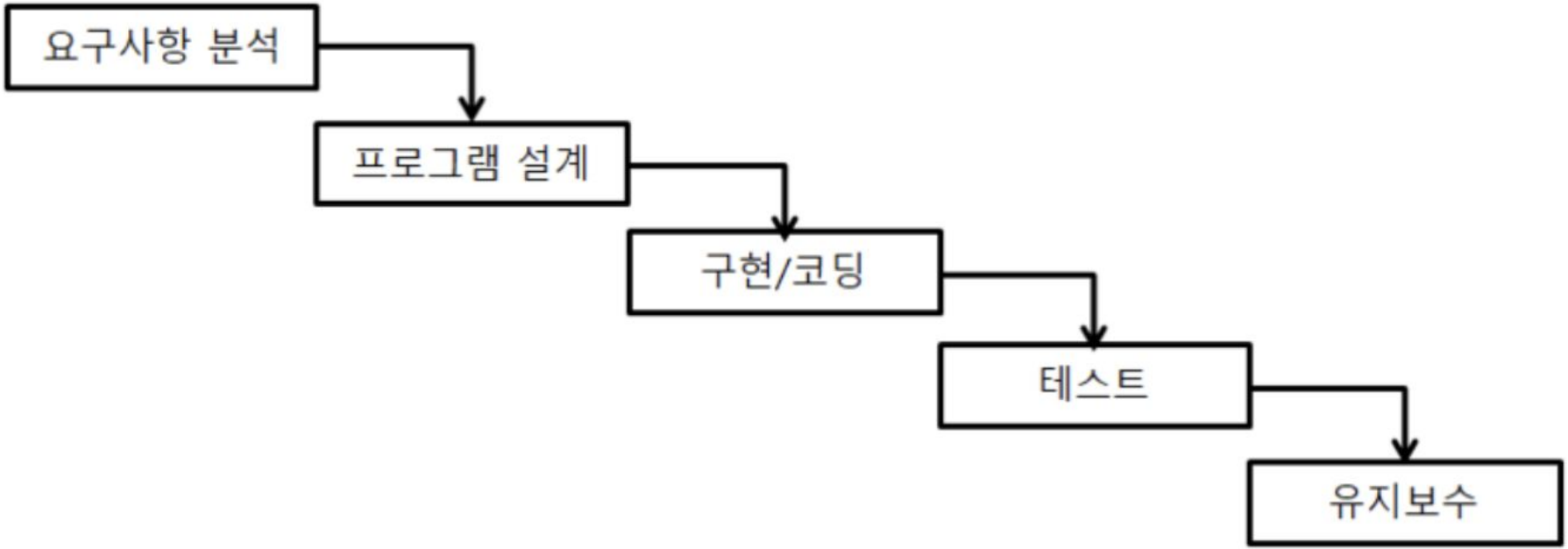
도메인 주도 설계 (1장 ~ 2장)

