2019311801_이균서_QA_sheet_4

2019311801 이균서

1. Describe why rapid development and delivery are important from a business and technology perspective.

business 측면:

- Business가 빠르게 변하는 requirement에서 작동한다.
- 초기 단계에 안정적인 software requirements를 set up한다는 것은 불가능에 가깝다.
- 사전에 모든 것을 정의하고 반영하는 것이 어렵다.
- 그럴 바에는 빠르게 개발하고 delivery하고 cycle을 잘게 쪼개서 반복적으로 수행해 나가는 것이 효과적이다.
- software는 변하는 business needs를 반영하여 빠르게 evolve해야 된다.

technology 측면:

- 새로운 기술을 바로 도입해서 반영하기 좋다.

2. Explain the principles of agile method with reasons.

- customer involvement: customer는 반드시 development process에 면밀히 연관돼야 한다. 그들의 역할은 새로운 system requirements를 제공하고 우선순위를 정하는 것 그리고 system 개발의 iteration (한 cycle)을 평가하는 것이다.
- incremental delivery: software는 customer가 명시하는 requirement가 increment에 포함되도록 개발돼야 한다.
- people, not process: process가 아닌 사람에 관해야 한다. 개발팀의 기술이 인정되고 이용돼야 한다. 팀 구성원은 규정된 process 없이 자신만의 방식으로 개발하도록 남겨둬야 한다.
- embrace change: system requirements가 바뀔 것을 예상해야 한다. 그래서 그런 변화를 수용할 수 있도록 system을 디자인해야 한다.
- maintain simplicity: software가 개발되는 것 그리고 개발 process에서의 simplicity에 집중해야 한다. 가능하다면, system의 complexity를 없애려고 해야 한다.

이런 principle은 결국 과도한 rework없이, software process에서 overhead를 줄이고, 변화하는 requirements에 빠르게 반응하기 위해서이다.

3. Explain the strengths and problems of agile methods. (중간 시험 문제)

Strengths

- software process에서 overhead(문서 작업)를 줄일 수 있다.
- 과도한 rework 없이, software process에서 overhead를 줄일 수 있고, 변화하는 requirements에 빠르게 반응할 수 있다.

Problems

- process에 관여하는 customers의 관심을 유지하는 것이 어렵다.
- 팀 멤버가 agile method의 특징인 intense involvement에 적합하지 않을 수 있다.
- stakeholder가 여러 명이면 change의 우선순위를 정하는 것이 어려울 수 있다.
- simplicity를 유지한다는 것이 extra work를 요구할 수 있다.
- 계약은 iterative development에 대한 약점이 될 수 있다.

4. Summarize the determinants when deciding on a plan-driven or agile approach to system development.

- 구현을 하기 전에 detailed specification과 design을 갖는 것이 중요한가? 그렇다면, plan-driven approach를 사용해야 한다.
- 고객에게 소프트웨어를 제공하고 고객으로부터 빠른 피드백을 받는 incremental delivery 전략이 현실적인가? 그렇다면, agile method를 고려할 수 있다.
- 개발 중인 시스템의 규모는 얼마나 되는가? agile method은 비공식적으로 의사소통할 수 있는 소규모 co-located 팀과 함께 시스템을 개발할 수 있을 때 가장 효과적이다. 이는 대규모 개발 팀이 필요한 대규모 시스템에서는 불가능할 수 있으므로 plan-driven approach을 사용해야 할 수도 있다.
- 어떤 종류의 시스템이 개발되는가?
  - 구현 전에 많은 분석이 필요한 시스템(예: 복잡한 타이밍 요구 사항이 있는 실시간 시스템)에는 plan-driven approach가 필요할 수 있다.
- 기대 system lifetime은 어떠한가?
  - long-life system에는 시스템 개발자의 원래 의도를 지원 팀에 전달하기 위해 더 많은 설계 문서가 필요할 수 있다.
- system development를 지원하는 기술들 중 어떤 것이 사용 가능한가?
  - agile method는 evolving design을 추적하기 위해 좋은 tool에 의존한다.
- 어떻게 개발팀이 조직되어 있는가?
  - 개발 팀이 분산되어 있거나 개발의 일부가 아웃소싱되는 경우 개발 팀 전체에 걸쳐 의사소통하기 위해 디자인 문서를 개발해야 할 수도 있다.

