출제 포인트

이 문제는 Z 명세가 대수학적 명세가 아니라 집합론과 술어 논리에 기반한 모델 기반 정형 명세 기법이라는 점을 구별할 수 있는지를 묻는다.

핵심 개념

Z 명세는 시스템의 상태와 연산을 수학적으로 기술하는 정형 명세 언어이다. 대수학적 명세는 연산과 그 성질을 공리적으로 기술하는 방법이고, Z 명세는 집합, 관계, 함수, 1차 술어 논리를 바탕으로 스키마를 사용해 모델을 표현한다.

선지별 해설

1번: X. 틀린 부분은 "대수학적 방법"이다. Z 명세는 대수학적 명세가 아니라 모델 기반 정형 명세 기법이다. 올바르게는 "집합론과 술어 논리에 기반한 정형 명세 방법에 해당한다"이다.
2번: O. Z는 수학적 기호를 사용하여 시스템을 엄밀하게 기술하는 정형적 명세 언어이므로 맞는 설명이다.
3번: O. Z 명세에서 상태와 연산을 기술하는 기본 작성 단위는 스키마이므로 맞는 설명이다.
4번: O. Z는 집합, 함수, 관계와 함께 1차 술어 논리의 수식을 사용하여 제약조건과 성질을 표현하므로 맞는 설명이다.

오답 포인트

대수학적 명세와 Z 명세를 둘 다 형식 명세라는 이유만으로 같은 계열로 묶으면 1번에 걸린다. 이 문항은 "형식 명세"라는 공통점보다 "공리 중심인가, 모델 중심인가"의 차이를 묻는다.

다시 나오면 이렇게 판단

형식 명세 기법 문제가 나오면 먼저 공리로 연산 성질을 적는지, 아니면 집합과 논리로 상태 모델을 기술하는지부터 구분한다. 스키마, 집합, 술어 논리가 보이면 Z 명세로 판단할 수 있다.

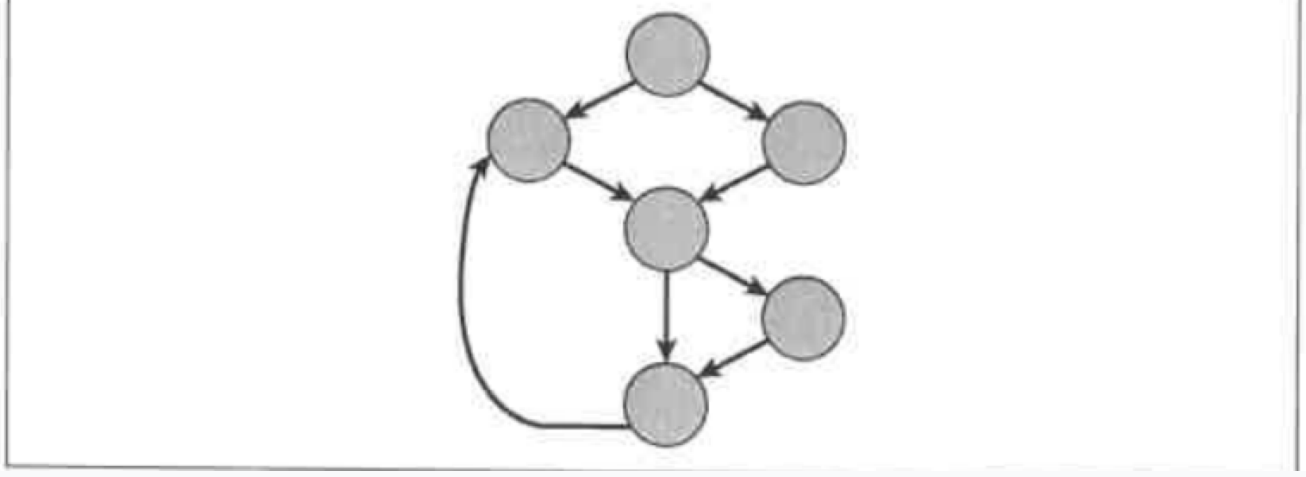
암기 포인트

Z 명세는 스키마 중심의 모델 기반 명세이고, 대수학적 명세는 공리 중심의 연산 명세이다.

NoteBase

문 10. <보기>의 제어 흐름 그래프의 순환 복잡도(cyclomatic complexity)는?

<보기>



① 4

② 5

③ 6

④ 7

정답 1번

출제 포인트

McCabe의 순환 복잡도(Cyclomatic Complexity) 공식을 제어 흐름 그래프에 적용해 값을 산출하는 문항이다. 노드 수와 간선 수를 정확히 세는 것이 핵심이다.

조건 정리

<보기> 그래프의 구성 요소는 다음과 같다.
노드(N) = 6개: 상단(n1), 좌측(n2), 우상(n3), 중앙(n4), 우하(n5), 하단(n6)
간선(E) = 8개: n1→n2, n1→n3, n2→n4, n3→n4, n4→n5, n4→n6, n5→n6, n6→n2(역방향 곡선)
연결된 그래프 수(P) = 1

사용할 공식

$V(G) = E - N + 2P$
여기서 E는 간선 수, N은 노드 수, P는 연결된 그래프 수이다. 단일 프로그램이면 P = 1이므로 $V(G) = E - N + 2$가 된다.

NoteBase

계산 과정

$V(G) = E - N + 2P = 8 - 6 + 2 \times 1 = 4$

영역(region) 수로 교차 확인: 그래프를 평면에 그렸을 때 닫힌 영역이 3개, 외부 영역이 1개이므로 합계 4개이다. 두 방법 모두 4가 나와 일치한다.

