제12장 정보은닉과 추상화:우리의 적인가 친구인가

- 정보 은닉과 추상화 정의 : 사물이나 시스템을 연구할 때 물리적, 공간적 또는 시간적 세부 사항이나 속성을 제거해 더 중요한 세부 사항에 주의를 집중하는 과정

12.1 정보은닉과 추상화는 한몸이다

- 내가 숨기고 있는 정보는 코드의 동작이다 우리의 목표가 복잡성을 관리해 머릿속에 편안하게 담을 수 있는 범위보다 더 복잡한 시스템을 구축하는 것이라면 정보를 숨겨야 한다는 것은 당연하다

눈앞의 작업/코드에만 집중할 수 있었으면 좋겠다.

12.2 큰 진흙탕이 된 코드의 원인을 찾아서

- 작업하기 어려운 코드 기반을 '큰 진흙탕'이라고 부른다. 코드 기반은 너무 복잡하고 얽혀 있기에 사람들이 변경을 두려워하는 경우

12.3 조직적이고 문화적인 문제

- 소프트웨어 개발 전문가이기 때문에 무엇이 효과가 있고 무엇이 효과가 없는지 가장 잘 이해할 수 있는 위치에 있다 오랜 기간 동안 안정적이고 반복적이며 지속 가능한 코드를 제공할 수 있도록 작업을 최적화할 필요가 있다 개발자는 작동하는 코드를 만들 필요가' 있지만, 또한 시간이 지나도 반복적이고 안정적으로 코드를 작성할 수 있는 능력을 유지할 필요도 있다

소프트웨어 전문가로서 우리는 소프트웨어를 개발하는 데 필요한 사안이 무엇인지 이해하는 것이 의무다. 우리는 우리가 작업하는 코드의 품질은 책임질 필요가 있다. 제대로 일하는 것은 우리의 주의 의무다

우리의 목표는 더 나은 소프트웨어를 더 빨리 개발하기 위해 필요한 모든 일을 하는 것이어야 한다.

우리의 일은 문제 해결이며, 이를 위해서는 설계에 주의를 기울이고 우리가 만든 해법의 효율성을 고려해야 한다.

12.4 기술적인 문제와 설계의 문제

-기존 코드 변경은 좋은 일이고 합리적인 일이라는 사고방식이다. 많은 조직이 코드를 변경을 두려워하거나 현실과 동떨어져 코드에 대한 일종의 경외심을 갖고 있다. 하지만 코드를 변경할 수 없거나 변경하지 않으려 한다면 그 코드는 사실상 죽은 코드다

새로운 것을 배워 무엇이 설계를 위해 최적인지에 대한 관점을 바꿀 때 그 시점에서 새롭고 더 깊은 이해를 반영해 설계를 변경할 수 있어야 한다.

12.5 과도하게 공들인 공학의 우려

- 다양한 요인이 개발자가 품질에 대한 책임을 포기하도록 압박한다

공을 과하게 들여 해법을 만들려는 유혹에 빠지는 또 다른 방법

-미래에 대비하게 만드는 것(지금은 필요하지 않지만 아마도 미래에는 필요할 것)

이런 미래 지향적 설계는 향후 개선사항이나 요구사항 변경에 대처할 수 있는 보험을 제공하기 위해 시도한다.

코드 변경을 두려워하는 문제에 대한 진정한 해법은 추상화와 테스트다. 코드를 추상화하면 정의상 시스템의 한 부분의 복잡성을 다른 부분으로부터 숨긴다

12.6 추상화를 높이려면 테스트 코드부터 작성하라

- 소프트웨어가 '항상 릴리스 가능한 상태'가 되도록 작업하며, 효율적이고 효과적인 자동화된 테스트를 통해 '릴리스 가능성'을 결정한다

추상화라는 컨텍스트에서 테스트를 코드의 바람직한 동작에 대한 미니 명세로 접근한다면, 바람직한 동작을 외부에서 내부로 들어오면서 설명하는 것이다

작업을 완료한 후에 명세를 작성하는 것이 아니라 시작하기 전에 명세가 필요하다. 따라서 코드를 작성하기 전에 테스트를 작성한다 이런 접근 방식을 따르면 정의에 따라 설계를 추상화한다

모든 설계 행위는 코드의 구현 세부 사항에 도달하기 전에 이뤄진다. 추상화에 기반한 이런 접근 방식은 코드가 수행할 필요가 있는 작업과 수행 방법을 분리하는 데 도움이 된다.

