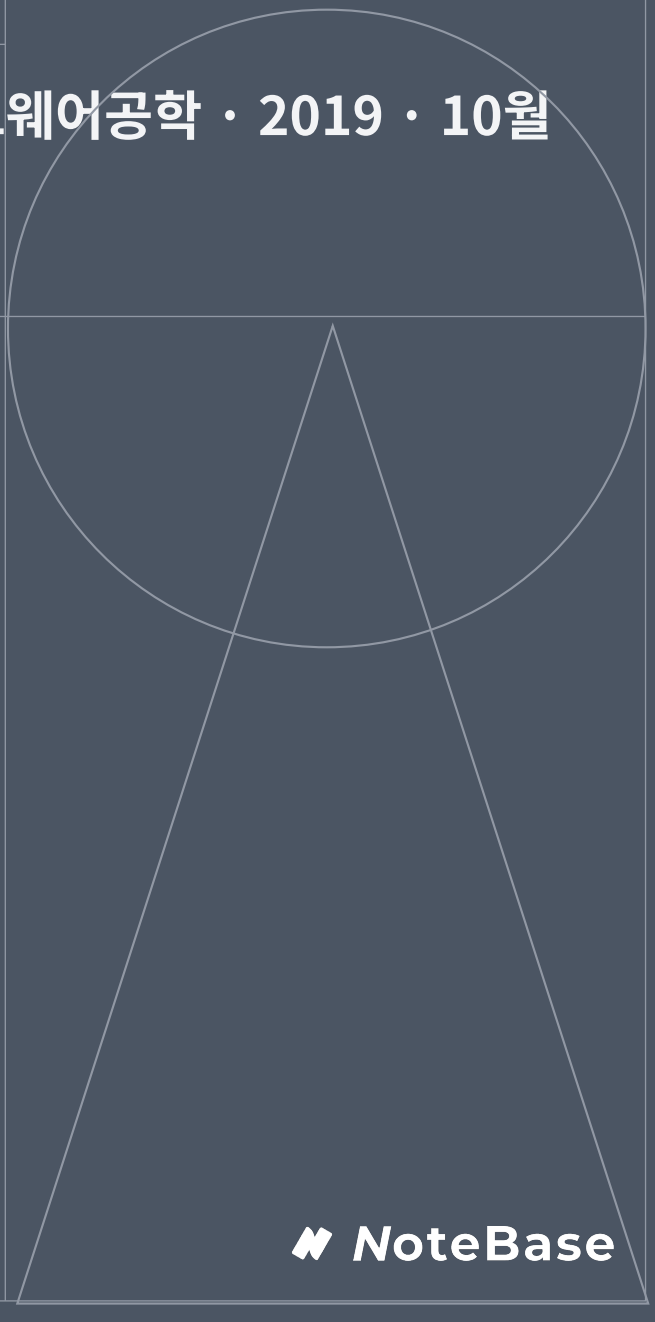
기출OX 해설집

공무원/서울시/7급 · 소프트웨어공학 · 2019 · 10월



NoteBase

기출OX 해설집

공무원/서울시/7급 · 소프트웨어공학 · 2019 · 10월

공무원/서울시/7급 · 소프트웨어공학 · 2019 · 10월

문 1. 소프트웨어 위기(crisis)에 대한 설명으로 가장 옳지 않은 것은?

① 소프트웨어 생산성이 사용자들의 서비스에 대한 요구를 따라가지 못한다.

② 소프트웨어 품질 향상과 유지보수가 힘들다.

③ 대부분의 경우 프로젝트 개발 일정과 소요 비용 예측이 부정확하다.

④ 고객의 기대치를 낮추고, 개발 생산성은 높여 해결하는 것이 바람직하다.

정답 4번

출제 포인트

이 문제는 소프트웨어 위기의 전형적 증상과 올바른 해결 방향을 구별할 수 있는지를 묻는다.

핵심 개념

소프트웨어 위기는 사용자 요구와 시스템 규모가 빠르게 커지는데 개발 생산성, 품질, 유지보수성이 그 속도를 따라가지 못해 일정 지연, 비용 초과, 품질 저하가 나타나는 현상이다.
해결 방향은 고객 기대치를 인위적으로 낮추는 것이 아니라 개발 방법론, 도구, 표준화, 재사용 등을 통해 생산성과 품질을 높이는 데 있다.

선지별 해설

1번 O: 소프트웨어 위기의 핵심 증상 중 하나가 증가하는 사용자 요구를 개발 생산성이 따라가지 못하는 것이다. 요구 증가 속도와 생산성 격차를 짚은 선지이므로 옳다.
2번 O: 소프트웨어 위기에서는 품질 확보가 어렵고, 복잡도가 커질수록 유지보수 부담도 커진다. 품질 향상과 유지보수가 힘들다는 설명은 대표적 문제점을 정확히 말한 것이다.
3번 O: 소프트웨어는 무형성, 복잡성, 요구 변경 때문에 개발 일정과 비용을 정확히 예측하기 어렵다. 일정 지연과 비용 초과는 소프트웨어 위기의 전형적 결과이므로 옳다.
4번 X: 틀린 표현은 "고객의 기대치를 낮추고"이다. 소프트웨어 위기의 해결은 기대치를 낮추는 데 있지 않다. 올바르게는 요구를 합리적으로 관리하면서 개발 생산성, 품질, 유지보수성을 높여 문제를 줄여야 한다.

NoteBase

오답 포인트

4번은 "개발 생산성을 높여"라는 맞는 표현을 섞어 두어 전체 문장을 맞는 말처럼 보이게 만든다. 그러나 소프트웨어 위기의 해결책을 묻는 문항에서 기대치 하향은 본질적 해법이 아니므로 이 부분을 잡아내야 한다.

다시 나오면 이렇게 판단

소프트웨어 위기 문제는 먼저 증상인지 해결책인지를 나눈다. 증상이라면 일정 지연, 비용 초과, 품질 저하, 유지보수 곤란을 찾고, 해결책이라면 생산성 향상, 품질 개선, 방법론과 도구의 개선인지 확인한다. 기대치를 낮춘다는 식의 소극적 표현은 오답 가능성이 크다.

암기 포인트

소프트웨어 위기 = 요구 증가는 빠르고 생산성·품질·유지보수성은 뒤처져 일정과 비용이 흔들리는 상태이다.

NoteBase

문 2. <보기>와 같은 장점을 가지는 테스팅 기법은?

<보기>
• 테스팅이 매일 시행되기 때문에, 비호환성 및 다른 중대한 오류들을 조기에 발견할 수 있어서 통합 위험이 최소화된다.
• 접근 방식이 통합 지향적이기 때문에, 컴포넌트 수준의 설계 오류뿐만 아니라 기능 오류를 발견하기 쉬우므로 최종 제품의 품질이 향상된다.
• 테스팅 중에 발견된 오류는 '소프트웨어의 새로운 증분(increment)'과 관련된 경우가 많다. 즉, 빌드에 새롭게 추가된 소프트웨어가 새로 발견된 오류의 원인일 가능성이 크기 때문에 오류 진단과 수정이 간단해진다.
• 날이 갈수록, 더 많은 소프트웨어가 통합되어 잘 동작하는 것을 보게 된다. 이러한 상황은 팀의 사기를 높이며 관리자들은 프로젝트 진척도에 대해 잘 알게 된다.

① 베타 테스팅
② 스모크 테스팅
③ 회귀 테스팅
④ 복구 테스팅

정답 2번

출제 포인트

이 문제는 스모크 테스팅, 베타 테스팅, 회귀 테스팅, 복구 테스팅의 목적과 적용 상황을 구별할 수 있는지를 묻는다.

핵심 개념

스모크 테스팅: 매일 또는 빌드마다 주요 기능이 치명적 문제 없이 동작하는지 빠르게 확인하는 빌드 검증 테스트이다. 통합 직후의 큰 결함을 조기에 찾는 데 초점이 있다.
베타 테스팅: 실제 사용자 환경에서 사용자가 제품을 써 보며 출시 전 문제를 찾는 현장 테스트이다.
회귀 테스팅: 수정이나 변경 뒤에 기존 기능이 깨지지 않았는지 다시 확인하는 테스트이다.
복구 테스팅: 장애 발생 후 시스템이 백업, 복원, 재시작 과정을 통해 정상 상태로 회복되는지를 확인하는 테스트이다.