선지 판단

① 4 — 옳다. E=8, N=6, P=1이므로 V(G) = 8 − 6 + 2 = 4이다.
② 5 — 틀리다. 간선을 9개로 잘못 세거나 노드를 5개로 줄일 때 나오는 오답이다.
③ 6 — 틀리다. 간선을 10개로 과다 계산할 때 나오는 오답이다.
④ 7 — 틀리다. 간선을 11개로 과다 계산하면 11 − 6 + 2 = 7이 되는 오답이다.

오답 포인트

가장 흔한 실수는 역방향 곡선 간선(n6→n2)을 빠뜨려 E를 7로 세는 것이다. 그 경우 7 − 6 + 2 = 3이 나오는데 선지에 없으므로 잘못 셌다는 신호가 된다. 또한 노드 수를 잘못 세면 전혀 다른 값이 나오므로 노드를 하나씩 번호 붙여 확인한다.

다시 나오면 이렇게 판단

① 그래프의 모든 노드에 번호를 붙인다(세기 오류 방지). ② 화살표를 따라 간선을 하나씩 나열한다. ③ $V(G) = E - N + 2$에 대입한다. 역방향(back edge) 간선을 빠뜨리지 않도록 그래프를 전체적으로 한 번 따라가는 것이 효과적이다.

암기 포인트

순환 복잡도 공식: $V(G) = E - N + 2P$ (단일 그래프이면 P=1).
직관적 의미: V(G)는 그래프 내 독립적 선형 경로의 수이며, 소프트웨어 테스트에서 최소 테스트 케이스 수의 하한이 된다. 값이 클수록 소프트웨어 복잡도가 높다.

NoteBase

문 11. 소프트웨어 아키텍처 패턴에 해당하는 것은?

① 객체 지향 설계(Object Oriented Design)

② 리포지토리(repository)

③ 테스트 주도 개발(Test-Driven Development)

④ 유스케이스 다이어그램(usecase diagram)

정답 2번

출제 포인트

이 문제는 소프트웨어 아키텍처 패턴과 설계 방법, 개발 방법론, 모델링 산출물을 구별할 수 있는지를 묻는다.

핵심 개념

소프트웨어 아키텍처 패턴은 시스템의 상위 구조와 구성요소 간 상호작용 방식을 정한 구조적 해법이다.
리포지토리 패턴은 공용 데이터 저장소를 중심으로 여러 컴포넌트가 데이터를 읽고 쓰는 아키텍처 패턴이다.
객체 지향 설계는 객체와 클래스의 책임과 관계를 설계하는 방법이고, 테스트 주도 개발은 테스트를 먼저 작성하는 개발 방법이며, 유스케이스 다이어그램은 요구사항을 표현하는 UML 모델이다.

선지별 해설

1번 X: 객체 지향 설계는 객체, 클래스, 캡슐화, 상속 같은 개념으로 내부 구조를 설계하는 설계 접근이다. 틀린 표현은 "아키텍처 패턴"이다. 올바르게는 "객체 지향 설계는 설계 방법 또는 설계 패러다임이다"이다.
2번 O: 리포지토리는 중앙 저장소를 두고 주변 구성요소가 이를 공유하는 구조이므로 아키텍처 패턴에 해당한다. 시스템의 데이터 관리 구조와 상호작용 방식을 정한다는 점이 핵심이다.
3번 X: 테스트 주도 개발은 테스트를 먼저 작성하고 구현을 뒤따르게 하는 개발 실천 방법이다. 틀린 표현은 "아키텍처 패턴"이다. 올바르게는 "테스트 주도 개발은 개발 방법론 또는 개발 기법이다"이다.
4번 X: 유스케이스 다이어그램은 사용자와 시스템의 상호작용을 표현하는 요구분석용 UML 다이어그램이다. 틀린 표현은 "아키텍처 패턴"이다. 올바르게는 "유스케이스 다이어그램은 모델링 산출물이다"이다.

오답 포인트

리포지토리라는 단어를 저장소 구현 기술 정도로만 기억하면 2번을 단순 자료구조나 컴포넌트로 오인하기 쉽다. 반대로 객체 지향 설계, TDD, 유스케이스 다이어그램은 모두 소프트웨어 개발에서 자주 쓰이는 용어이므로 상위 개념인 아키텍처와 혼동하기 쉽다.

다시 나오면 이렇게 판단

선지가 시스템의 전체 구조와 구성요소 관계를 정하면 아키텍처 패턴이고, 객체 책임을 정하면 설계 방법이며, 개발 절차를 정하면 방법론이고, 문서나 그림으로 표현하면 모델링 산출물로 판단한다.

NoteBase

암기 포인트

리포지토리는 구조, 객체 지향 설계는 설계, TDD는 절차, 유스케이스 다이어그램은 산출물이다.

NoteBase

문 12. <보기>의 조건을 따라 원시코드 라인 수(LOC: Line of Code) 기법으로 계산한 소프트웨어 개발의 노력(Man-Month)으로 가장 옳은 값[MM]은?

<보기>
• 소프트웨어 프로젝트의 예상 원시코드 라인 수: 80,000 LOC
• 생산성 계수: 600 LOC/MM
• 프로젝트 복잡도 계수: 1.2
• 추가적인 유지보수 인력: 전체의 10% 추가 투입

① 133.3
② 146.7
③ 160.0
④ 176.0

정답 4번

출제 포인트

이 문제는 LOC 기법에서 생산성 계수, 복잡도 계수, 추가 인력을 순서대로 반영해 총 노력량을 구할 수 있는지를 묻는 문제이다.