12.7 좋은 추상화가 핵심이다

- 평문 텍스트’ 추상화는 매우 강력한 추상화다. 또 다른 매우 강력한 추상화는 컴퓨팅에서 ‘파일’이며, 모든 사물이 파일인 유닉스 컴퓨팅 모델에서 그 정점을 찍었다

추상화는 컴퓨터를 다루는 우리 능력의 기본이다. 또한 우리가 컴퓨터에 가치를 더하기 위해 만드는 시스템을 이해하고 다루는 우리 능력의 기본이기도 하다.
소프트웨어를 작성할 때 우리가 하는 일을 바라보는 한 가지 방법은 새로운 추상화를 만드는 것이다

12.8 구멍 난 추상화

-구멍 난 추상화는 ‘추상화할 것으로 기대되는 세부 사항이 누수되는 추상화’

구멍 난 추상화’라는 개념은 추상화를 반대하는 것이 아니라 추상화는 우리가 관리해야 하는 복잡한 것임을 설명한다

누수 : 하드웨어의 한계에 최대한 근접’해 데이터를 처리하는 저지연 시스템을 구축 하려는 경우, ‘가비지 컬렉션’과 ‘임의 접근 메모리’라는 추상적인 개념은 대기 시간을 변수로 만들어 시간 측면에서 누수를 유발하기 때문에

누수의 영향을 최소화하고 싶다면 추상화, 캐시, 프리페치 주기 등을 이해하고 이를 설계에 반영할 필요가 있다

일반적으로 누수에 대처하려면 최대한 일관된 추상화 수준을 유지하려고 노력해야 한다

추상화, 모든 추상화는 근본적으로 모델링에 관한 것이다. 우리의 목표는 문제에 대해 추론하고 작업을 수행하는 데 도움이 되는 모델을 만드는 것이다

12.9 세계 지도와 지하철 노선의 비유로 배우는 추상화 기법

-우리가 선택하는 추상화의 성격이 중요하다. 여기에는 보편적인 ‘진리’가 존재하지 않으며 모델이 존재한다. 이에 대한 좋은 예로 지도를 들 수 있다

지도로 예시를 둔 추상화는 이동 거리가 절대적으로 필요한 것보다 더 길다는 사실에서 누수가 일어나지만, 사용 편의성을 위해 최적화하고 있으므로 계획하고 항해하는 동안에는 별문제가없다

지하철 노선도에는 완전히 다른 추상화가 사용된다 물리적인 지리와는 상관없는 위상학적으로 정확한 네트워크 지도를 만들었다

추상화, 그리고 그 핵심인 모델링은 설계의 기본이다. 추상화가 해결하고자 하는 문제에 초점을 맞출수록 더 나은 설계가 된다

12.10 이벤트 스토밍으로 추상화를 달성하자

-이벤트 스토밍은 관심 있는 개념을 나타낼 수 있는 행위의 군집을 식별하는 데 도움이 되며, 다른 기술적인 부분보다 더 결합도가 낮은 경향이 있는 문제 영역에서 경계 컨텍스트와 자연스러운 추상화의 경계선을 강조할 수 있다

12.11 추상화된 우발적인 복잡성

-10장에 나온 세 가지 응집성의 예)
추상화와 우발적인 복잡성과 본질적 인 복잡성의 분리라는 관점에서 보면 더 많은 통찰력을 얻을 수 있다.