- 시스템 개발에 영향을 미칠 수 있는 문화적 또는 조직적인 issue가 있는가?
  - 전통적인 엔지니어링 조직에는 계획 기반 개발 문화가 있으며 이는 엔지니어링의 표준이다.
- 개발팀의 디자이너와 프로그래머는 얼마나 뛰어난가?
  - 프로그래머가 단순히 세부 설계를 코드로 변환하는 plan-driven approach보다 agile method가 더 높은 기술 수준을 요구한다.
- 시스템이 외부 규제에 영향을 받는가?
  - 시스템이 외부 규제 기관(예: 항공기 작동에 중요한 FAA 승인 소프트웨어)의 승인을 받아야 하는 경우 시스템 안전 사례의 일부로 자세한 문서를 작성해야 할 수 있다. architecture 디자인이 잘 돼야 한다.

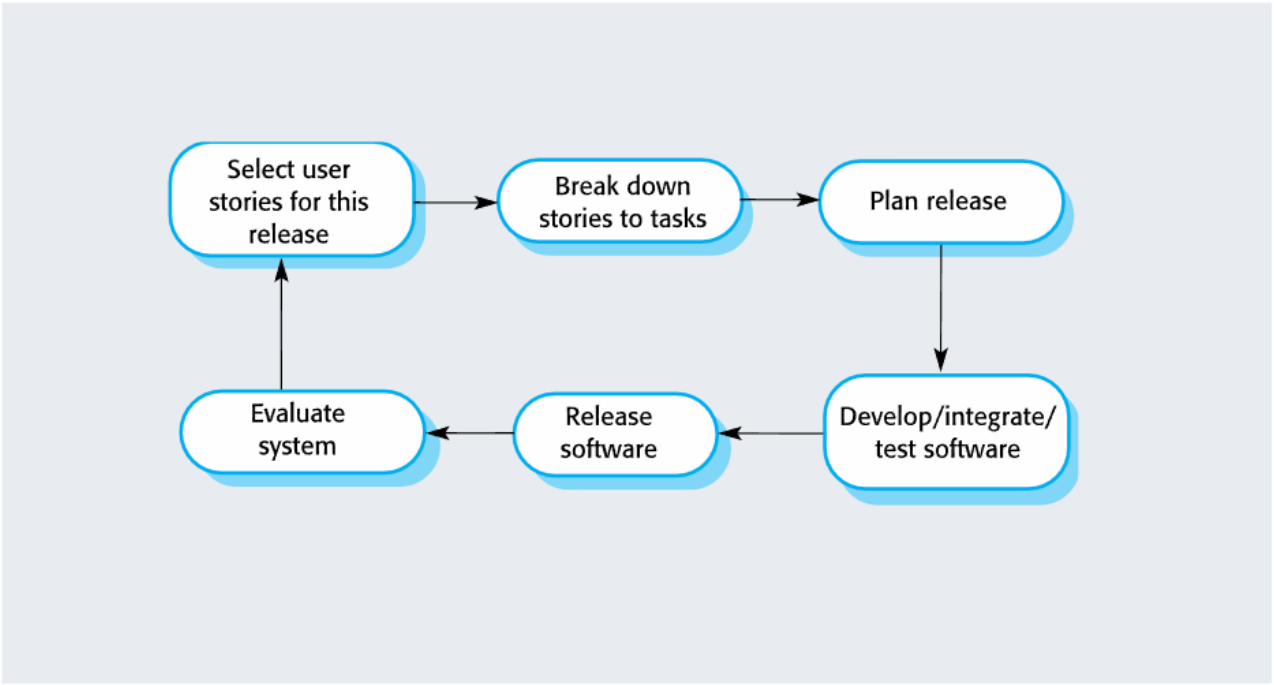
5. Explain the release cycle of XP and compare it to the waterfall model.