도메인

- 비지니스 영역
- 소프트웨어로 해결하고자 하는 문제
- 항공기 운항 관리 소트프웨어라면 → 항공기 운항에 관한 전문 지식

일반적인 개발 프로세스



담당 영역

도메인전문가, 기획자 → 요구사항 분석

개발자 → 프로그램 설계, 구현/코딩, 테스트, 유지보수

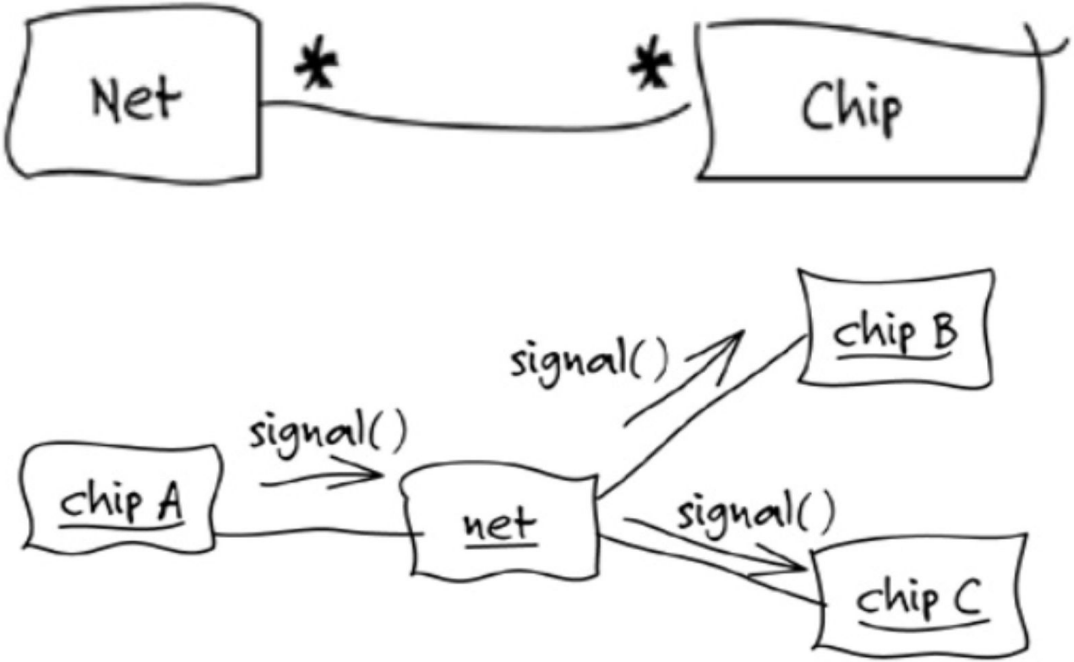
서로간의 지식 불일치

도메인 전문가, 기획자 → 도메인에 대해서는 잘 알고 있지만, 프로그램 설계에 대해서는 알지 못함

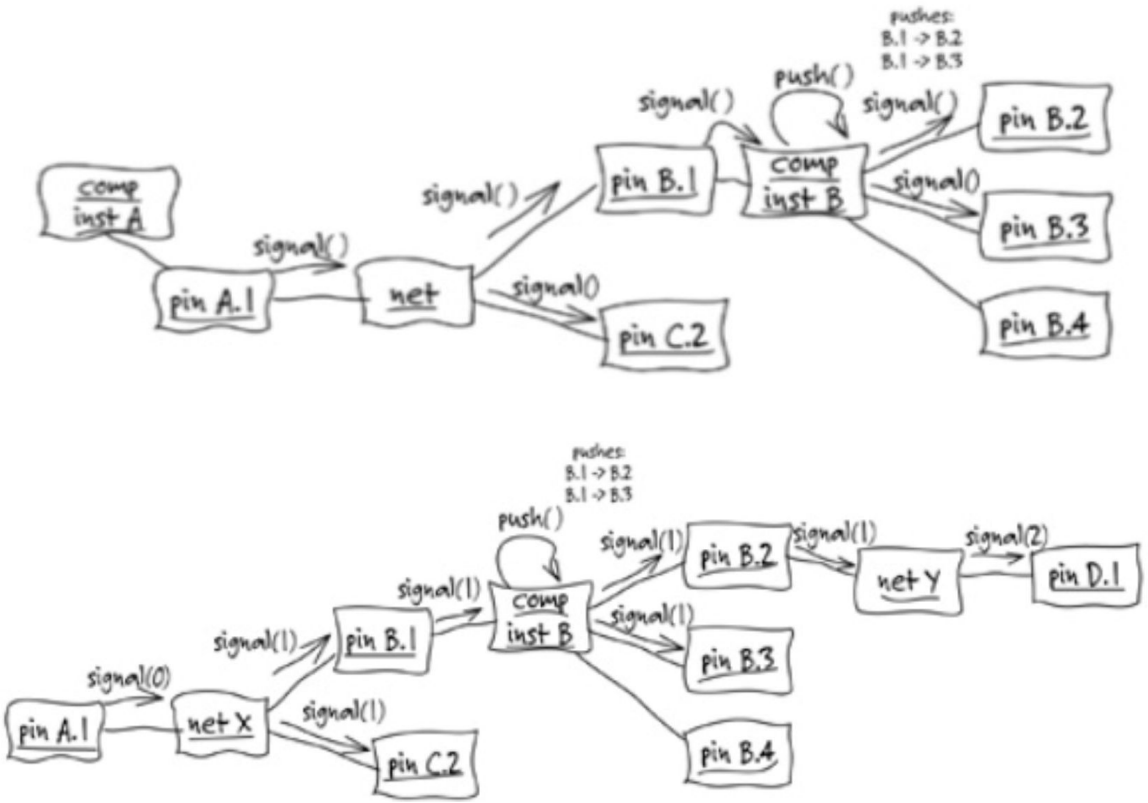
개발자 → 도메인에 대해서는 잘 알지 못할 경우가 많지만, 프로그램 설계에 대해서는 잘 알고 있음