NoteBase

선지별 해설

1번 X: 베타 테스팅은 실제 사용자나 고객이 운영과 유사한 환경에서 제품을 사용해 보는 테스트이다. 보기처럼 매일 시행되고 새 increment와 관련된 통합 오류를 조기에 찾는 설명은 베타 테스팅이 아니라 스모크 테스팅에 해당한다.
2번 O: 스모크 테스팅은 빌드가 기본적으로 사용할 만한 상태인지 빠르게 확인하는 통합 지향 테스트이다. 매일 빌드에서 중대한 비호환성이나 새로 추가된 부분의 결함을 빨리 찾아 통합 위험을 줄이고, 오류 원인을 최근 increment에서 찾기 쉬운 점이 보기와 일치한다.
3번 X: 회귀 테스팅은 변경 후 기존 기능이 의도치 않게 깨지지 않았는지를 확인하는 테스트이다. 틀린 부분은 보기의 초점이 기존 기능 보존이 아니라 매일 빌드와 통합 오류 조기 발견에 있다는 점이다. 회귀 테스팅으로 고치면 변경 이후 기존 기능의 부작용 여부를 재검사한다는 내용이 들어가야 한다.
4번 X: 복구 테스팅은 장애나 고장 뒤에 시스템이 정상적으로 복구되는지를 확인하는 테스트이다. 틀린 부분은 보기가 장애 복원 절차가 아니라 빌드 단위의 통합 건전성 확인을 설명한다는 점이다. 복구 테스팅으로 고치면 백업 복원, 재기동, 장애 후 회복 능력을 점검한다는 내용이 되어야 한다.

오답 포인트

회귀 테스팅도 변경 후 다시 수행된다는 점 때문에 3번으로 오인하기 쉽다. 그러나 보기의 핵심 단서는 매일 시행, 통합 지향, 새 increment와 관련된 오류 발견이다. 이는 기존 기능 영향 확인보다 빌드 건전성 확인에 초점이 있는 스모크 테스팅의 특징이다.

다시 나오면 이렇게 판단

지문에서 매일 빌드, 빌드 직후, 통합 위험 감소, 새로 추가된 부분의 치명적 오류 조기 발견이 보이면 스모크 테스팅으로 판단한다. 변경이 기존 기능에 미친 영향을 다시 보는지, 실제 사용자가 현장에서 시험하는지, 장애 후 복원을 점검하는지를 함께 비교하면 다른 선택지와 구별된다.

암기 포인트

스모크는 빌드 통과 확인, 회귀는 변경 영향 확인, 베타는 사용자 현장 확인, 복구는 장애 복원 확인이다.

NoteBase

문 3. 구조적 방법론에서 모델링에 사용하는 표현 도구인 자료 흐름도(DFD: Data Flow Diagram)의 구성 요소가 아닌 것은?

① 자료 사전(Data Dictionary)

② 프로세스(Process)

③ 자료 저장소(Data Store)

④ 자료 흐름(Data Flow)

정답 1번

출제 포인트

이 문제는 자료 흐름도(DFD)의 기본 구성 요소와 자료 사전의 역할을 구분하는 문제이다.

핵심 개념

DFD의 기본 구성 요소는 프로세스, 자료 흐름, 자료 저장소, 외부 개체이다. 자료 사전은 DFD에 등장하는 데이터의 의미와 구조를 정의하는 보조 명세 문서이지, DFD 자체의 구성 요소는 아니다.

선지별 해설

① X
자료 사전은 데이터 항목, 자료 흐름, 자료 저장소의 의미와 구조를 정의하는 문서이다.
틀린 부분은 "DFD의 구성 요소"라는 판단이다.
올바르게는 "자료 사전은 DFD를 보완하는 명세 도구이며 DFD의 기본 구성 요소는 아니다"이다.

② O
프로세스는 입력 데이터를 처리하여 출력으로 바꾸는 기능 단위이므로 DFD의 기본 구성 요소이다.

③ O
자료 저장소는 파일이나 데이터베이스처럼 데이터가 보관되는 위치를 나타내므로 DFD의 구성 요소이다.

④ O
자료 흐름은 프로세스, 자료 저장소, 외부 개체 사이를 이동하는 데이터를 나타내므로 DFD의 구성 요소이다.

오답 포인트

자료 사전도 구조적 분석에서 함께 사용되므로 같은 수준의 도형 요소로 착각하기 쉽다. 그러나 자료 사전은 DFD 안에 그리는 기호가 아니라 DFD를 설명하는 보조 문서라는 점이 함정이다.

NoteBase

다시 나오면 이렇게 판단

DFD 문제에서는 그림 안에서 기호로 직접 표현되는 것인지, 아니면 DFD를 설명하는 보조 문서인지부터 구분하면 된다. 프로세스, 자료 흐름, 자료 저장소, 외부 개체는 구성 요소이고 자료 사전은 보조 명세이다.

암기 포인트

DFD는 프로세스, 자료 흐름, 자료 저장소, 외부 개체이고 자료 사전은 설명서이다.

NoteBase

문 4. 비기능적 요구사항에 대한 설명으로 가장 옳지 않은 것은?

① 시스템에서 사용하는 최대 메모리는 1GB를 넘지 않아야 한다.

② 시스템은 초당 100개 이상의 트랜잭션을 처리할 수 있어야 한다.

③ 시스템의 가용 상태는 99.0% 이상이어야 하며 다운 시간이 1분을 초과할 수 없다.

④ 시스템의 현재 상황에 대한 음성 안내 기능을 제공해야 한다.

정답 4번

출제 포인트

이 문제는 비기능 요구사항과 기능 요구사항을 구분하는 기준을 묻는 문제이다.

핵심 개념

기능 요구사항은 시스템이 무엇을 해야 하는지를 정의한 요구사항이다.
비기능 요구사항은 그 기능을 어느 수준의 성능, 자원 사용, 가용성, 신뢰성, 응답시간으로 제공해야 하는지를 정의한 품질 요구사항이다.

선지별 해설

① O: 최대 메모리를 1GB 이하로 제한한 것은 자원 사용에 대한 제약이다. 시스템이 어떤 기능을 수행하는지가 아니라 어떤 한계 안에서 동작해야 하는지를 제시하므로 비기능 요구사항이다.
② O: 초당 100개 이상의 트랜잭션 처리 능력은 처리량 성능 요구이다. 성능 수준을 수치로 제시한 전형적인 비기능 요구사항이다.
③ O: 가용 상태 99.0% 이상, 다운 시간 제한은 가용성과 신뢰성 수준을 수치로 표현한 요구사항이다. 품질 속성을 정한 것이므로 비기능 요구사항이다.
④ X: 틀린 부분은 '현재 상황에 대한 음성 안내 기능을 제공해야 한다'이다. 이는 시스템이 제공해야 할 기능 자체를 말하므로 기능 요구사항이다. 비기능 요구사항으로 고치면 '음성 안내는 1초 이내에 제공되어야 한다'처럼 기능의 품질이나 제약을 명시하는 형태가 된다.

오답 포인트

기능을 설명하면서 '해야 한다'로 끝나면 모두 요구사항처럼 보여 혼동하기 쉽다. 그러나 비기능 요구사항은 메모리, 처리량, 가용성처럼 기능을 수행하는 품질 기준을 말하고, 음성 안내 제공처럼 서비스 자체를 추가하는 내용은 기능 요구사항이다.

다시 나오면 이렇게 판단

선지가 시스템이 무엇을 제공하는지 설명하면 기능 요구사항으로 보고, 얼마나 빠르게·안정적으로·적은 자원으로 동작해야 하는지 설명하면 비기능 요구사항으로 판단한다.

NoteBase

암기 포인트

무엇을 한다는 기능 요구사항이고, 얼마나 잘해야 한다는 비기능 요구사항이다.

NoteBase

문 5. PMBOK(프로젝트관리지식체계)의 9가지 관점 중 프로젝트 통합관리에 해당하지 않는 것은?

① 프로젝트 계획 개발

② 일정 통제

③ 프로젝트 계획 실행

④ 통합된 변경 통제

정답 2번

출제 포인트

이 문제는 PMBOK의 9개 지식영역 중 프로젝트 통합관리와 일정관리의 소속 프로세스를 구분하는 문제이다.

핵심 개념

프로젝트 통합관리는 프로젝트 전체를 하나로 묶어 계획·실행·변경을 조정하는 관리영역이다. 키워드는 전체 통합, 프로젝트 계획 개발, 프로젝트 계획 실행, 통합된 변경 통제이다.
일정 통제는 프로젝트 시간관리의 프로세스로서 작업 일정의 변경 여부를 감시하고 일정 기준선을 통제하는 활동이다. 키워드는 일정, 진도, 시간관리이다.

선지별 해설

① O: 프로젝트 계획 개발은 개별 관리계획을 통합하여 전체 프로젝트 계획을 수립하는 통합관리 프로세스이므로 옳다.
② X: 일정 통제는 일정의 변경과 진척을 관리하는 시간관리 프로세스이므로 통합관리에 해당하지 않는다. 틀린 표현은 '프로젝트 통합관리'이고, 올바르게는 '프로젝트 시간관리의 일정 통제'이다.
③ O: 프로젝트 계획 실행은 수립된 프로젝트 계획에 따라 작업을 수행하도록 조정하는 통합관리 프로세스이므로 옳다.
④ O: 통합된 변경 통제는 프로젝트 전반의 변경 요청을 종합적으로 검토하고 승인 여부를 관리하는 통합관리 프로세스이므로 옳다.