1. 하나의 유저 스토리를 선택
2. 스토리를 여러 개의 task로 분할한다.
3. release planning한다.
4. 개발/통합/테스트를 한다.
5. software를 릴리즈한다.
6. 시스템을 평가한다
7. 다시 1번으로

waterfall은?
requirements specification을 최대한 명세화를 하고, 진행하고, 이 과정들이 한번 반복되는 반면, agile 주기적으로 이를 받아 들인다.

rework은 waterfall 방법론에서 치명적이다.
agile은 rework를 한다면 그렇게 치명적이지 않다.

The extreme programming release cycle



6. Explain the important features on practices of XP.

- Incremental planning: 요구 사항은 스토리 카드에 기록되며 릴리스에 포함될 스토리는 사용 가능한 시간과 상대적 우선 순위에 따라 결정된다. 개발자들은 이러한 이야기를 개발 'task'로 분해한다.
- Small releases: 비즈니스 가치를 제공하는 minimal useful set of functionality가 먼저 개발된다. 시스템 릴리스는 빈번하며 첫 번째 릴리스에서 기능을 점진적으로 추가한다.
- Simple design: 현재 요구 사항을 충족하기 위해서 필요한 만큼의 충분한 design이 수행된다.
- Test-first development (Test Driven Development): 자동화된 unit test framework는 기능 자체가 구현되기 전에 새로운 기능에 대한 테스트를 작성하는 데 사용된다.
- Refactoring: 모든 개발자는 가능한 코드 개선 사항이 발견되는 즉시 코드를 지속적으로 refactoring해야 한다. 이는 코드를 단순하고 유지 관리하기 쉽게 유지한다.
- Pair programming: 개발자들은 2인 1조로 작업하며, 서로의 작업을 확인하고 항상 잘 할 수 있도록 지원한다.
- Collective ownership: 한 쌍의 개발자가 시스템의 모든 영역에서 작업하므로 전문 지식의 고립이 발생하지 않으며, 모든 개발자가 모든 코드에 대한 책임을 진다. 누구나 무엇이든 바꿀 수 있다.
- Continuous integration: task가 완료되면, 이것이 전체 시스템이 바로 통합이 된다. 이런 통합이 완료가 되면, 시스템에 있는 모든 유닛 테스트가 통과돼야 한다.

- Sustainable pace: 많은 양의 초과근무는 순효과로 인해 코드 품질과 중기 생산성이 저하되는 경우가 많기 때문에 허용되지 않는다.
- On-site customer: 시스템 최종 사용자(고객)의 대표는 XP 팀의 사용을 위해 풀타임으로 대기해야 한다. 극단적인 프로그래밍 프로세스에서 고객은 개발 팀의 구성원이며 구현을 위해 시스템 요구 사항을 팀에 가져오는 책임을 진다.

Principle or practice	Description
Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple design	Enough design is carried out to meet the current requirements and no more.
Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site customer	A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

7. Read the story card on slide 22 carefully and look for another task.

새로운 약을 처방한다. (prescribe new medication):
새로운 약 처방을 선택하면, 시스템은 당신이 원하는 처방약을 안다고 가정한다. 약 이름의 첫 몇 글자를 작성한다. 당신은 그 글자로 시작하는 가능한 몇 가지의 약 리스트를 보게 된다. 요구되는 약을 고른다. 그리고 당신은 당신이 고른 약이 알맞은지 체크한다. 복용량을 입력하고 처방을 확인한다.

