3주차(10~14장)

10장 존재하지 않는 오류 정의

- 예외처리는 소프트웨어에서 최악의 복잡성 원인중 하나

10.1 예외가 복잡성을 더하는 이유

- 코드가 예외를 처리하는 경우
  - 호출자가 잘못된 인수 또는 구성정보를 제공한 경우
  - 네트워크 오류(지연, 손실)
  - 코드에서 버그, 내부 결함 등에 대한 상황이 벌어지는 경우
- 예외를 처리하는 방법
  - 예외에도 불구하고 진행 중인 작업을 진행하고 완료하는 것
  - 진행중인 작업을 중단하고 예외를 위로 보고하는 것
- 이 외에도 예외는 더 다양한 예외를 처리하도록 만든다.
- 예외처리 코드가 실제로 작동하는지 확인하는지도 어렵고, 중첩으로 사용될 수도 있다.

10.2 너무 많은 예외

- 불필요한 예외는 오히려 독이다. 시스템의 복잡성만 증가시킨다.
- 예외를 던져서 호출자에게 문제를 떠넘기지 마라.
- 예외를 던지는 것은 쉽지 핸들링하기는 어렵다.
- 예외를 처리해야하는 위치의 수를 줄여라.

10.3 존재하지 않는 오류 정의

- 처리할 예외가 없도록 API를 정의하는 것이 제일 좋은 방법이다.

10.4 예: Windows에서 파일 삭제하기

- Windows에서는 열려있는 파일을 삭제하기 위해서 프로세스를 종료해야한다. 별로임
- 유닉스 운영체제에서는 기존 파일은 유지하고 삭제할 파일을 디렉터리에서 제거해버린다.

    - 현재 파일을 사용중이더라도 삭제작업에서 오류 반환 안함
    - 사용중인 파일을 삭제해도 해당 파일을 사용하는 프로세스에 대한 예외가 생성되지 않음

10.5 예: Java substring 메서드

- String 클래스의 substring 메서드를 사용할 때 주어진 인덱스가 문자열 범위를 벗어나면 IndexOutOfBoundsException 예외를 던짐
    - 이건 불필요함
- 만약 존재하지 않는 인덱스를 넣었을 때 빈문자열이 나온다면 자연스러워진다. 이렇게 하면 깊은 메소드가 된다.

전반적으로 버그를 가장 좋은 방법은 소프트웨어를 더 간단하게 만드는 것이다.

10.6 예외들 표시하기

- 예외를 마스킹하면 시스템의 낮은 수준에서 예외적인 조건이 감지되고 처리되므로 더 높은 수준의 소프트웨어가 조건을 인식할 필요가 없다.
- 분산시스템에서 더 일반적이다.
- NFS(네트워크 파일 시스템)는 중단되면 사용자에게 오류가 났다고 한다.
    - 하지만 오류를 숨기고 응용프로그램을 정지시키는 것이 제일 좋다.
- 예외 마스킹은 모든 상황에서 작동하지 않지만 작동하는 상황에서는 강력한 도구다.
- 예외 마스킹은 복잡성을 낮추는 예이다.

10.7 예외 집계

- 예외 집계의 기본은 단일 코드로 많은 예외를 처리하는 것이다.
- 많은 개별 예외에 대해 별도의 핸들러를 작성하는 대신 상위의 단일 핸들러를 사용해 한 곳에서 모두 처리해라.
- 캡슐화 및 정보 은닉의 관점에서 좋은 특성을 가지고 있음.
- dispatcher와 에러 핸들링에 관한 예시
- 마스킹과 집계는 두 접근 방식 모두 대부분의 예외를 포착할 수 있는 예외 처리기를 배치하여 그렇지 않으면 만들어야 하는 많은 처리기를 제거한다는 점에서 유사함
- 많은 작은 오류를 더 큰 오류로 승격하는 방법도 있음. 하지만 트레이드 오프를 잘 생각할 것

10.8 그냥 충돌?

- 응용프로그램을 그냥 충돌 시켜버리기
- 처리할 가치가 없는 특정 오류는 잘 발생하지 않기 때문에 발생하면 그냥 진단 정보 인쇄하고 응용프로그램 중단하기
    - 메모리 부족 예
- 모든 곳에서 그냥 종료시켜버리는 것은 옳지 않다.
    - 복제된 스토리지 시스템의 경우 I/O 오류로 중단하는 경우는 좀… 별로임

10.9 존재하지 않는 특별한 경우 설계하기

- 가장 좋은 방법은 예외 추가 코드 없이 특수 케이스를 자동으로 처리하는 방식으로 일반 케이스를 설계하는 것이다.

10.10 너무 지나침

- 예외를 정의하거나 모듈 내부에서 마스킹 하는 것은 모듈 외부에서 예외 정보가 필요하지 않은 경우에만 의미가 있다.
- 예외를 제외하고 무엇이 중요하고 중요하지 않은지 결정해야한다. 중요한 것은 노출하고, 중요하지 않은 것은 노출하면 안된다.