오답 포인트

일정 통제도 프로젝트 전체 일정에 영향을 주므로 통합관리처럼 보이기 쉽다. 그러나 PMBOK는 전체 변경을 종합 조정하는 영역과 일정 자체를 통제하는 영역을 구분하므로, 일정이라는 단어가 보이면 시간관리로 분류된다.

다시 나오면 이렇게 판단

PMBOK 문제에서 계획 수립, 계획 실행, 통합된 변경 통제처럼 프로젝트 전체를 묶는 표현은 통합관리로 판단한다. 반대로 일정, 원가, 품질, 위험처럼 특정 관리대상을 직접 말하면 해당 개별 지식영역으로 판단한다.

NoteBase

암기 포인트

통합관리는 전체를 묶고, 일정 통제는 시간을 다룬다.

NoteBase

문 6. <보기>의 클래스 다이어그램을 작성하는 과정을 순서대로 바르게 나열한 것은?

<보기>
ㄱ. 가장 확실한 일반화 관계부터 시작하여 파악한다.
ㄴ. 클래스의 주요 임무를 찾아 나열한다.
ㄷ. 클래스 후보 집합을 파악한다.
ㄹ. 가장 중요한 클래스부터 시작하여 연관 관계와 꼭 필요한 속성을 추가한다.
ㅁ. 임무를 기준으로 필요한 오퍼레이션을 결정한다.

① ㄱ - ㄴ - ㄷ - ㄹ - ㅁ

② ㄱ - ㄹ - ㅁ - ㄴ - ㄷ

③ ㄷ - ㄴ - ㄱ - ㅁ - ㄹ

④ ㄷ - ㄹ - ㄱ - ㄴ - ㅁ

정답 4번

출제 포인트

이 문제는 클래스 다이어그램을 점진적으로 작성할 때의 표준 절차 흐름을 묻는 문제이다.

올바른 순서

1. ㄷ 클래스 후보 집합을 파악한다. - 다이어그램에 등장할 대상 클래스를 먼저 정리하는 단계이다.
2. ㄹ 가장 중요한 클래스부터 시작하여 연관 관계와 꼭 필요한 속성을 추가한다. - 후보 클래스 중 핵심 구조를 먼저 세우는 단계이다.
3. ㄱ 가장 확실한 일반화 관계부터 시작하여 파악한다. - 핵심 클래스가 잡힌 뒤 상속과 일반화 구조를 정리하는 단계이다.
4. ㄴ 클래스의 주요 임무를 찾아 나열한다. - 각 클래스가 무엇을 담당하는지 책임을 부여하는 단계이다.
5. ㅁ 임무를 기준으로 필요한 오퍼레이션을 결정한다. - 책임이 정해진 뒤 그 책임을 수행할 기능을 메서드 수준으로 정리하는 단계이다.

왜 이 순서: 오퍼레이션은 임무에서 나오고, 임무는 어떤 클래스가 어떤 구조로 존재하는지가 정해져야 붙일 수 있으므로 구조 파악이 책임과 기능 결정의 전제이다.

NoteBase

선지별 해설

1번 X: ㄱ을 맨 앞에 둔 것이 틀리다. 일반화 관계는 클래스 후보와 핵심 클래스의 연관 관계가 어느 정도 정리된 뒤에 파악해야 한다.
틀린 부분: 일반화 관계를 클래스 후보 파악보다 앞세웠다.
올바르게는: ㄷ-ㄹ-ㄱ-ㄴ-ㅁ이다.

2번 X: ㄱ으로 시작한 것도 틀리고, ㅁ을 ㄴ보다 앞세운 것도 틀리다. 오퍼레이션은 클래스의 주요 임무가 먼저 정리되어야 결정할 수 있다.
틀린 부분: 일반화 관계를 첫 단계에 두고, 오퍼레이션 결정을 임무 나열보다 앞에 두었다.
올바르게는: ㄷ-ㄹ-ㄱ-ㄴ-ㅁ이다.

3번 X: ㄴ을 너무 앞에 둔 것이 틀리다. 클래스의 주요 임무는 핵심 클래스와 관계, 일반화 구조가 잡힌 뒤에 붙이는 것이 자연스럽다. 또 ㅁ을 ㄹ보다 앞에 둔 것도 틀리다.
틀린 부분: 임무 나열과 오퍼레이션 결정을 구조 정리보다 앞세웠다.
올바르게는: ㄷ-ㄹ-ㄱ-ㄴ-ㅁ이다.

4번 O: 클래스 후보를 찾고, 핵심 클래스의 연관 관계와 속성을 잡은 뒤, 일반화 관계를 정리하고, 그 다음 책임과 오퍼레이션을 결정하는 올바른 순서이다.

오답 포인트

임무와 오퍼레이션을 빨리 정하고 싶어지는 점이 함정이다. 그러나 클래스 다이어그램은 먼저 클래스 후보와 구조를 잡고 그 다음 책임과 기능을 붙이는 순서로 진행한다. 특히 ㅁ은 반드시 ㄴ 뒤에 와야 한다는 점이 자주 흔들리는 지점이다.

다시 나오면 이렇게 판단

절차형 문제는 구조 요소를 먼저, 행위 요소를 나중에 배치하는 흐름으로 판단한다. 클래스 후보 파악 -> 핵심 클래스의 관계와 속성 정리 -> 일반화 정리 -> 책임 부여 -> 오퍼레이션 결정 순서로 보면 된다.

암기 포인트

클래스 다이어그램 작성 순서는 후보 클래스 -> 핵심 클래스 구조 -> 일반화 -> 임무 -> 오퍼레이션이다.

NoteBase

문 7. 맥콜(McCall)의 품질 요소 분류 중 제품 개선(product revision)에 해당하지 않는 것은?

① 유지보수 용이성(maintainability)

② 정확성(correctness)

③ 테스트 용이성(testability)

④ 유연성(flexibility)

정답 2번

출제 포인트

이 문제는 맥콜의 소프트웨어 품질 요인을 제품 운용, 제품 개선, 제품 전이로 정확히 구분할 수 있는지를 묻는 문제이다.

핵심 개념

제품 운용(product operation)은 실행 중 소프트웨어가 올바르고 효율적으로 동작하는 성질이다. 핵심 키워드는 정확성, 신뢰성, 효율성, 무결성, 사용성이다.
제품 개선(product revision)은 수정과 시험, 적응이 쉬운 성질이다. 핵심 키워드는 유지보수 용이성, 유연성, 테스트 용이성이다.
제품 전이(product transition)는 다른 환경으로 옮기거나 재사용하기 쉬운 성질이다. 핵심 키워드는 이식성, 재사용성, 상호운용성이다.

선지별 해설

1번 X: 유지보수 용이성은 제품 개선에 해당한다. 수정과 보완을 쉽게 할 수 있는 성질이므로 이 문제의 정답이 아니다. 틀린 표현은 '해당하지 않는 것'에 대한 판단이다. 올바르게는 제품 개선에 해당한다.
2번 O: 정확성은 제품 운용에 속한다. 소프트웨어가 요구사항대로 올바른 결과를 내는 성질이므로 제품 개선에는 해당하지 않는다.
3번 X: 테스트 용이성은 제품 개선에 해당한다. 수정 후 검증이 쉬운 성질이므로 제품 개선 항목이다. 틀린 표현은 '해당하지 않는 것'에 대한 판단이다. 올바르게는 제품 개선에 해당한다.
4번 X: 유연성은 제품 개선에 해당한다. 요구 변화에 맞추어 수정하기 쉬운 성질이므로 제품 개선 항목이다. 틀린 표현은 '해당하지 않는 것'에 대한 판단이다. 올바르게는 제품 개선에 해당한다.

오답 포인트

정확성과 유연성을 모두 막연한 좋은 품질 특성으로만 외우면 분류가 흔들린다. 정확성은 운용 중 올바르게 동작하는 성질이고, 유연성은 수정과 변화 대응이 쉬운 성질이라는 차이를 놓치면 4번과 2번을 바꾸어 고르기 쉽다.

NoteBase

다시 나오면 이렇게 판단

선지의 품질요인이 실행 중 성능과 결과의 올바름을 말하면 제품 운용으로 본다. 수정, 시험, 변경 대응의 쉬움을 말하면 제품 개선으로 본다. 이동, 재사용, 연동을 말하면 제품 전이로 본다.

암기 포인트

운용은 정확하게 돌리는 것, 개선은 쉽게 고치는 것, 전이는 쉽게 옮기는 것이다.