조건 정리

주어진 값: 예상 원시코드 라인 수 80,000 LOC, 생산성 계수 600 LOC/MM, 프로젝트 복잡도 계수 1.2이다.
구해야 할 값: 소프트웨어 개발 노력인 총 Man-Month이다.
주의할 조건: 기본 노력만 구하면 안 되고, 복잡도 계수 적용 뒤 전체의 10%를 추가 투입해야 한다.

사용할 공식

기본 노력은 노력 = LOC / 생산성으로 계산한다. 복잡도 반영 노력은 기본 노력 × 복잡도 계수이다. 최종 노력은 복잡도 반영 노력 × 1.1이다.
이 공식을 쓰는 이유: 문제에서 LOC와 LOC/MM가 주어졌으므로 먼저 기본 MM를 구할 수 있고, 이어서 복잡도 계수와 추가 인력 10%를 차례대로 반영해야 하기 때문이다.

계산 과정

1. 기본 노력은 $80,000/600 = 133.3\overline{3} MM$이다.
2. 복잡도 계수 1.2를 반영하면 $133.3\overline{3} \times 1.2 = 160.0 MM$이다.
3. 여기에 전체의 10%를 추가 투입하면 $160.0 \times 1.1 = 176.0 MM$이다.

NoteBase

선지 판단

1번 X: 133.3은 80, 000/600만 계산한 값이다. 복잡도 계수 1.2와 추가 인력 10%가 빠져 있으므로 틀리다. 틀린 표현은 최종 노력으로 본 점이며, 올바르게는 기본 노력값이다.
2번 X: 146.7은 기본 노력 133.3에 10%만 더한 값에 해당한다. 복잡도 계수 1.2를 누락한 실수이므로 틀리다. 올바르게는 복잡도까지 반영한 뒤 10%를 추가해야 한다.
3번 X: 160.0은 복잡도 계수 1.2까지 반영한 값이다. 그러나 추가 유지보수 인력 10%가 아직 더해지지 않았으므로 틀리다. 올바르게는 160.0에 다시 10%를 반영한 176.0이다.
4번 O: 기본 노력 133.3 MM에 복잡도 계수 1.2를 적용해 160.0 MM를 구하고, 여기에 전체의 10%를 추가하면 176.0 MM가 된다. 따라서 정답이다.

오답 포인트

133.3, 146.7, 160.0은 모두 계산 중간값이므로 그럴듯해 보인다. 특히 ②는 추가 10%만 보고 복잡도 계수를 빼먹을 때 나오고, ③은 복잡도까지만 반영하고 마지막 추가 인력을 놓칠 때 나온다.

다시 나오면 이렇게 판단

LOC 계산형 문제는 기본 노력 LOC / 생산성을 먼저 구한 뒤, 보정 계수와 추가 투입 조건을 문제에 나온 순서대로 끝까지 반영해 최종값인지 중간값인지 확인하면 된다.

암기 포인트

LOC/MM 문제는 기본 노력 계산 후 보정 계수와 추가 투입률을 순차 적용한 값이 최종 MM이다.

NoteBase

문 13. 프로젝트관리 지식체계(PMBOK)의 관리 항목과 세부 프로세스의 연결이 옳은 것을 <보기>에서 모두 고른 것은?

<보기>
ㄱ. 프로젝트 의사소통 관리: 프로젝트 조직 기획, 팀 확보, 팀 개발
ㄴ. 프로젝트 위험 관리: 자원 기획, 비용 산정, 비용 예산 수립, 비용 통제
ㄷ. 프로젝트 범위 관리: 착수, 범위 기획, 범위 정의, 범위 검증, 범위 변경 통제
ㄹ. 프로젝트 일정 관리: 작업 정의, 작업 순서, 작업 기간 산정, 일정 개발, 일정 통제

① ㄱ, ㄷ
② ㄴ, ㄷ
③ ㄴ, ㄹ
④ ㄷ, ㄹ

정답 4번

출제 포인트

이 문제는 PMBOK의 관리 영역 이름과 각 영역에 속하는 세부 프로세스를 정확히 대응시킬 수 있는지를 묻는 문제이다.

핵심 개념

의사소통 관리는 이해관계자 사이의 정보 계획, 배포, 보고를 다루는 영역이다. 인적자원 관리는 프로젝트 조직 기획, 팀 확보, 팀 개발처럼 사람과 조직을 다루는 영역이다. 비용 관리는 비용 산정, 예산 수립, 비용 통제를 다루는 영역이고, 일정 관리는 작업 정의, 작업 순서, 기간 산정, 일정 개발, 일정 통제를 다루는 영역이다. 범위 관리는 범위 정의와 검증, 변경 통제를 통해 무엇을 할 것인지를 확정하는 영역이다.

NoteBase

선지별 해설

1번 X: ㄱ과 ㄷ의 조합이다. ㄷ은 프로젝트 범위 관리와 그 세부 프로세스의 연결이 맞다. 반면 ㄱ은 틀리다. ㄱ의 '프로젝트 조직 기획, 팀 확보, 팀 개발'은 의사소통 관리가 아니라 인적자원 관리의 세부 프로세스이다. 따라서 1번은 틀리다.
2번 X: ㄴ과 ㄷ의 조합이다. ㄷ은 맞지만 ㄴ이 틀리므로 전체는 틀리다. ㄴ의 '자원 기획, 비용 산정, 비용 예산 수립, 비용 통제'는 위험 관리가 아니라 비용 관리에 대응하는 흐름이다. 따라서 2번은 틀리다.
3번 X: ㄴ과 ㄹ의 조합이다. ㄹ은 프로젝트 일정 관리와 작업 정의, 작업 순서, 작업 기간 산정, 일정 개발, 일정 통제가 정확히 연결되어 맞다. 그러나 ㄴ은 위험 관리가 아니라 비용 관리이므로 틀리다. 따라서 3번은 틀리다.
4번 O: ㄷ과 ㄹ의 조합이다. ㄷ의 범위 관리는 착수, 범위 기획, 범위 정의, 범위 검증, 범위 변경 통제로 연결되고, ㄹ의 일정 관리는 작업 정의, 작업 순서, 작업 기간 산정, 일정 개발, 일정 통제로 연결된다. 두 보기 모두 올바르므로 4번이 정답이다.