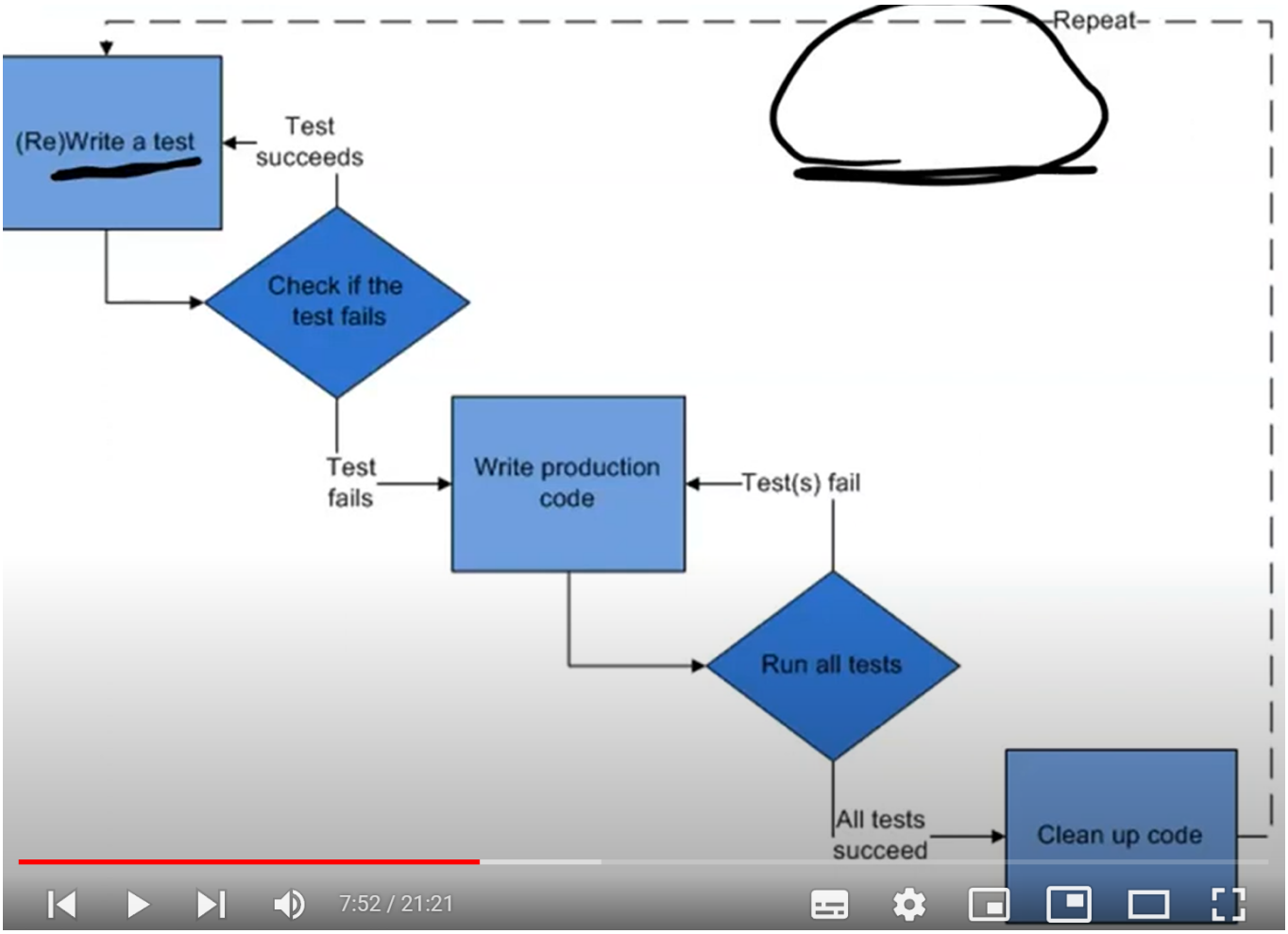
> If you choose, 'new medication', the system assumes that you know which medication you wish to prescribe. Type the first few letters of the drug name. You will then see a list of possible drugs starting with these letters. Choose the required medication. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

8. Explain test-driven development and think about its impact on SW quality.

test case와 그에 상응하는 expected output을 먼저 작성하고, 그 test case를 모두 pass할 수 있도록 program을 작성하고 계속 refine하는 것을 말한다.