11장 두 번 설계하기

- 설계를 할 때 서로 근본적으로 다른 접근방식을 이용해라.
- 하나의 설계가 나쁘더라도 무조건 두개의 설계를 해라. 대조하는 것도 도움이 된다.
- 그리고 각 설계의 장단점 목록을 작성해라.
- 대조해볼 때 체크리스트
    - 하나의 대안이 다른 대안보다 간단한 인터페이스를 가지고 있는가?
    - 무엇이 더 범용적인가?
    - 더 효율적인 구현을 가능하게 하는가?
- 어떤 것도 맘에 안들면 다른 설계를 해봐라.
- 똑똑하다고 자만하지말고 두번째 세번째 일을 해라

12장 왜 주석을 쓰는가? 4개의 이유들

- 코멘트를 작성하는 프로세스가 올바르게 수행되면 실제로 시스템 설계를 개선할 수 있다.
- 하지만 주석은 고된 작업으로 간주되는 경우가 많다.
- 세가지 팩트
  - 좋은 주석은 소프트웨어의 전반적인 품질에 큰 차이를 만들 수 있다.
  - 좋은 주석을 쓰는 것은 어렵지 않다.
  - 주석을 작성하는 것은 재밌을 수도 있다.

12.1 좋은 코드는 스스로 문서화된다.

- 코드를 잘 작성하면 주석이 필요없다고 생각하는데 아니다.
  - 코드로 표현할 수 있는 양의 제한이 있기 때문
- 메서드가 무슨 일을 하는지 알고 싶다면 해당 메서드의 코드를 읽으라고 하지만 시간이 많이 걸리고 고통스럽다.
  - 메서드를 읽기 좋게 만들려고 작은 메서드들로 만들고, 이는 얕은 메서드를 만들게 한다.
- 주석은 추상화의 기본이다.
  - 복잡성을 낮추는 요소이다.
  - 코드보다 덜 정확하지만 훨씬 직관적으로 설명할 수 있다.

12.2 주석 쓸 시간이 없어요

- 문서의 우선순위를 낮추게 되면 문서는 하나도 없게 된다.
- 장기적으로 프로젝트를 운영하기 위해 추가 시간을 들여야 한다. (문서작성/주석작성)

12.3 주석이 오래되어 오해의 소지가 있다.

- 큰 문제가 되지 않는다. 문서에 대한 대규모 변경은 코드에 대해서 대규모 변경이 필요하다.
- 코드변경은 문서변경보다 더 많은 시간을 사용해야 한다. 코드 검토는 오래된 주석을 감지하고 수정하는 훌륭한 메커니즘을 제공한다.

12.4 내가 본 주석은 가치가 없습니다.

- 유용하지 않은 주석이 있는 경우가 있음

12.5 잘 작성된 주석의 이점

- 주석의 전반적인 아이디어는 설계자의 마음에는 있지만 코드로는 표현할 수 없는 정보를 작성하는 것이다.
- 주석은 설계자가 변경하는 경우에도 유용하다. 세부사항 리마인드 가능하다.
- 복잡성이 나타나는 세 가지 방식의 소프트웨어 시스템
  - 변경 증폭
  - 인지부하
  - 어떤 코드를 어떻게 처리해야하는지 모름
- 주석은 마지막 두가지를 해결하는데 도움을 줌
- 좋은 문서는 종속성을 명확히 하고, 모호성을 제거하기 위해 간격을 채울 수 있다.

13장 주석은 코드에서 명확하지 않은 사항을 설명해야한다.

- 주석은 코드에서 명확하지 않은 많은 정보를 포함하는 추상화이다.
  - 코드는 너무 상세하다.

13.1 규칙 선택

- 댓글 작성의 첫 번째 단계는 주석을 달고 주석에 사용할 형식과 같은 주석 작성 규칙을 결정하는 것이다.
  - typedoc 등
- 컨벤션의 목적
  - 일관성 보장으로 주석을 더 읽기 쉽게 만들어 줌
  - 실제로 주석 작성 확인시 도움이 된다.
- 대부분의 주석은 다음 범주중 하나에 속함
  - 인터페이스: 클래스, 데이터 구조, 함수, 메서드 등
  - 데이터 구조 멤버: 클래스 인스턴스 변수 및 정적변수

    - 구현 설명: 코드가 내부적으로 작동하는 방식에 대한 메서드/함수 내의 설명
    - 교차 모듈 주석: 모듈 경계를 넘는 종속성을 설명하는 주석
- 처음 두개가 중요하다. 웬만하면 다 달기 주석

13.2 코드를 반복하지 마라

- 많은 주석은 도움이 되지 않는다. 반복하기 때문에
- 주석 반복하지 마라
- 메서드 파라미터와 메서드 명에서 알수있는 내용가지고 굳이 주석 작성하지 마라

> 💡 위험 신호: 주석 반복 코드
>
> 주석의 정보가 주석 옆의 코드에서 이미 명백하다면 주석은 도움이 되지 않는다.

- 좋은 주석을 달기 위한 첫 단계
    - 엔티티의 이름에 있는 단어와 다른 단어를 사용하는 것이다.

13.3 낮은 수준의 주석은 정밀도를 추가한다.