→ 서로가 잘 이해할 수 있는 추상화된 객체 모델을 만들고 이를 정제해나가자 !!

지식 탐구 (pcb 회로기판 예제)



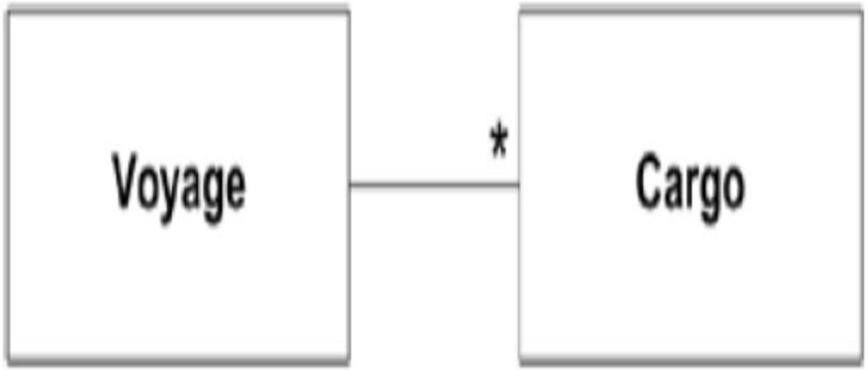
지식 탐구 (pcb 회로기판 예제)



지식 탐구

- 모델을 발전시키면서 자연스럽게 코드도 발전시킬 수 있었음
- 객체 모델을 통해 얘기하기 때문에 서로의 언어 (도메인 전문 용어, 기술 용어)를 이해할 필요가 없어짐
- 모델에 상태, 행동, 업무 규칙등 다양한 지식이 들어가 있기 때문에 모델만 보고도 핵심적인 부분 이해 가능
- 모델을 서로 이해하면서 불완전한 부분은 계속 사라지고 본질적인 부분만 남게됨

감춰진 개념 추출하기



- 화물 운송 애플리케이션
- voyage (운항)
- cargo (화물)

만약 선박이 나를 수 있는 화물 최대치보다 10%더 예약을 받아야 한다면 ?