오답 포인트

ㄱ의 '팀 확보, 팀 개발'을 보고도 의사소통 관리로 읽거나, ㄴ의 '비용 산정, 비용 예산 수립, 비용 통제'를 보고도 위험 관리로 읽으면 틀리게 된다. 사람 관련 용어는 인적자원 관리, 비용 관련 용어는 비용 관리로 바로 연결해야 한다.

다시 나오면 이렇게 판단

관리 영역 이름부터 외우기보다 세부 프로세스의 핵심 단어를 먼저 본다. 팀과 조직이 나오면 인적자원 관리, 비용과 예산이 나오면 비용 관리, 범위 정의와 검증이 나오면 범위 관리, 작업 순서와 일정 개발이 나오면 일정 관리로 판단하면 된다.

암기 포인트

팀은 인적자원, 비용은 비용관리, 범위는 정의·검증, 일정은 작업순서·기간산정으로 묶인다.

NoteBase

문 14. SPICE(ISO/IEC 15504) 모델에서 정의한 프로세스 수행 능력 단계(level)에 해당하지 않는 것은?

① 수행 단계(performed process)

② 관리 단계(managed process)

③ 검증 단계(verified process)

④ 최적화 단계(optimizing process)

정답 3번

출제 포인트

SPICE(ISO/IEC 15504)의 프로세스 수행능력 단계 명칭을 정확히 구분할 수 있는지를 묻는 문제이다.

핵심 개념

수행 단계는 프로세스가 목적한 작업을 실제로 수행하는 수준이다. 관리 단계는 수행되는 프로세스가 계획되고 점검되며 통제되는 수준이다. 확립 단계는 표준화된 프로세스가 조직 차원에서 정의되어 사용되는 수준이다. 예측 단계는 측정과 정량 관리가 가능한 수준이고, 최적화 단계는 지속적 개선이 이루어지는 수준이다. SPICE의 공식 능력 단계 명칭에는 검증 단계가 없다.

선지별 해설

1번 O: 수행 단계(performed process)는 SPICE 능력수준 1의 명칭이므로 맞다. 프로세스가 목적을 달성하도록 실제 수행되는 상태를 뜻한다.
2번 O: 관리 단계(managed process)는 능력수준 2의 명칭이므로 맞다. 수행된 프로세스가 계획, 감시, 조정되는 상태를 뜻한다.
3번 X: 틀린 부분은 \"검증 단계(verified process)\"이다. SPICE 능력 단계에는 이런 명칭이 없고, 해당 자리의 올바른 표현은 \"확립 단계(established process)\"이다.
4번 O: 최적화 단계(optimizing process)는 능력수준 5의 명칭이므로 맞다. 정량적 관리 결과를 바탕으로 지속적으로 개선하는 상태를 뜻한다.

오답 포인트

수험생이 CMMI의 검증 활동이나 일반적인 품질검증 용어를 SPICE 능력수준 명칭과 섞어 기억하면 3번에 흔들린다. SPICE의 단계명은 수행-관리-확립-예측-최적화의 흐름으로 외워 두어야 한다.

다시 나오면 이렇게 판단

SPICE 문제는 먼저 능력수준의 공식 명칭 순서를 떠올린 뒤, 보기의 용어가 그 목록에 실제로 존재하는지를 확인하면 된다. 특히 3수준은 검증이 아니라 확립, 4수준은 예측이라는 점을 기준으로 판단한다.

NoteBase

암기 포인트

SPICE 능력수준은 1 수행, 2 관리, 3 확립, 4 예측, 5 최적화이다.

NoteBase

문 15. 맥콜(McCall)의 품질 요소에 대한 설명으로 옳은 것을 <보기>에서 모두 고른 것은?

<보기>
ㄱ. 맥콜의 품질 요소는 크게 제품 운영, 제품 개선, 제품 변환으로 분류할 수 있다.
ㄴ. 효율성은 사용자가 요구하는 기능을 수행하는 데 CPU와 메모리 같은 자원을 얼마나 사용하는지와 관련된 특성이다.
ㄷ. 이식성은 시스템의 일부나 전체를 다른 애플리케이션에서도 얼마나 쉽게 사용할 수 있는지를 나타낸다.
ㄹ. 무결성은 개발된 소프트웨어가 사용자의 기능적 요구사항을 담은 요구분석명세서와 얼마나 일치하는지를 나타낸다.

① ㄱ, ㄴ
② ㄱ, ㄷ
③ ㄴ, ㄷ
④ ㄴ, ㄹ

정답 1번

출제 포인트

맥콜의 품질 요소에서 분류 체계와 효율성, 이식성, 무결성의 의미를 정확히 구별하는 문제이다.