- test code를 먼저 작성한다. 그 테스트 코드는 구현돼야 할 요구사항을 명확히한다.
- 테스트 코드는 프로그램이다. 그래서 자동으로 실행이 될 수 있다.
- 테스트는 제대로 돌아갔는지에 대한 체크를 포함한다.
  - 대개 테스팅 프레임워크에 의존한다.
- 새로운 기능이 추가되면 이전 테스트와 새로운 테스트가 모두 자동으로 실행되어 새 기능으로 인해 오류가 발생하지 않았는지 확인한다. (regression test)

-> SW quality가 높아질 것이다.



9. Explain the advantages of pair programming.

XP에서는 프로그래머가 pair로 모여 함께 앉아 코드를 개발한다.

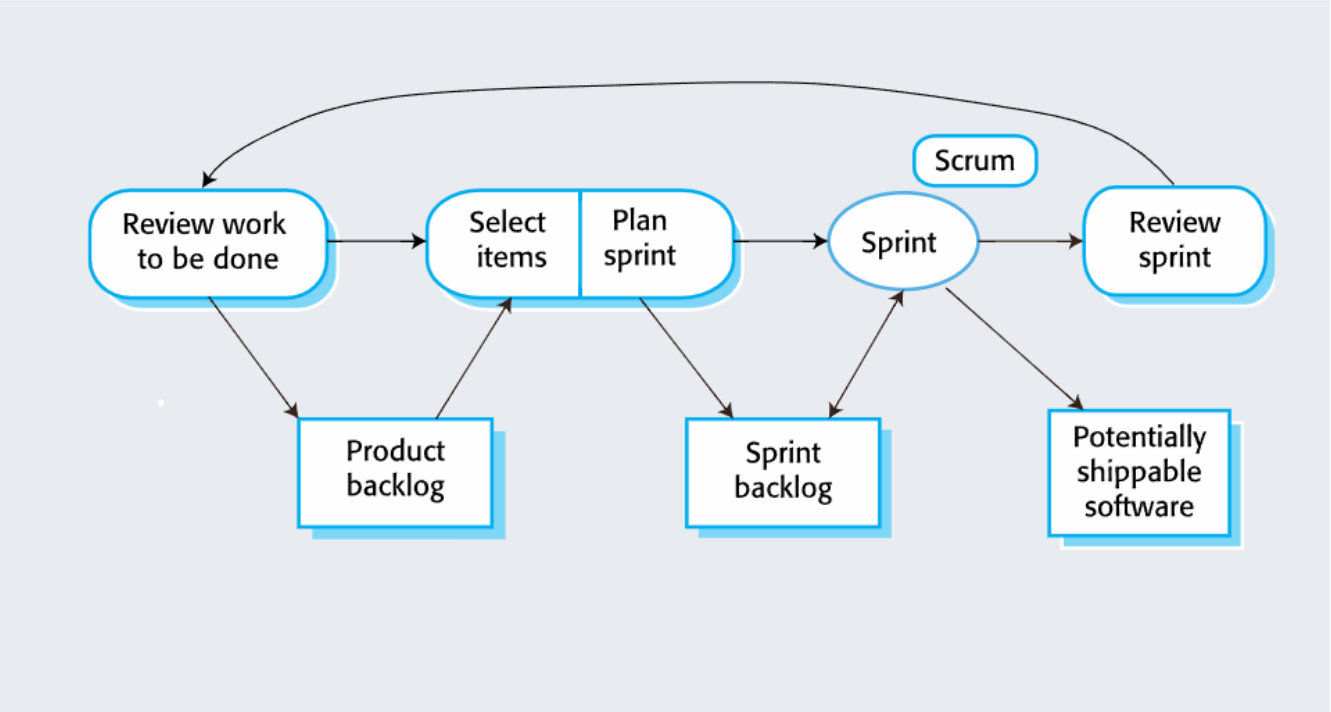
- 이는 코드에 대한 공동 소유권을 개발하고 팀 전체에 지식을 전파하는 데 도움이 된다.
- 각 코드 줄을 두 명 이상이 검토하므로 비공식 검토 프로세스 역할을 한다.
- 전체 팀이 리팩토링로부터 이익을 얻을 수 있으므로 이를 권장한다.
- 측정 결과에 따르면 페어 프로그래밍의 개발 생산성은 두 사람이 독립적으로 작업하는 것과 유사하다.