- 일부 주석은 코드보다 낮고 자세한 수준의 정보를 제공한다. 이것은 정밀도와 정확도를 더해준다.
- 다른 주석은 더 높고 추상적인 수준의 정보를 제공한다. 이것은 직관을 제공한다.
- 주석으로 다음과 같은 누락된 세부 정보를 채울 수 있다.
    - 변수의 단위
    - 경계조건이 교집합인지 합집합인지
    - 널값이 허용되는 경우의 의미
    - 변수가 해제하거나 닫아야하는 리소스를 참조하는 경우 이를 처리하는 것은 누구에게 책임이 있는가?
    - 이 array에는 항상 하나 이상의 항목이 포함되어 있다. 와같이 변수(불변량)에 대해 항상 참인 특정 속성이 있는가?
- 주석을 모호하게 작성하지 마라
- 변수를 문서화 할때 동사가 아닌 명사를 생각하라. 변수가 어떻게 조작됨이 아닌 변수가 나타내는 것에 초점을 맞춰라

13.4 높은 수준의 주석은 직관을 높인다.

- 메서드 내부의 주석과 인터페이스 주석에 주로 사용된다.
- 상위 수준에서 내부 코드의 전반적인 기능을 설명한다.
    - 무엇을 하는 코드인가
    - 코드의 모든 것을 설명하기 위해 가장 간단한 것은 무엇인가
    - 코드에서 가장 중요한 것은 무엇인가.
    - 실행되는 이유는 무엇인가

13.5 인터페이스 문서화

- 코드는 추상화를 설명하는데에 적합하지 않다. 주석이 유일한 방법이다
- 추상화를 문서화하는 방법
    - 구현 주석에서 인터페이스 주석을 분리하기
- 메서드에 대한 인터페이스 주석의 정보
    - 호출자가 인식하는 메서드의 동작을 설명하는 한두문장의 설명
    - 인수와 반환값을 설명. 매우 정확해야하며 제약조건과 인수간의 종속성을 설명해야한다.
    - 메서드에 사이드 이펙트가 있는 경우 이를 문서화해야한다.
    - 메서드에서 발생할 수 있는 모든 예외를 설명해야한다.
    - 메서드를 호출하기 전 충족해야하는 전제조건이 있는 경우 설명해야한다.

위험신호: 구현 주석이 인터페이스 주석을 오염시킬 때

인터페이스 주석이 구현 정보를 가지고 있을 때 발생

13.6 구현 주석: 어떻게가 아니라 무엇을, 왜

- 구현 주석의 목표는 독자가 코드가 수행하는 작업을 이해하도록 돕는 것
- for loop의 경우 주석이 있으면 유용하다.
- 이름이 애매한 경우의 지역 변수에도 붙여주면 좋다. 그렇다고 다붙이지 말자.

13.7 교차 모듈 설계 결정

- 교차 모듈 결정은 실제 시스템에서 필연적으로 나타나며 좋은 문서가 필요하다.
- 개발자가 자연스럽게 발견할 수 있는 위치를 찾아서 모듈간 문서화를 해야한다.
- 중앙 문서를 참조할 수 있는 링크를 달아두자. 각 교차 모듈에

14장 이름 고르기

14.1 예: 나쁜 이름은 버그를 일으킴

- 잘못된 네이밍으로 여러 사람이 읽어도 문제를 찾아내지 못함.
- 이름에 대해서 많은 시간을 들여야 한다. 합리적으로 가까운 수준이 아니라 명확하고 직관적이어야한다.

14.2 이미지 생성

- 이름을 선택할 때 목표는 이름이 지정되는 사물의 특성에 대한 이미지를 독자의 마음에 만드는 것이다.
- 단어가 너무 많으면 다루기 어려워지므로 가장 중요한 측면을 캡처하는 몇개의 단어를 찾아야한다.
- 이름은 추상화의 한 형태이다.

14.3 이름은 정확해야한다.

- 좋은 이름에는 정확성과 일관성이라는 두가지 속성이 있다.
    - 일반적이거나 모호하지 않게 지어야한다.

위험신호: 모호한 이름

변수 또는 메서드 이름이 다양한 항목을 참조할 수 있을 만큼 광범위하면 개발자에게 많은 정보를 전달하지 못하고 기본 엔티티가 오용될 가능성이 더 크다.

위험신호: 이름을 선택하기 어려움

기본 객체의 명확한 이미지를 생성하는 변수 또는 메서드에 대한 간단한 이름을 찾기 어렵다면 기본 객체의 설계가 깔끔하지 않음을 암시한다.

14.4 일관된 이름 사용

- 일관성을 유지하기 위한 세가지 요구사항
  1. 주어진 목적을 위해 항상 일반 이름을 사용한다.
  2. 주어진 목적 이외의 다른 용도로 일반 이름을 사용하지 말라.
  3. 이름을 가진 모든 변수가 동일한 동작을 갖도록 목적이 충분히 좁은지 확인하라.

14.5 다른 의견: Go 스타일 가이드

- Go 언어에서는 매우 짧아야하며 네이밍이 단일 문자여야한다고 주장한다.
- 하지만 가독성은 작가가 아닌 독자에 의해 결정되어야 한다.