핵심 개념

제품 운영은 실행 중 품질을 보는 범주로 정확성, 신뢰성, 효율성, 무결성, 사용성이 포함된다. 제품 개선은 유지보수 관점의 범주로 유지보수성, 유연성, 시험성이 포함된다. 제품 변환은 환경 변화에 대한 범주로 이식성, 재사용성, 상호운용성이 포함된다. 효율성은 기능 수행에 드는 시간, CPU, 메모리 같은 자원 사용 정도이다. 이식성은 다른 하드웨어나 운영체제 환경으로 쉽게 옮길 수 있는 성질이고, 다른 애플리케이션에서 쉽게 활용할 수 있는 성질은 재사용성이다. 무결성은 비인가 접근이나 변조로부터 시스템과 데이터를 보호하는 성질이고, 요구사항과의 일치 정도는 정확성에 해당한다.

NoteBase

선지별 해설

① O: ㄱ은 맥콜의 품질 요소를 제품 운영, 제품 개선, 제품 변환으로 나눈 설명이므로 맞다. ㄴ은 효율성을 CPU와 메모리 같은 자원 사용 정도로 설명하므로 맞다.
② X: ㄱ은 맞지만 ㄷ이 틀려서 전체 조합이 틀리다. ㄷ의 틀린 표현은 '다른 애플리케이션에서도 얼마나 쉽게 사용할 수 있는지'이다. 실제로는 다른 애플리케이션에서 쉽게 사용할 수 있는 성질은 재사용성이고, 이식성은 다른 환경으로 쉽게 이전할 수 있는 성질이다.
③ X: ㄴ은 맞지만 ㄷ이 틀리므로 전체 조합이 틀리다. ㄷ은 이식성과 재사용성을 서로 바꾸어 설명했다. 올바르게는 이식성은 환경 이전의 용이성이고, 재사용성은 다른 응용에서의 활용 가능성이다.
④ X: ㄴ은 맞지만 ㄹ이 틀리므로 전체 조합이 틀리다. ㄹ의 틀린 표현은 '요구분석명세서와 얼마나 일치하는지'를 무결성으로 본 부분이다. 실제로 그 설명은 정확성에 해당하고, 무결성은 비인가 접근이나 변조로부터 보호하는 성질이다.

오답 포인트

이식성과 재사용성, 무결성과 정확성을 서로 바꾸어 외우면 쉽게 틀린다. 특히 '다른 애플리케이션에서 사용'은 재사용성이고 '요구사항과의 일치'는 정확성이라는 대응을 놓치면 ②나 ④를 고르기 쉽다.

다시 나오면 이렇게 판단

맥콜 문제는 먼저 세 분류인 제품 운영, 제품 개선, 제품 변환을 확인한 뒤 각 품질 요소를 한 줄 정의로 대조하면 된다. 자원 사용은 효율성, 환경 이전은 이식성, 다른 응용 활용은 재사용성, 비인가 접근 방지는 무결성, 요구사항 일치는 정확성으로 연결하면 조합형 선지를 빠르게 걸러낼 수 있다.

암기 포인트

효율성은 자원 사용, 이식성은 환경 이전, 재사용성은 다른 응용 활용, 무결성은 보호, 정확성은 요구사항 일치이다.

NoteBase

문 16. 화이트박스 테스트의 케이스 선정 기준에 대한 설명으로 가장 옳지 않은 것은?

① 경로 검증 기준: 프로그램에 존재하는 모든 실행 가능한 경로를 적어도 1번은 실행해야 한다.

② 기능 검증 기준: 프로그램에 구현된 모든 기능을 적어도 1번은 실행해야 한다.

③ 문장 검증 기준: 프로그램의 모든 문장을 적어도 1번은 실행해야 한다.

④ 분기 검증 기준: 프로그램에 존재하는 모든 분기점에서 참과 거짓에 해당하는 경로를 각각 적어도 1번씩 실행해야 한다.

정답 2번

출제 포인트

이 문제는 화이트박스 테스트의 대표 커버리지 기준인 문장·분기·경로 검증과 기능 검증의 차이를 구별하는지를 묻는다.

핵심 개념

문장 검증은 프로그램의 각 문장을 한 번 이상 수행하는 기준이다. 분기 검증은 각 분기점의 참·거짓 결과를 모두 한 번 이상 수행하는 기준이다. 경로 검증은 실행 가능한 모든 경로를 한 번 이상 수행하는 기준이다. 기능 검증은 구현 기능이 요구사항대로 동작하는지를 보는 관점으로, 화이트박스 커버리지 기준의 대표 항목이 아니라 블랙박스 테스트 성격에 가깝다.

선지별 해설

1번: O. 경로 검증 기준은 프로그램 안의 실행 가능한 각 경로를 적어도 한 번씩 수행하는 것이다. 화이트박스 테스트에서 가장 강한 수준의 커버리지 개념으로 보는 설명이므로 옳다.
2번: X. 틀린 표현은 "기능 검증 기준"을 화이트박스 테스트의 케이스 선정 기준처럼 제시한 부분이다. 기능 중심 검증은 내부 제어구조 커버리지가 아니라 요구 기능 수행 여부를 확인하는 관점이므로 블랙박스 테스트에 가깝다. 올바르게는 "기능 검증은 구현된 기능이 요구사항대로 동작하는지 확인하는 관점이다" 정도로 고쳐야 한다.
3번: O. 문장 검증 기준은 프로그램의 모든 문장을 적어도 한 번 이상 실행하는 것이다. 문장 커버리지의 정의와 일치하므로 옳다.
4번: O. 분기 검증 기준은 각 분기점에서 참과 거짓이 모두 한 번 이상 나오도록 테스트하는 것이다. 분기 커버리지의 정의를 정확히 설명한 선지이므로 옳다.