NoteBase

문 8. 소프트웨어 아키텍처를 잘 설계하기 위한 고려 사항에 대한 설명으로 가장 옳지 않은 것은?

① 시스템 품질 속성을 결정할 수 있어야 한다.

② 의사소통 도구로 활용할 수 있어야 한다.

③ 구현에 대한 제약 사항은 고려하지 않아도 된다.

④ 재사용할 수 있게 설계해야 한다.

정답 3번

출제 포인트

이 문제는 소프트웨어 아키텍처 설계 시 고려해야 하는 품질 속성, 의사소통 수단, 구현 제약, 재사용성의 의미를 구분하는 문제이다.

핵심 개념

소프트웨어 아키텍처는 시스템의 큰 구조와 구성요소 사이의 관계를 정하고, 품질 속성과 제약 사항을 초기 단계에서 반영하는 설계이다.
품질 속성은 성능, 유지보수성, 신뢰성처럼 아키텍처가 달성해야 할 비기능 요구이다.
구현 제약은 사용할 기술, 플랫폼, 예산, 조직 환경처럼 설계가 반드시 고려해야 하는 현실 조건이다.
재사용성은 기존 구조나 컴포넌트를 다른 시스템에도 활용할 수 있게 만드는 성질이다.

NoteBase

기출OX 해설집 · 공무원/서울시/7급 · 소프트웨어공학 · 2019 · 10월

선지별 해설

① O
시스템 품질 속성은 아키텍처 설계의 핵심 입력이다. 성능, 확장성, 보안성, 유지보수성 같은 속성을 충족하도록 구조를 정해야 하므로 옳은 설명이다.

② O
소프트웨어 아키텍처는 개발자, 설계자, 이해관계자 사이의 공통 청사진 역할을 한다. 따라서 구조를 설명하고 합의하는 의사소통 도구로도 활용할 수 있으므로 옳다.

③ X
틀린 부분은 "구현에 대한 제약 사항은 고려하지 않아도 된다"이다.
실제로는 구현 제약 사항을 반드시 고려해야 한다. 적용 가능한 기술, 운영 환경, 비용, 조직 표준을 무시하면 아키텍처가 현실적으로 구현되지 못하므로 이 선지가 정답이다.

④ O
재사용할 수 있게 설계하는 것은 아키텍처의 중요한 목적 중 하나이다. 공통 컴포넌트나 패턴을 재사용 가능하게 설계하면 개발 효율과 유지보수성이 높아질 수 있으므로 옳다.

오답 포인트

③은 아키텍처를 추상적인 개념 설계로만 생각하면 속기 쉽다. 그러나 아키텍처는 추상 구조를 정하는 동시에 실제 구현 환경의 제약까지 반영해야 한다는 점을 놓치면 오답이 된다.

다시 나오면 이렇게 판단

아키텍처 관련 선지는 먼저 비기능 요구와 구현 제약을 반영하는 상위 설계인지부터 본다. 품질 속성, 의사소통, 재사용성은 대체로 맞는 방향이고, 제약을 무시해도 된다고 하면 틀린 선지로 판단할 수 있다.

암기 포인트

아키텍처는 품질 속성을 반영하고, 구현 제약을 고려하며, 의사소통과 재사용의 기준이 되는 상위 구조이다.

NoteBase

문 9. 소프트웨어에 대한 설명으로 가장 옳지 않은 것은?

① 소프트웨어는 제조되는 것이 아니고, 개발되는 것이다.

② 소프트웨어란 프로그램과 프로그램의 개발, 운용, 보수에 필요한 모든 관련 정보를 말한다.

③ 소프트웨어는 수학이나 물리학처럼 규칙적이고 정형적인 구조를 가진다.

④ 소프트웨어는 그 생산물의 구조가 코드 안에 숨겨져 있는 비가시성(invisibility)을 갖는다.

정답 3번

출제 포인트

이 문제는 소프트웨어의 본질적 특성인 개발성, 비가시성, 관련 정보 포함 여부를 구분할 수 있는지를 묻는 문제이다.

핵심 개념

소프트웨어는 물리 제품처럼 제조되는 대상이 아니라 설계와 구현을 통해 개발되는 논리적 산물이다.
소프트웨어의 범위에는 실행 프로그램만이 아니라 개발·운영·유지보수에 필요한 문서와 관련 정보도 포함된다.
소프트웨어는 구조가 눈에 바로 보이지 않는 비가시성을 가지므로 수학식이나 물리 구조물처럼 정형적으로 드러난 구조를 가진다고 보기 어렵다.

NoteBase

기출OX 해설집 · 공무원/서울시/7급 · 소프트웨어공학 · 2019 · 10월

선지별 해설

① O
소프트웨어는 공장에서 동일한 물건을 반복 생산하는 제조물이 아니라 분석, 설계, 코딩, 테스트를 거쳐 만들어지는 개발 산물이라는 설명이므로 옳다.

② O
소프트웨어를 프로그램 자체로만 보지 않고 그 프로그램의 개발, 운용, 보수에 필요한 관련 정보까지 포함하는 설명이므로 옳다.

③ X
틀린 부분은 "수학이나 물리학처럼 규칙적이고 정형적인 구조"이다.
소프트웨어는 비가시적이고 복잡한 논리 구조를 가지므로 물리적 대상처럼 정형 구조가 눈에 드러난다고 보기 어렵다.
올바르게는 "소프트웨어는 수학이나 물리적 구조물처럼 정형적으로 보이는 구조를 가지지 않으며, 비가시성과 복잡성을 가진다"이다.

④ O
소프트웨어는 구조가 코드 내부 논리로 표현되어 눈에 보이는 형태로 바로 드러나지 않는 비가시성을 가진다는 설명이므로 옳다.

오답 포인트

③은 "규칙적이고 정형적"이라는 표현으로 물리적 구조물의 특성을 소프트웨어에 뒤집어 씌운 선택지이다.
소프트웨어의 핵심 특성은 오히려 비가시성과 복잡성이므로, 구조가 명확히 보일 것처럼 서술하면 틀린 선지가 된다.

다시 나오면 이렇게 판단

소프트웨어 특성 문제는 제조 여부, 관련 문서 포함 여부, 비가시성·복잡성 여부를 먼저 확인하면 된다. 물리 제품의 성질인 정형성·가시성이 소프트웨어에 붙어 있으면 우선적으로 의심할 수 있다.

암기 포인트

소프트웨어는 제조가 아니라 개발, 프로그램만이 아니라 관련 정보까지 포함, 구조는 정형적이 아니라 비가시적이라는 점이 핵심이다.

NoteBase

문 10. LOC 기법에 의해 예측된 총 라인 수가 60,000라인이고, 개발자는 5명이 참여한다. 개발자들이 1인당 월 평균 600라인을 코딩할 때, 개발 기간은?

① 10개월

② 15개월

③ 20개월

④ 25개월

정답 3번

출제 포인트

이 문제는 LOC 추정치와 인당 월 코딩 생산성을 이용해 전체 개발 기간을 계산하는 문제이다.

조건 정리

주어진 값: 총 라인 수 60,000라인, 개발자 5명, 1인당 월 600라인이다.
구해야 할 값: 프로젝트 개발 기간이다.
주의할 조건: 1인당 생산성을 그대로 쓰지 않고 팀 전체 생산성으로 합산해야 한다.

사용할 공식

팀의 월 생산성은 개발자 수 × 1인당 월 생산성이다. 개발 기간은 총 라인 수 ÷ 팀의 월 생산성이다.

이 공식을 쓰는 이유:
문제는 전체 작업량과 인원별 월 처리량이 주어졌으므로, 먼저 팀 전체가 한 달에 처리하는 LOC를 구한 뒤 총 작업량을 나누면 기간이 나온다.

계산 과정

1. 팀의 월 생산성을 구하면 $5 \times 600 = 3{,}000$라인/월이다.
2. 개발 기간을 구하면 $60{,}000 \div 3{,}000 = 20$개월이다.
3. 계산 결과가 20개월이므로 보기 ③과 일치한다.

NoteBase

선지별 해설

① X: 10개월은 계산 결과와 다르다. 틀린 표현은 "10개월"이며, 올바르게는 "20개월"이다. 개발자 5명의 월 생산성은 3,000라인이므로 10개월 동안 처리 가능한 양은 30,000라인뿐이다.
② X: 15개월은 계산 결과와 다르다. 틀린 표현은 "15개월"이며, 올바르게는 "20개월"이다. 15개월이면 처리량은 45,000라인이어서 60,000라인에 못 미친다.
③ O: 팀의 월 생산성은 3,000라인이고, 60,000 ÷ 3,000 = 20이므로 20개월이 맞다.
④ X: 25개월은 계산 결과와 다르다. 틀린 표현은 "25개월"이며, 올바르게는 "20개월"이다. 25개월이면 처리량이 75,000라인이 되어 주어진 총 라인 수를 초과한다.