10. Explain the SCRUM process on slide 40.

1. 이번 프로젝트 동안 해야 될 일에 대한 리뷰 작업을 한다.
2. to-do list (product backlog) 찾아 내고 작성
3. 이번 사이클에서는 무엇을 할 것인가 선택하는 작업 (select items)
4. 그 선택에 기반한 increment를 planning 한다. (plan sprint)
5. 해당 sprint에 대한 backlog가 만들어 진다.
6. sprint가 돌아간다. (scrum이 이뤄진다.) (daily meeting을 통해 잘 이뤄지는지 평가한다.)
7. shippable한 software가 발생한다.

8. sprint를 리뷰하고 다시 1로 간다. (모든 backlog가 해결될 때까지 진행한다.)

product backlog에서 sprint backlog를 뽑아낼 때는 고객이 원하는 것을 먼저 뽑아 내야 한다. 왜냐면 그래야 그것들이 increment1, increment2, increment3를 거치며 계속해서 테스트되기 때문이다. (IBM 통계도 이러한 이야기)



11. Summarize the benefits of the SCRUM.

- 제품은 관리 및 이해가 가능한 단위로 구분된다.
- 불안정한 요구사항이 진행을 방해하지 않는다.
- 팀 전체가 모든 것을 볼 수 있으므로 팀 커뮤니케이션이 향상된다.
- customer는 increment의 정시 납품을 확인하고 제품 작동 방식에 대한 피드백을 얻는다.
- customer과 개발자 간의 신뢰가 형성되고 모두가 프로젝트의 성공을 기대하는 긍정적인 문화가 조성된다.

12. Explain the characteristics of large systems development.

- 대규모 시스템은 일반적으로 별도의 팀이 각 시스템을 개발하는 별도의 separate, communicating system 모음이다. 이들 팀은 서로 다른 장소에서 작업하는 경우가 많으며 때로는 서로 다른 시간대에서 작업한다.
- 대규모 시스템은 'brown field system'이다. 즉, 여러 기존 시스템을 포함하고 상호 작용한다. 많은 시스템 요구 사항이 이러한 상호 작용과 관련되어 있으므로 실제로 유연성과 점진적인

개발에 적합하지 않다.

- 여러 시스템을 통합하여 하나의 시스템을 생성하는 경우 개발의 상당 부분은 원본 코드 개발보다는 시스템 구성과 관련된다.
- 대규모 시스템과 그 개발 프로세스는 개발 방식을 제한하는 외부 규칙과 규제, 규정에 의해 제약을 받는 경우가 대부분이다.
- 대규모 시스템은 조달 및 개발 시간이 길다. 사람들이 필연적으로 다른 직업과 프로젝트로 이동하기 때문에 해당 기간 동안 시스템에 대해 알고 있는 일관된 팀을 유지하는 것은 어렵다.
- 대규모 시스템에는 일반적으로 다양한 stakeholder가 있다. 개발 프로세스에 이러한 다양한 stakeholder를 모두 참여시키는 것은 사실상 불가능하다.

Questions from your ownself

1. 소규모 startup team이나 1인 개발과 같은 상황에서 굳이 TDD를 할 여유나 필요가 있는가?

제품이 성공할지도 모르는 상황에서, 높은 test coverage ratio, architecture의 무결함을 가지고 있더라도 고객이 쓰지 않으면 아무 소용이 없다.

2. 효율적으로 다양한 테스트 케이스를 생성할 수 있을까?

code를 얼마나 cover하는가를 평가하는 test coverage ratio가 있는데, 기존의 코드 베이스가 너무 방대하여 코드의 모든 부분을 cover하는 것은 어렵다.
그래서, 조건 coverage, 분기 coverage 같은 것으로 테스트 코드를 구성하면 좋을 것이다.