▼ 코드 12.1 세 가지 응집성의 예(다시 보기)

```
def add_to_cart1(self, item):
    self.cart.add(item)

    conn = sqlite3.connect('my_db.sqlite')
    cur = conn.cursor()
    cur.execute('INSERT INTO cart (name, price) values (item.name, item.price)')
    conn.commit()
    conn.close()

    return self.calculate_cart_total();

def add_to_cart2(self, item):
    self.cart.add(item)
    self.store.store_item(item)

    return self.calculate_cart_total();

def add_to_cart3(self, item, listener):
    self.cart.add(item)
    listener.on_item_added(self, item)
```

첫 번째 예제인 add_to_cartl은 추상화가 전혀 이뤄지지 않고 모듈화되어 있지 않고, 응집성이 부족하며, 우발적인 복잡성과 본질적인 복잡성이 혼재되어 있고, 관심사를 분리하지 않아 나쁜코드이다

add_to_cart2는 정보 저장을 위한 추상화를 추가 했다.

add_to_cart3에는 본질적인 복잡성 코드를 그대로 유지한 추상화가 있다.
버전 2와 버전 3은 관심사를 분리하고 추상화를 선택했기 때문에 더 유연하고, 결합도가 더 낮으며, 더 모듈화되 어 있고, 응집성이 더 높다

12.2를 참고해 누수 범위를 제한하는 작업

▼ 코드 12.2 추상화 누수를 줄이기

```
def add_to_cart2(self, item):
    if (self.store.store_item(item))
        self.cart.add(item)

    return self.calculate_cart_total();
```

코드 12.2에서는 완전히 추상화된 버전 3에서 한 걸음 물러나 추상화에 '저장'이라는 개념을 허용했다. 항목을 저장하고 장바구니에 추가하는 관계의 트랜잭션특성을 성공 또는 실패 반환값으로 표현했다

12.12 타사 시스템과 타사 코드를 격리하자

-버전 1은 코드를 특정 타사 코드 (이 경우 sqlite3)에 노출하고 결합한다. sqlite3는 파이썬 세계에서 일반적인 라이브러리이지만, 그럼에도 불구하고 이 코드는 이 특정 타사 라이브러리에 구체적으로 묶인다. 버전 1 코드가 세 가지 코드 중 최악인 또 다른 이유는 바로 이 타사 코드와의 결합 때문이다.

타사 코드를 코드에 허용하는 순간 우리는 그 코드와 결합된다. 항상 자체 추상화를 통해 타사 코드와 내부 코
드를 분리하는 것이다

12.13 추상화와 구상화 사이의 트레이드오프

-YAGNI(**You Ain't Gonna Need it) : 지금 필요없는 기능을 만들지 말라!**

더 구체적인 표현보다는 더 일반적인 표현을 선호하라는 것

▼ 코드 12.3 정보 은닉을 선호함

```
public ArrayList<String> doSomething1(HashMap<String, String> map);

public List<Sting> doSomething2(Map<String, String> map);

public Object doSomething3(Object map);
```

첫 번째 버전은 지나치게 구체적이다 ⇒거의 확실하게 관심이 있는 사안은 ArrayLtst가 아닌 List의 특성!

doSomethtng3의 다소 불쾌한 함수 서명을 만든다면 멍청한 짓이다 Object가 올바른 수준의 추상화일 때가 있을수 있지만, 그런 경우는 드물고 항상 본질적인 복잡성 이 아니라 우발적 인 복잡성의 영역에 속한다.

일반적으로 doSomething2가 가장 일반적인 목표다

추상화를 위한 최적의 지점을 파악하는 능력은 테스트 가능성을 고려한 설계를 통해 향상된다.

정리:

추상화는 소프트웨어 개발의 핵심이다. 소프트웨어 엔지니어를 꿈꾸는 사람이 계발해야 할 필수적인 기술이다. 코드의 성격이 무엇이든 정보를 은닉하는 봉합 지점을 만들면 코드가 더 나아진다.