오답 포인트

개발 기간 문제에서는 1인당 생산성과 팀 전체 생산성을 혼동하기 쉽다. 이 문항의 함정은 600라인/월을 바로 총 라인 수에 나누거나, 인원 수를 잘못 반영해 10개월·15개월·25개월처럼 어긋난 값을 고르게 만드는 데 있다.

다시 나오면 이렇게 판단

LOC 기반 기간 계산은 총 작업량을 바로 나누지 않고, 먼저 개발자 수와 1인당 월 생산성을 곱해 팀의 월 생산성을 만든 뒤 총 라인 수를 나누어 판단한다.

암기 포인트

개발 기간 = 총 LOC ÷ (개발자 수) × (1인당 월 LOC)이다.

NoteBase

문 11. 객체지향 소프트웨어 개발을 위한 UP(Unified Process)의 개발 단계에 속하지 않는 것은?

① 도입 단계(inception phase)

② 구축 단계(construction phase)

③ 검증 단계(validation phase)

④ 전이 단계(transition phase)

정답 3번

출제 포인트

UP의 표준 4단계인 도입, 정련, 구축, 전이 단계를 정확히 구분하는 문제이다.

핵심 개념

도입 단계는 프로젝트의 목표, 범위, 타당성을 정하는 단계이다. 정련 단계는 요구사항을 구체화하고 핵심 아키텍처를 확정하는 단계이다. 구축 단계는 구현과 테스트를 반복하며 제품을 완성하는 단계이다. 전이 단계는 완성된 시스템을 사용자 환경에 배포하고 정착시키는 단계이다. 검증 단계는 UP의 표준 4단계 명칭이 아니며, 이 자리에 들어가는 공식 명칭은 정련 단계(elaboration phase)이다.

선지별 해설

① X
도입 단계는 UP의 실제 단계이므로 '속하지 않는 것'에 해당하지 않는다. 프로젝트의 비전과 범위를 정하는 초반 단계이다. 올바르게는 UP 개발 단계에 속한다.

② X
구축 단계는 UP의 실제 단계이므로 정답 선택지가 아니다. 구현과 통합을 통해 시스템을 완성해 가는 단계이다. 올바르게는 UP 개발 단계에 속한다.

③ O
검증 단계(validation phase)라는 명칭은 UP의 표준 4단계에 없다. 테스트와 검증 활동이 존재한다고 해서 그것이 공식 단계명이 되는 것은 아니다. 올바르게는 검증 단계가 아니라 정련 단계(elaboration phase)이다.

④ X
전이 단계는 UP의 실제 단계이므로 '속하지 않는 것'이 아니다. 완성된 시스템을 사용자에게 배포하고 운영 환경에 안착시키는 단계이다. 올바르게는 UP 개발 단계에 속한다.

NoteBase

오답 포인트

validation이라는 일반 용어 때문에 ③을 테스트 전용 단계처럼 착각하기 쉽다. 이 문항은 활동 이름을 묻는 것이 아니라 UP의 공식 단계 명칭을 묻는 문제이므로, 정련 단계가 빠져 있다는 점을 잡아내야 한다.

다시 나오면 이렇게 판단

UP 문제는 먼저 공식 4단계를 순서대로 대조하면 풀린다. 도입-inception, 정련-elaboration, 구축-construction, 전이-transition이 완성되면, 그 밖의 검증·분석 같은 일반 용어는 비공식 단계명으로 판단하면 된다.

암기 포인트

UP 4단계는 도입-정련-구축-전이이며, 검증은 활동일 수 있어도 공식 단계명은 아니다.

NoteBase

문 12. <보기>는 소프트웨어 공학 계층에 대한 설명이다. (가) ~ (다)의 설명에 해당하는 용어를 옳게 짝지은 것은?

<보기>
(가) 기술 계층을 묶어주는 접착제 역할을 하는 것이고 소프트웨어를 합리적으로 빨리 개발할 수 있도록 한다.
(나) 소프트웨어를 개발하기 위한 전문적인 실용기술을 제공한다.
(다) 소프트웨어 개발의 각 계층에 사용된 개념들에 대해 자동적 또는 반자동적 지원을 한다.

① (가): 방법(methods), (나): 프로세스(process), (다): 도구(tool)
② (가): 도구(tool), (나): 방법(methods), (다): 프로세스(process)
③ (가): 프로세스(process), (나): 도구(tool), (다): 방법(methods)
④ (가): 프로세스(process), (나): 방법(methods), (다): 도구(tool)

정답 4번

출제 포인트

이 문제는 소프트웨어 공학의 3요소인 프로세스, 방법, 도구의 역할을 구분하는 문제이다.

핵심 개념

프로세스는 개발 활동을 묶어 주는 작업 골격으로서 기술 계층을 연결하는 틀이다. 키워드는 접착제, 개발 순서, 관리 틀이다.
방법은 소프트웨어를 만들기 위한 구체적이고 전문적인 실무 기법이다. 키워드는 기법, 절차, 설계·분석 방법이다.
도구는 프로세스와 방법을 실제 개발에서 자동 또는 반자동으로 지원하는 수단이다. 키워드는 CASE, 자동화 지원, 반자동 지원이다.

NoteBase

선지별 해설

① X: (가)를 방법으로 본 것이 틀리다. (가)의 핵심 표현은 "기술 계층을 묶어주는 접착제 역할"이므로 프로세스에 해당한다. (나)를 프로세스로 본 것도 틀리다. 전문적인 실용기술 제공은 방법에 해당한다. ①은 올바르게는 (가) 프로세스, (나) 방법, (다) 도구가 되어야 한다.
② X: (가)를 도구로 본 것이 틀리다. 도구는 각 계층의 개념을 자동적 또는 반자동적으로 지원하는 수단이므로 (다)에 들어가야 한다. ②는 올바르게는 (가) 프로세스, (나) 방법, (다) 도구가 되어야 한다.
③ X: (나)를 도구로, (다)를 방법으로 바꿔 놓은 것이 틀리다. (나)의 "전문적인 실용기술"은 방법의 정의이고, (다)의 "자동적 또는 반자동적 지원"은 도구의 정의이다. ③은 올바르게는 (가) 프로세스, (나) 방법, (다) 도구가 되어야 한다.
④ O: (가)=프로세스, (나)=방법, (다)=도구의 대응이 모두 맞다. 프로세스는 계층을 연결하는 틀이고, 방법은 기술적 방법론이며, 도구는 이를 지원하는 자동화 수단이다.

오답 포인트

방법과 도구를 모두 개발에 쓰이는 것이라고만 외우면 (나)와 (다)를 쉽게 바꿔 적게 된다. 이 문항은 "전문적인 실용기술"은 방법, "자동적 또는 반자동적 지원"은 도구라는 표현 차이를 이용해 함정을 만든 문제이다.

다시 나오면 이렇게 판단

문장에 개발 전체를 묶는 틀, 접착제, 절차의 골격이 나오면 프로세스로 본다. 구체적인 개발 기법이나 실무 기술이 나오면 방법으로 본다. 자동화, 반자동화, 지원 시스템이 나오면 도구로 판단한다.

암기 포인트

프로세스는 틀, 방법은 기법, 도구는 지원 수단이다.

NoteBase

문 13. PERT/CPM에 대한 설명으로 가장 옳지 않은 것은?

① 임계 경로는 그래프에서 작업 진행에 여유 시간이 없는 경로를 말한다.

② 전체 프로젝트 완료 시간을 단축하고 싶다면 임계 경로의 작업시간을 줄여야 한다.

③ 병행 작업을 계획할 수 있다.

④ 그래프에서 가능한 전체 작업 시간의 합이 가장 작은 경로의 작업들을 잘 관리하여야 한다.

정답 4번

출제 포인트

이 문제는 PERT/CPM에서 임계 경로의 의미와 프로젝트 기간 단축 원리를 구분하는 문제이다.

핵심 개념

PERT/CPM은 작업들의 선후관계를 네트워크로 표현하여 일정 계획과 통제를 수행하는 기법이다. 결정 차이축은 어떤 경로가 전체 프로젝트 완료시간을 결정하는가이다. 키워드는 네트워크, 병행 작업, 일정 통제이다.
임계 경로는 시작부터 종료까지 가능한 경로들 중 총 소요시간이 가장 길어 전체 프로젝트 완료시간을 결정하는 경로이다. 키워드는 최장 경로, 여유시간 0, 완료시간 결정이다.
비임계 경로는 여유시간이 있어 일부 지연이 곧바로 전체 완료시간 지연으로 이어지지 않는 경로이다. 키워드는 여유시간, 일정 조정 가능이다.