오답 포인트

기능이라는 단어 때문에 2번을 그럴듯한 화이트박스 기준으로 착각하기 쉽다. 그러나 화이트박스 기준은 내부 코드 구조의 실행 여부를 따지고, 기능 검증은 외부 요구 기능 충족 여부를 본다는 차이가 핵심이다.

NoteBase

다시 나오면 이렇게 판단

화이트박스 선지에서는 문장, 분기, 조건, 경로처럼 코드 내부 제어구조를 세는 기준인지 먼저 확인한다. 기능, 요구사항, 입출력 결과처럼 외부 동작 중심 표현이면 블랙박스 관점일 가능성이 크다.

암기 포인트

화이트박스는 코드 커버리지, 블랙박스는 기능 확인이라는 축으로 구분된다.

NoteBase

문 17. 델파이 기법의 비용 산정 과정에 대한 설명으로 옳은 것을 <보기>에서 모두 고른 것은?

<보기>
ㄱ. 조정자는 전문가가 모여 비용 산정을 하는 회의에서 간사 역할을 한다.
ㄴ. 전문가 각자가 비용을 산정하고 이때 계산된 결과를 조정자에게 익명으로 제출한다.
ㄷ. 전문가는 비용을 산정할 수 있는 자료를 충분히 검토하고, 필요하다면 익명으로 의견을 나눌 수 있다.
ㄹ. 조정자는 각 전문가가 제출한 자료에서 산정 내용에 차이가 크면 이 문제를 해결하기 위해 회의를 소집한다.

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

정답 4번

출제 포인트

이 문제는 델파이 기법의 핵심 절차인 조정자 역할, 전문가의 익명 추정, 반복적 의견 수렴 과정을 정확히 아는지를 묻는 문제이다.

핵심 개념

델파이 기법은 여러 전문가가 비용을 독립적으로 추정한 뒤 조정자에게 익명으로 제출하고, 조정자가 이를 정리해 다시 피드백하면서 견해 차이를 줄여 가는 비용 산정 기법이다. 핵심 키워드는 전문가 판단, 익명성, 조정자, 반복 조정이다.

선지별 해설

1번 X: 틀린 부분은 'ㄱ, ㄴ'만 고른 점이다. ㄱ의 조정자 역할과 ㄴ의 익명 제출은 맞지만, 전문가가 자료를 충분히 검토한 뒤 필요하면 익명으로 의견을 내는 ㄷ과 산정 차이가 클 때 조정자가 조정 절차를 진행하는 ㄹ도 옳다. 올바르게는 ㄱ, ㄴ, ㄷ, ㄹ을 모두 골라야 한다.
2번 X: 틀린 부분은 'ㄴ'을 빠뜨린 점이다. 델파이 기법에서는 각 전문가가 비용을 개별적으로 산정하고 그 결과를 조정자에게 익명으로 제출하므로 ㄴ도 옳다. 올바르게는 ㄱ, ㄴ, ㄷ, ㄹ이다.
3번 X: 틀린 부분은 'ㄱ'을 빠뜨린 점이다. 델파이 기법에는 전문가 의견을 수합하고 조정하는 조정자 역할이 포함되므로 ㄱ도 옳다. 올바르게는 ㄱ, ㄴ, ㄷ, ㄹ이다.
4번 O: ㄱ, ㄴ, ㄷ, ㄹ이 모두 델파이 기법의 설명에 해당하므로 옳다. 조정자가 절차를 운영하고, 전문가는 충분한 자료를 검토한 뒤 익명으로 추정치를 제출하며, 차이가 크면 추가 조정 과정을 거친다.

NoteBase

오답 포인트

델파이 기법을 단순한 집단회의 기법으로만 기억하면 익명 제출과 반복 피드백을 놓치기 쉽다. 반대로 익명성만 기억하면 조정자 역할이나 차이 조정 절차를 빠뜨려 1번이나 3번으로 흔들릴 수 있다.

다시 나오면 이렇게 판단

델파이 기법이 나오면 전문가 개별 추정, 익명 제출, 조정자 수합, 차이 조정의 네 요소가 들어 있는지 먼저 확인한다. 이 네 요소가 모두 보이면 옳은 설명으로 판단할 수 있다.

암기 포인트

델파이 = 전문가 개별 추정 + 익명 제출 + 조정자 피드백 + 반복 조정이다.

NoteBase

문 18. <보기>는 프로젝트 P의 완성에 필요한 작업 T1~T5에 대해, 작업별 최소한의 소요 기간과 선행 작업을 보여 준다. 프로젝트 P 완성에 소요되는 최소한의 기간[일]은?

<보기>

작업	소요 기간(일)	선행 작업
T1	5	-
T2	8	-
T3	1	T1
T4	2	T1, T2
T5	3	T3, T4

① 13

② 14

③ 15

④ 16

정답 1번

출제 포인트

이 문제는 작업별 선행관계를 따라 가장 오래 걸리는 경로, 즉 주공정을 계산해 프로젝트의 최소 완료 기간을 구하는 문제이다.

조건 정리

주어진 값: T1은 5일, T2는 8일, T3은 1일, T4는 2일, T5는 3일이다.
구해야 할 값: 프로젝트 P 전체가 끝나는 최소 기간이다.
주의할 조건: 선행 작업이 둘 이상이면 가장 늦게 끝나는 선행 작업까지 모두 끝나야 다음 작업을 시작할 수 있다.

NoteBase

계산 과정

T1과 T2는 선행 작업이 없으므로 동시에 시작할 수 있다.
T3는 T1 후에 시작하므로 완료 시점은 5+1=6일이다.
T4는 T1과 T2가 모두 끝나야 시작하므로 시작 시점은 max(5, 8)=8일이고 완료 시점은 8+2=10일이다.
T5는 T3와 T4가 모두 끝나야 시작하므로 시작 시점은 max(6, 10)=10일이고 완료 시점은 10+3=13일이다.
따라서 최소 완료 기간은 13일이다.