선지별 해설

① O. 임계 경로는 총여유시간이 0인 작업들로 이어진 경로이므로 작업 진행에 여유 시간이 없는 경로라는 설명이 맞다.
② O. 전체 프로젝트 기간은 임계 경로가 결정하므로 완료 시간을 단축하려면 임계 경로 위 작업의 소요시간을 줄여야 한다.
③ O. PERT/CPM은 작업의 선후관계를 네트워크로 나타내므로 동시에 진행 가능한 작업을 찾아 병행 작업을 계획할 수 있다.
④ X. 틀린 부분은 '전체 작업 시간의 합이 가장 작은 경로'이다. 실제로는 전체 작업 시간의 합이 가장 큰 경로, 즉 프로젝트 완료시간을 결정하는 임계 경로의 작업을 중점 관리하여야 한다.

오답 포인트

임계 경로를 단순히 중요한 경로라고만 외우면 ④처럼 최단 경로나 관리가 쉬운 경로로 오해하기 쉽다. 임계 경로는 중요해서가 아니라 전체 완료시간을 결정하는 최장 경로이기 때문에 관리 대상이 된다.

NoteBase

다시 나오면 이렇게 판단

PERT/CPM 문제에서는 먼저 임계 경로가 최장 경로인지와 여유시간이 0인지 확인한다. 프로젝트 기간 단축 여부를 묻는 선택지는 임계 경로 작업의 단축 여부로 판단한다.

암기 포인트

임계 경로는 최장 경로이고 여유시간이 0이며 프로젝트 완료시간을 결정하는 경로이다.

NoteBase

문 14. 그래프에서 신뢰성(ROCOF)은 소프트웨어 결함 발생 비율을 의미한다. <보기>에서 설명하는 모델에 해당하는 그래프는?

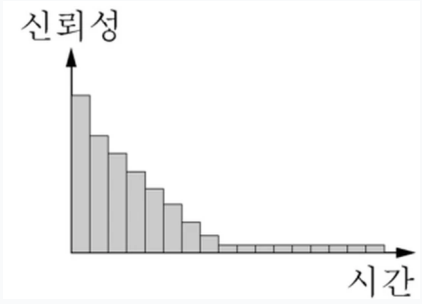
<보기>

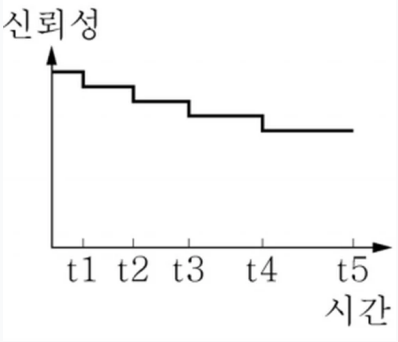
• 소프트웨어 결함을 수정함에 따라 동일한 정도로 신뢰성이 향상되지 않고 랜덤 분포에 따라 변화한다.

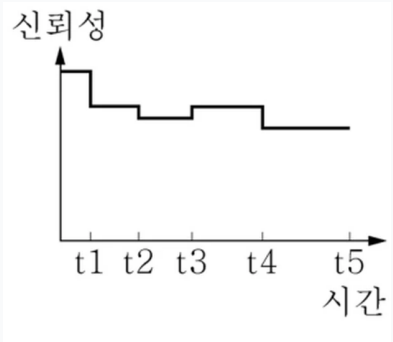
• 디버깅 동안 소프트웨어 결함이 항상 수정되지는 않고, 때로는 새로운 결함이 삽입된다.

• 결함의 수정에 따라 새로운 결함이 야기되면 신뢰성이 감소한다.

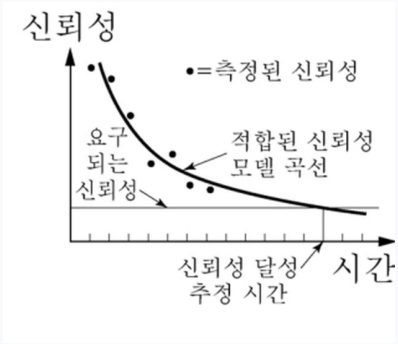
• 자주 발생하는 결함은 테스팅 프로세스 초기에 발견되는 경향이 있기 때문에, 결함이 수정됨에 따라 평균적인 신뢰성 개선 정도가 감소된다.

① 

② 

③ 

NoteBase

④ 

정답 3번

출제 포인트

이 문제는 결함 수정이 항상 같은 크기로 ROCOF를 낮추지 않고, 불완전 디버깅과 새 결함 삽입 때문에 전체적으로는 감소하되 중간에 다시 증가할 수도 있는 계단형 그래프를 식별하는 문제이다.

선지별 해설

① X: 막대가 시간에 따라 매끈하게 줄어드는 형태는 결함 발생 비율이 일방적으로 감소하는 분포형 표현에 가깝다. 보기처럼 수정 과정에서 새 결함이 들어가 ROCOF가 다시 나빠지는 반등이 드러나지 않는다. 올바르게는 감소 도중 일시적 상승이 섞인 계단형이어야 한다.
② X: 계단식으로 계속 내려가기만 하므로 수정할수록 ROCOF가 단조롭게 개선되는 모델이다. 보기의 핵심인 '때로는 새로운 결함이 삽입된다'는 상황이 반영되지 않았다. 올바르게는 일부 시점에서 ROCOF가 다시 높아지는 구간이 있어야 한다.
③ O: 전체적으로는 ROCOF가 감소하지만 중간에 다시 올라가는 구간이 있어, 결함 수정이 항상 성공하지 않고 새 결함이 유입될 수 있는 불완전 디버깅 상황을 잘 나타낸다. 또한 자주 발생하는 결함이 초기에 먼저 제거되어 시간이 갈수록 평균 개선폭이 줄어드는 모습도 계단형 변화로 해석할 수 있다.
④ X: 측정점에 곡선을 적합한 연속형 신뢰성 성장 곡선으로, 평균적인 추세를 매끈하게 표현한 모델이다. 보기처럼 수정 결과가 랜덤하게 흔들리고 중간 악화가 나타나는 실제 디버깅 과정을 직접 보여 주지 않는다. 올바르게는 실측 변화가 위아래로 흔들리는 비단조 계단형이어야 한다.

오답 포인트

보기의 첫 문장만 보고 '결함이 수정되니 계속 좋아진다'라고 읽으면 ②를 고르기 쉽다. 그러나 핵심 단서는 '항상 수정되지는 않고', '새로운 결함이 삽입된다', '신뢰성이 감소한다'는 부분이다. 이 단서가 있으면 그래프는 단조감소가 아니라 중간 반등이 있는 형태여야 한다.

다시 나오면 이렇게 판단

모델 설명에 불완전 디버깅, 랜덤 변화, 새 결함 삽입이 나오면 그래프에서 먼저 '전체 추세는 개선되지만 중간에 역행 구간이 있는가'를 본다. 반대로 계속 한 방향으로만 움직이는 매끈한 감소 곡선이나 단조 계단형은 제외한다.

NoteBase

암기 포인트

불완전 디버깅 ROCOF 그래프는 '대체로 하강, 중간 반등 가능'이다.

NoteBase

문 15. <보기>에서 설명하는 디자인 패턴은?

<보기>
• 객체 생성 인터페이스를 정의하기 위한 디자인 패턴이다.
• 상위클래스에서는 객체를 만드는 인터페이스를 정의하고, 하위클래스에서는 구체적인 인스턴스를 생성하도록 한다.
• 객체를 생성하는 인터페이스와 실제 객체를 생성하는 클래스를 분리할 수 있다.

① factory method 패턴
② prototype 패턴
③ builder 패턴
④ abstraction factory 패턴

정답 1번

출제 포인트

이 문제는 객체 생성 책임을 상위 클래스의 인터페이스와 하위 클래스의 구체 생성으로 분리하는 Factory Method 패턴의 핵심 구조를 묻는 문제이다.

핵심 개념

Factory Method는 상위 클래스가 생성 메서드의 틀을 정의하고 실제로 어떤 구체 객체를 만들지는 하위 클래스가 결정하는 패턴이다.
Prototype는 기존 객체를 복제하여 새 객체를 만드는 패턴이다.
Builder는 복잡한 객체를 단계적으로 조립하여 생성 과정과 표현을 분리하는 패턴이다.
Abstract Factory는 서로 관련된 객체들의 제품군을 일관된 방식으로 생성하는 인터페이스를 제공하는 패턴이다.

NoteBase

선지별 해설

① O: 상위 클래스가 객체 생성 인터페이스를 정의하고 하위 클래스가 구체 인스턴스를 생성한다는 설명이므로 Factory Method의 정의와 정확히 일치한다.
② X: 틀린 부분은 'prototype 패턴'이라는 판단이다. Prototype는 상속으로 생성 책임을 넘기는 패턴이 아니라 기존 객체를 복제해서 새 객체를 만드는 패턴이다.
③ X: 틀린 부분은 'builder 패턴'이라는 판단이다. Builder는 생성 인터페이스와 실제 생성 클래스를 상하위 클래스로 나누는 패턴이 아니라 복잡한 객체를 단계별로 조립하는 패턴이다.
④ X: 틀린 부분은 'abstract factory 계열 패턴'이라는 판단이다. Abstract Factory는 관련 있는 여러 제품군을 함께 생성하는 인터페이스를 제공하는 패턴이고, 보기처럼 하위 클래스가 단일 객체의 구체 인스턴스를 결정하는 구조와는 다르다.

오답 포인트

Factory Method와 Abstract Factory를 모두 '생성 패턴'으로만 외우면 ④로 흔들리기 쉽다. 이 문항의 핵심 단서는 '상위 클래스가 생성 인터페이스를 정의하고 하위 클래스가 구체 인스턴스를 만든다'는 상속 기반 생성 위임 구조이다.

다시 나오면 이렇게 판단

문장에서 하위 클래스가 어떤 객체를 만들지 결정한다고 나오면 Factory Method를 먼저 본다. 기존 객체 복제이면 Prototype, 단계적 조립이면 Builder, 관련 객체군을 한꺼번에 생성하면 Abstract Factory로 판단할 수 있다.

암기 포인트

Factory Method는 하위 클래스가 생성 대상을 정하고, Abstract Factory는 제품군 전체의 생성 인터페이스를 제공한다.

NoteBase

문 16. 소프트웨어의 설계에서 모듈 사이의 좋은 관계로 가장 옳지 않은 것은?

① 모듈 간에는 꼭 필요한 데이터만 주고받는 것이 좋다.

② 약한 결합을 유지하기 위해 인터페이스가 복잡하지 않아야 한다.

③ 가장 좋은 방법은 모듈 간의 결합도는 낮게, 응집도는 높게 하는 것이다.

④ 모듈 간의 인터페이스는 데이터보다 제어 플래그를 사용하는 것이 좋다.

정답 4번

출제 포인트

이 문제는 모듈 설계에서 좋은 관계를 만드는 기준인 낮은 결합도와 높은 응집도, 그리고 바람직한 인터페이스 성격을 묻는 문제이다.

핵심 개념

결합도는 한 모듈이 다른 모듈에 얼마나 의존하는지를 뜻하며 낮을수록 좋다.
응집도는 한 모듈 내부 기능들이 얼마나 한 목적에 밀접하게 모여 있는지를 뜻하며 높을수록 좋다.
자료 결합은 필요한 데이터만 주고받는 형태이고, 제어 결합은 제어 플래그를 넘겨 상대 모듈의 동작을 좌우하는 형태이므로 자료 결합이 더 바람직하다.

선지별 해설

① O
모듈 간에는 꼭 필요한 데이터만 주고받을수록 의존이 줄어들어 결합도가 낮아진다. 자료를 최소화하는 인터페이스는 좋은 설계 원칙에 맞다.

② O
약한 결합을 유지하려면 인터페이스가 단순하고 명확해야 한다. 인터페이스가 복잡해질수록 다른 모듈의 내부 사정에 더 많이 얽혀 결합도가 높아지기 쉽다.

③ O
좋은 모듈 설계의 대표 원칙은 결합도는 낮게, 응집도는 높게 하는 것이다. 모듈끼리 덜 얽히고 각 모듈은 자기 책임에 집중할수록 유지보수가 쉬워진다.

④ X
틀린 표현은 `데이터보다 제어 플래그를 사용하는 것`이다. 제어 플래그를 넘기면 상대 모듈의 처리 흐름을 외부에서 제어하게 되어 제어 결합이 생기므로 바람직하지 않다. 올바르게는 필요한 데이터만 단순하게 전달하는 인터페이스가 더 좋다.

NoteBase

오답 포인트

④는 `인터페이스`라는 말 때문에 그럴듯해 보이지만, 실제 함정은 `제어 플래그`에 있다. 인터페이스가 존재하는 것 자체가 아니라 그 인터페이스가 데이터 중심인지 제어 중심인지를 구분해야 한다.

다시 나오면 이렇게 판단

모듈 설계 문제는 먼저 `결합도는 낮을수록 좋다`, `응집도는 높을수록 좋다`를 기준으로 본다. 그다음 선지에 제어 플래그, 내부 로직 간섭, 불필요한 데이터 교환 같은 표현이 나오면 결합도를 높이는 요소인지 확인하면 된다.

암기 포인트

좋은 모듈은 `낮은 결합도, 높은 응집도, 단순한 데이터 인터페이스`로 정리된다.

NoteBase

문 17. 블랙 박스 테스트에서 발견할 수 없는 에러는?

① 인터페이스 에러

② 제어구조 에러

③ 외부 데이터베이스 접근 에러

④ 틀리거나 누락된 기능

정답 2번

출제 포인트

이 문제는 블랙박스 테스트가 외부 동작 관점에서 발견할 수 있는 오류와 내부 구현 구조를 봐야 발견되는 오류를 구별할 수 있는지를 묻는다.

핵심 개념

블랙박스 테스트는 내부 코드나 제어 흐름을 보지 않고 입력과 출력, 기능 요구사항, 인터페이스 동작이 맞는지를 확인하는 테스트이다. 화이트박스 테스트는 내부 로직, 분기, 반복, 경로, 제어구조가 올바른지를 코드 수준에서 확인하는 테스트이다. 따라서 기능 누락, 인터페이스 오류, 외부 자원 접근 오류는 블랙박스 테스트로 드러날 수 있지만, 제어구조 오류는 주로 화이트박스 테스트의 대상이다.

선지별 해설

1번: X. 인터페이스 에러는 블랙박스 테스트로 발견할 수 있는 대표적 오류이다. 입력 형식, 출력 형식, 화면이나 모듈 간 연동 결과가 명세와 다르면 외부 동작만 확인해도 드러난다. 틀린 표현은 '발견할 수 없는'에 해당한다고 본 점이다. 올바르게는 '인터페이스 에러는 블랙박스 테스트에서 발견할 수 있다'이다.
2번: O. 제어구조 에러는 분기, 반복, 경로 같은 내부 논리 구조의 오류이므로 코드를 보지 않는 블랙박스 테스트만으로는 직접 찾기 어렵다. 이런 오류는 화이트박스 테스트로 확인하는 것이 적절하다.
3번: X. 외부 데이터베이스 접근 에러는 데이터 조회나 저장 결과가 요구사항과 다르게 나타나면 입출력과 기능 수행 결과를 통해 확인할 수 있다. 틀린 표현은 이 오류를 블랙박스 테스트로 찾을 수 없다고 본 점이다.
올바르게는 '외부 데이터베이스 접근 에러는 블랙박스 테스트에서 발견될 수 있다'이다.
4번: X. 틀리거나 누락된 기능은 블랙박스 테스트가 가장 잘 찾는 오류 유형 중 하나이다. 요구사항에 있는 기능이 없거나 결과가 다르면 사용자 관점의 시험만으로도 바로 드러난다. 틀린 표현은 이 기능 오류를 발견 불가능한 대상으로 본 점이다. 올바르게는 '틀리거나 누락된 기능은 블랙박스 테스트에서 발견할 수 있다'이다.

오답 포인트

블랙박스 테스트도 실행 결과를 통해 여러 오류를 발견할 수 있으므로 1번, 3번, 4번이 모두 그럴듯해 보인다. 그러나 문제의 기준은 오류가 외부 동작으로 드러나는지, 내부 제어 로직을 봐야 하는지이다. '제어구조'라는 표현이 나오면 내부 구현 관점이라는 점을 먼저 잡아야 한다.

NoteBase

다시 나오면 이렇게 판단

선지의 오류가 요구사항, 입력과 출력, 인터페이스, 외부 연동 결과처럼 외부 동작에서 보이면 블랙박스 테스트 대상으로 본다. 반대로 분기, 반복, 경로, 조건문 같은 내부 흐름 자체의 오류이면 화이트박스 테스트 대상으로 판단한다.

암기 포인트

블랙박스는 기능과 결과, 화이트박스는 코드와 제어구조를 본다.

NoteBase

문 18. 요구사항 분석 모델 중 유스케이스(use case)가 속한 모델은?

① 시나리오 기반 모델(scenario based model)

② 클래스 모델(class model)

③ 행위 모델(behavioral model)

④ 플로우 모델(flow model)

정답 1번

출제 포인트

이 문제는 요구사항 분석 모델을 시나리오 기반, 클래스, 행위, 플로우 모델로 구분하고 유스케이스가 어느 범주에 속하는지를 묻는다.