선지별 해설

① O: 계산 결과가 13일이므로 옳다. 경로로 보면 T2→T4→T5가 8+2+3=13일로 가장 길다.
② X: 14일이 아니다. T4를 계산할 때 T1과 T2를 더해서 5+8+2로 보는 오해에서 나오기 쉽다. 병렬 작업인 T1과 T2는 더하지 않고 더 늦게 끝나는 8일만 반영한다. 올바르게는 13일이다.
③ X: 15일이 아니다. 모든 작업 시간을 단순 합산하거나 병렬 가능 작업까지 직렬로 보는 오해에서 나오는 값이다. T1과 T2는 동시에 진행 가능하므로 전부 더하면 안 된다. 올바르게는 13일이다.
④ X: 16일이 아니다. T3와 T4 뒤의 T5를 계산하면서 선행 경로를 중복 합산하는 오해에서 나오는 값이다. T5는 T3와 T4가 모두 끝난 뒤 시작하지만 두 경로 시간을 다시 더하는 구조는 아니다. 올바르게는 13일이다.

오답 포인트

이 문항의 함정은 선행 작업이 여러 개인 경우 시간을 합산하는 데 있다. T4는 T1과 T2를 모두 기다리지만 5일과 8일을 더하는 것이 아니라 더 늦게 끝나는 8일을 기준으로 시작한다. 또한 프로젝트 기간은 모든 작업 시간의 총합이 아니라 가장 긴 선행 경로의 길이이다.

다시 나오면 이렇게 판단

작업 스케줄 문제는 시작 가능한 작업을 먼저 찾고, 각 작업의 완료 시점을 앞에서부터 적어 나가면 된다. 선행 작업이 여러 개이면 선행 완료 시점의 최댓값을 잡고, 마지막 작업의 완료 시점이 전체 최소 기간이 된다.

암기 포인트

병렬 작업은 더하지 않고, 합류 지점에서는 선행 완료 시점의 최댓값을 본다.

NoteBase

문 19. <보기 1>의 UML 다이어그램 용어와 <보기 2>의 설명을 옳게 짝지은 것은?

<보기 1>
(가) 통신 다이어그램 (나) 상태 다이어그램
(다) 컴포넌트 다이어그램 (라) 배치 다이어그램

<보기 2>
A: 구현 관점에서 정적 모델링을 할 때 사용하는 것으로 어떤 실행 모듈이 존재하고 이들이 서로 어떤 연관성이 있는지의 종속 관계를 나타낸다.
B: 객체 간에 메시지를 주고받는 관계를 링크라고 하며 링크를 사용해 객체 간의 관계를 표현한다.
C: 하드웨어 자원을 명시적으로 정의해 시스템의 물리적인 요소를 모델링하고 노드 간의 관계를 나타낸다.
D: 다이어그램을 그리는 데 필요한 요소는 상태, 전이, 이벤트이다.

① (가): A, (나): B, (다): C, (라): D
② (가): B, (나): D, (다): A, (라): C
③ (가): C, (나): B, (다): D, (라): A
④ (가): D, (나): C, (다): A, (라): B

정답 2번

출제 포인트

이 문제는 UML 다이어그램의 대표 용도와 핵심 구성 요소를 구분하여 올바르게 짝짓는지를 묻는 문제이다.

핵심 개념

통신 다이어그램은 객체들이 링크를 통해 메시지를 주고받는 상호작용을 나타내는 다이어그램이다. 상태 다이어그램은 상태, 전이, 이벤트를 중심으로 객체의 상태 변화를 표현한다. 컴포넌트 다이어그램은 구현 관점에서 실행 모듈과 그 종속 관계를 나타낸다. 배치 다이어그램은 노드와 하드웨어 자원 등 시스템의 물리적 배치를 표현한다.

NoteBase

선지별 해설

1번 X: (가)를 A에 연결한 점이 틀리다. 통신 다이어그램은 실행 모듈의 종속 관계가 아니라 객체 간 링크와 메시지 흐름을 나타내므로 B가 맞다. 또한 (나)는 상태 다이어그램이므로 D가 맞고, (다)는 컴포넌트 다이어그램이므로 A가 맞으며, (라)는 배치 다이어그램이므로 C가 맞다.
2번 O: (가)=B, (나)=D, (다)=A, (라)=C의 연결이 모두 맞다. B는 객체 간 링크와 메시지 교환을 설명하므로 통신 다이어그램이고, D는 상태·전이·이벤트를 요소로 하므로 상태 다이어그램이다. A는 구현 관점의 정적 모델링에서 실행 모듈과 종속 관계를 다루므로 컴포넌트 다이어그램이고, C는 하드웨어 자원과 노드 간 관계를 나타내므로 배치 다이어그램이다.
3번 X: (가)를 C에 연결한 점이 틀리다. 통신 다이어그램은 물리적 노드 관계가 아니라 객체 상호작용을 나타낸다. 틀린 표현은 통신 다이어그램을 배치 다이어그램처럼 본 점이며, 올바르게는 (가)=B, (나)=D, (다)=A, (라)=C이다.
4번 X: (가)를 D에 연결한 점이 틀리다. 통신 다이어그램은 상태 변화가 아니라 객체 간 메시지 교환을 표현한다. 또한 (나)를 C로 둔 것도 틀리다. 상태 다이어그램의 핵심은 노드가 아니라 상태, 전이, 이벤트이다. 올바르게는 (가)=B, (나)=D, (다)=A, (라)=C이다.