핵심 개념

시나리오 기반 모델은 사용자와 시스템의 상호작용 시나리오를 기술하는 모델이며 대표 예가 유스케이스이다.
클래스 모델은 시스템의 정적 구조를 클래스, 속성, 연관관계로 표현하는 모델이다.
행위 모델은 상태 변화나 사건에 따른 동작을 표현하는 모델이다.
플로우 모델은 데이터나 처리의 흐름을 중심으로 기능 수행 과정을 표현하는 모델이다.

선지별 해설

1번 O: 유스케이스는 사용자가 시스템을 어떻게 이용하는지를 시나리오로 표현하므로 시나리오 기반 모델에 속한다.
2번 X: 틀린 부분은 유스케이스를 클래스 모델에 넣은 점이다. 클래스 모델은 클래스와 관계 같은 정적 구조를 표현하며, 유스케이스가 아니라 클래스 다이어그램이 여기에 해당한다.
3번 X: 틀린 부분은 유스케이스를 행위 모델에 넣은 점이다. 행위 모델은 상태 변화나 이벤트에 따른 동작을 나타내며, 실제로는 상태도나 활동 다이어그램 같은 표현이 여기에 가깝다.
4번 X: 틀린 부분은 유스케이스를 플로우 모델에 넣은 점이다. 플로우 모델은 데이터 흐름도처럼 정보와 처리의 흐름을 표현하며, 유스케이스는 흐름 자체보다 사용자 시나리오를 기술하는 모델이다.

오답 포인트

유스케이스에도 사건의 흐름과 동작 순서가 들어가므로 행위 모델이나 플로우 모델로 오해하기 쉽다. 그러나 분류 기준은 무엇을 중심으로 표현하느냐이며, 유스케이스는 사용자 시나리오 중심이라는 점이 핵심이다.

다시 나오면 이렇게 판단

선지에 유스케이스가 나오면 먼저 사용자와 시스템의 상호작용 시나리오를 기술하는지 본다. 시나리오 중심이면 시나리오 기반 모델, 정적 구조면 클래스 모델, 상태 변화면 행위 모델, 데이터 이동이면 플로우 모델로 판단된다.

NoteBase

암기 포인트

유스케이스는 사용자 시나리오, 클래스는 정적 구조, 행위는 상태 변화, 플로우는 데이터 흐름이다.

NoteBase

문 19. <보기>와 같은 프로세스 영역들이 속하는 CMMI의 Maturity Level은?

<보기>
• 요구사항 개발(requirement development)
• 제품 통합(product integration)
• 통합된 프로젝트 관리(integrated project management)
• 리스크 관리(risk management)

① Level 1
② Level 2
③ Level 3
④ Level 4

정답 3번

출제 포인트

CMMI 성숙도 단계별 프로세스 영역을 구분하는 문제이다.

핵심 개념

Level 1은 초기 단계로서 프로세스가 개인 역량에 의존하는 수준이다.
Level 2는 프로젝트 단위의 기본 관리 체계를 갖춘 관리 단계이다. 키워드는 요구사항 관리, 프로젝트 계획, 프로젝트 모니터링 및 통제, 형상관리이다.
Level 3는 조직 표준 프로세스를 정의하고 이를 프로젝트에 통합해 적용하는 정의 단계이다. 키워드는 요구사항 개발, 제품 통합, 통합된 프로젝트 관리, 리스크 관리이다.
Level 4는 프로세스 성과를 정량적으로 측정하고 통제하는 단계이다. 키워드는 조직 프로세스 성과, 정량적 프로젝트 관리이다.

NoteBase

선지별 해설

① X: Level 1은 초기 단계이므로 이런 식의 체계적인 프로세스 영역 묶음으로 설명되지 않는다. 제시된 요구사항 개발, 제품 통합, 통합된 프로젝트 관리, 리스크 관리는 Level 1이 아니라 Level 3의 영역이다.
② X: Level 2는 기본 프로젝트 관리 단계이므로 요구사항 관리, 프로젝트 계획, 프로젝트 모니터링 및 통제, 형상관리 등이 중심이다. 보기의 requirement development는 requirement management와 다른 개념이며, 제시된 묶음 전체는 Level 2가 아니라 Level 3이다.
③ O: 요구사항 개발, 제품 통합, 통합된 프로젝트 관리, 리스크 관리는 조직 차원의 표준 프로세스를 정립하고 프로젝트에 통합 적용하는 Level 3의 대표 영역이므로 옳다.
④ X: Level 4는 정량적 관리 단계이므로 조직 프로세스 성과와 정량적 프로젝트 관리처럼 수치 기반 통제가 핵심이다. 제시된 영역은 정량 관리 영역이 아니라 정의 단계인 Level 3 영역이다.

오답 포인트

가장 흔한 함정은 requirement development와 requirement management를 같은 것으로 보는 착각이다. 요구사항 개발은 Level 3이고 요구사항 관리는 Level 2이다. 또 제품 통합, 통합된 프로젝트 관리, 리스크 관리처럼 조직 표준 프로세스를 확장하는 표현이 보이면 Level 3 쪽으로 판단해야 한다.

다시 나오면 이렇게 판단

기본적인 일정·추적·형상관리처럼 프로젝트 관리의 기초를 묻으면 Level 2로 본다. 요구사항 개발, 제품 통합, 통합 프로젝트 관리, 리스크 관리처럼 조직 표준 프로세스의 정의와 통합을 묻으면 Level 3로 본다. 정량 측정과 통제를 묻으면 Level 4로 본다.

암기 포인트

Level 2는 관리, Level 3는 정의·통합, Level 4는 정량 관리이다.

NoteBase

문 20. 소프트웨어 아키텍처 유형 중 파이프 필터(pipe and filter) 구조가 속하는 것은?

① 데이터 중심 아키텍처(data-centered architecture)

② 데이터 흐름 아키텍처(data-flow architecture)

③ 계층 아키텍처(layered architecture)

④ 호출과 반환 아키텍처(call and return architecture)

정답 2번

출제 포인트

이 문제는 소프트웨어 아키텍처 유형을 데이터 저장 중심, 데이터 흐름 중심, 계층, 호출-반환 구조로 구분하는지를 묻는 문제이다.

핵심 개념

데이터 중심 아키텍처는 중앙 저장소를 여러 구성요소가 공유하는 구조이다.
데이터 흐름 아키텍처는 입력 데이터가 여러 처리 단계를 거치며 흘러가는 구조이며, 파이프 필터가 대표적이다.
계층 아키텍처는 시스템을 계층으로 나누고 상위 계층이 하위 계층의 서비스를 사용하는 구조이다.
호출과 반환 아키텍처는 주 프로그램과 서브루틴처럼 제어가 호출과 반환으로 이어지는 구조이다.

선지별 해설

① X: 데이터 중심 아키텍처는 저장소가 중심이 되는 구조이다. 틀린 부분은 파이프 필터를 데이터 중심 구조로 본 점이다. 올바르게는 저장소(repository)나 블랙보드처럼 공통 데이터 저장소를 여러 모듈이 공유하는 형태가 데이터 중심 아키텍처이다.

② O: 파이프 필터는 각 필터가 입력 데이터를 변환하고, 그 결과가 파이프를 따라 다음 필터로 전달되는 구조이다. 데이터가 단계적으로 흐르므로 데이터 흐름 아키텍처에 해당한다.

③ X: 계층 아키텍처는 기능을 계층별로 분리하고 상위 계층이 하위 계층 서비스를 이용하는 구조이다. 틀린 부분은 파이프 필터를 계층 구조로 본 점이다. 올바르게는 파이프 필터의 핵심은 계층 구분이 아니라 데이터의 연속적 변환 흐름이다.

④ X: 호출과 반환 아키텍처는 한 모듈이 다른 모듈을 호출하고 처리가 끝나면 반환되는 제어 구조가 핵심이다. 틀린 부분은 파이프 필터를 호출-반환 구조로 본 점이다. 올바르게는 파이프 필터는 제어 흐름보다 데이터 흐름이 중심인 구조이다.

NoteBase

오답 포인트

아키텍처 이름만 보고 저장소, 계층, 호출 같은 일반 용어에 끌리면 틀리기 쉽다. 이 문항은 '파이프'와 '필터'가 데이터를 단계적으로 넘겨 처리한다는 점을 떠올리면 바로 데이터 흐름 구조로 정리된다.

다시 나오면 이렇게 판단

아키텍처 유형 문제는 무엇이 중심인지부터 본다. 중앙 저장소가 중심이면 데이터 중심, 데이터의 연속 변환이 중심이면 데이터 흐름, 계층 분리가 중심이면 계층, 모듈 호출 관계가 중심이면 호출과 반환 구조로 판단된다.

암기 포인트

파이프 필터는 데이터가 흐르고, 저장소 구조는 데이터를 모은다.

NoteBase