오답 포인트

컴포넌트 다이어그램과 배치 다이어그램은 모두 구현 관점에서 보이기 때문에 자주 혼동된다. 그러나 컴포넌트는 모듈과 종속 관계를, 배치는 노드와 하드웨어 같은 물리 요소를 다룬다. 통신 다이어그램과 상태 다이어그램도 모두 동적인 느낌이 있지만, 통신은 객체 간 메시지 관계이고 상태는 한 객체의 상태 변화라는 점이 다르다.

다시 나오면 이렇게 판단

먼저 설명이 객체 간 메시지와 링크를 말하면 통신 다이어그램으로 본다. 상태·전이·이벤트를 말하면 상태 다이어그램으로 본다. 구현 모듈과 종속 관계를 말하면 컴포넌트 다이어그램으로, 노드와 하드웨어 자원 같은 물리 배치를 말하면 배치 다이어그램으로 판단한다.

암기 포인트

통신=객체 링크·메시지, 상태=상태·전이·이벤트, 컴포넌트=모듈·종속, 배치=노드·하드웨어이다.

NoteBase

문 20. <보기 1>은 옵서버 패턴(observer pattern)을 이용한 파이썬 코드를 보여준다. 이 코드를 실행한 결과가 <보기 2>가 되도록 (가), (나)에 들어갈 코드를 옳게 짝지은 것은?

<보기 1>

```
class Subject:
    def __init__(self):
        self._observers = []
        self._state = None

    def attach(self, o):
        self._observers.append(o)

    def notify(self):
        for o in self._observers:
            (가)

    def change_state(self, state):
        self._state = state
        (나)

class Observer:
    def __init__(self, id):
        self._id = id

    def update(self, state):
        print(f"{self._id}:{state}")

s = Subject()
s.attach(Observer(1))
s.attach(Observer(2))
s.change_state("Hello")
```

<보기 2>

1:Hello

2:Hello

① (가): self.change_state(self._state), (나): self.attach(state)

② (가): self.change_state(self._state), (나): self.notify()

③ (가): o.update(self._state), (나): self.attach(state)

④ (가): o.update(self._state), (나): self.notify()

정답 4번

출제 포인트

이 문제는 옵서버 패턴에서 상태 변경 시 Subject가 등록된 Observer들에게 update를 호출하는 흐름을 코드에 적용할 수 있는지를 묻는다.

NoteBase

사례 정리

Subject는 observers 목록과 state를 가진다.
Observer는 update(state) 메서드로 자신의 id와 전달받은 state를 출력한다.
최종 출력이 1:Hello, 2:Hello가 되려면 change_state에서 상태를 바꾼 뒤 notify를 호출하고, notify 안에서는 각 Observer의 update에 현재 상태를 넘겨야 한다.

핵심 개념

옵서버 패턴은 Subject의 상태가 바뀌면 등록된 Observer들에게 변경 사실을 통지하는 패턴이다.
적용 요건은 Subject가 Observer를 보관하고, 상태 변경 시 각 Observer의 갱신 메서드를 호출해야 한다는 점이다.

적용 기준

(가)는 for o in self._observers 반복문 안에서 각 옵서버를 직접 갱신해야 하므로 o.update(self._state)여야 한다.
(나)는 상태 변경 직후 등록된 옵서버 전체에 알림을 보내야 하므로 self.notify()여야 한다.

선지별 해설

1번: X. (나)가 self.attach(state)이므로 상태 문자열 "Hello"를 옵서버 목록에 추가할 뿐 알림이 발생하지 않는다. 틀린 표현은 self.attach(state)이고, 올바르게는 self.notify()이다. 또한 (가)도 각 옵서버를 갱신하는 코드가 아니라 Subject 자신의 change_state를 다시 호출하는 코드이므로 부적절하다. 올바르게는 o.update(self._state)이다.
2번: X. (나)의 self.notify()는 맞지만, (가)의 self.change_state(self._state)는 notify 안에서 다시 change_state를 호출해 notify를 반복시키므로 정상 출력이 아니라 재귀 호출 흐름이 된다. 틀린 표현은 self.change_state(self._state)이고, 올바르게는 o.update(self._state)이다.
3번: X. (가)의 o.update(self._state)는 맞지만, (나)의 self.attach(state)는 Observer 객체가 아니라 문자열 state를 등록하므로 통지가 일어나지 않는다. 틀린 표현은 self.attach(state)이고, 올바르게는 self.notify()이다.
4번: O. notify는 등록된 각 Observer 객체 o에 대해 o.update(self._state)를 호출하고, change_state는 상태를 바꾼 직후 self.notify()를 호출하므로 1번 옵서버와 2번 옵서버가 각각 Hello를 출력한다.

오답 포인트

attach와 notify의 역할을 섞으면 틀린다. attach는 구독자 등록이고 notify는 등록된 구독자에게 현재 상태를 전파하는 동작이다. 또한 notify 안에서 Subject 자신의 change_state를 다시 부르면 통지가 아니라 재귀 호출이 된다.

NoteBase

다시 나오면 이렇게 판단

옵서버 패턴 코드 문제는 등록 단계와 통지 단계를 분리해서 본다. 상태 변경 메서드에서는 notify 호출 여부를 먼저 확인하고, notify 내부에서는 각 Observer의 update에 현재 상태를 넘기는지만 확인하면 된다.

암기 포인트

Observer 패턴의 핵심 흐름은 상태 변경 -> notify -> 각 observer.update(state)이다.

NoteBase