첫 번째 구현 및 문제점

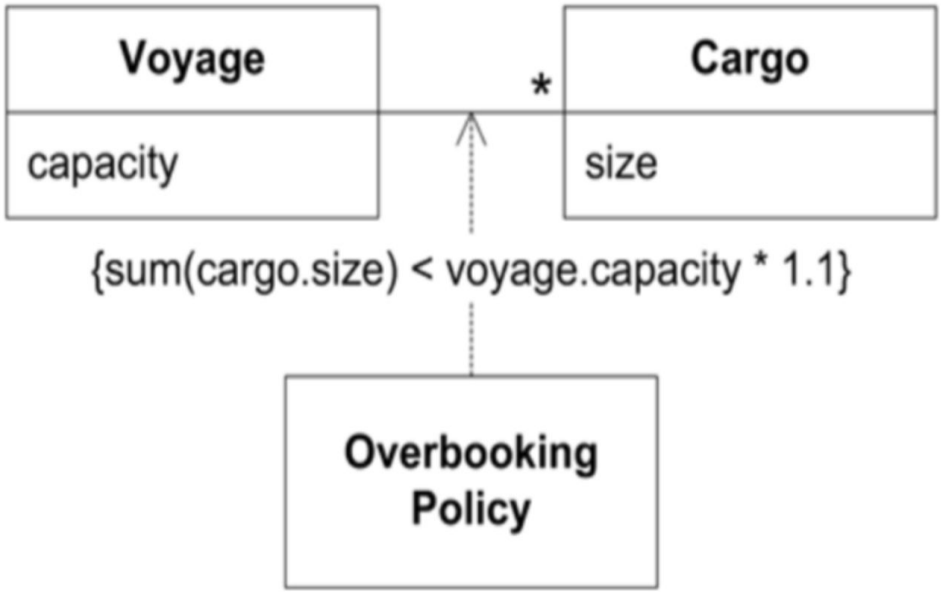
```
public int makeBooking(Cargo cargo, Voyage voyage) {
    double maxBooking = voyage.capacity() * 1.1;
    if ((voyage.bookedCargoSize() + cargo.size()) > maxBooking)
      return -1;

    int confirmation = orderConfirmationSequence.next();
    voyage.addCargo(cargo, confirmation);
    return confirmation;
}
```

- 중요한 업무 규칙이 메소드 안으로 감춰짐

  → 발견하기 어려움

- 도메인 전문가는 이 코드를 이해하지 못할 가능성이 높음

모델 변경



- 도메인 모델에 OverbookingPolicy 추가

변경된 구현

```
public int makeBooking(Cargo cargo, Voyage voyage) {
    if (!overbookingPolicy.isAllowed(cargo, voyage)) {
      return -1;
    }

    int confirmation = orderConfirmationSequence.next();
    voyage.addCargo(cargo, confirmation);
    return confirmation;
}
```

- 도메인에 세부사항이 추가되면서 모두가 규칙에 대해서 쉽게 이해할 수 있게 됨

도메인 전문가, 개발자 언어 불일치

도메인 전문가는 도메인 전문 용어는 잘 알지만, 프로그램 기술적인 용어는 알지 못함

개발자는 프로그램 기술 용어는 잘 알지만, 도메인 전문 용어는 모름

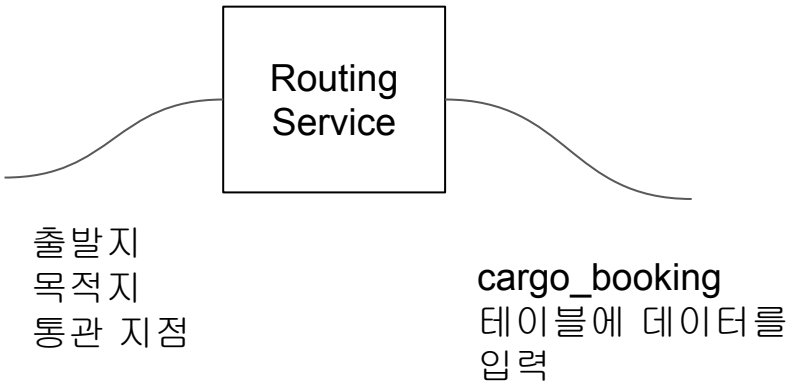
→ 서로 간의 언어를 번역, 이해하기 위해서 시간이 소모되고, 지식탐구를 더디게 만듬

→ UBIQUITOUS LANGUAGE 사용

UBIQUITOUS LANGUAGE

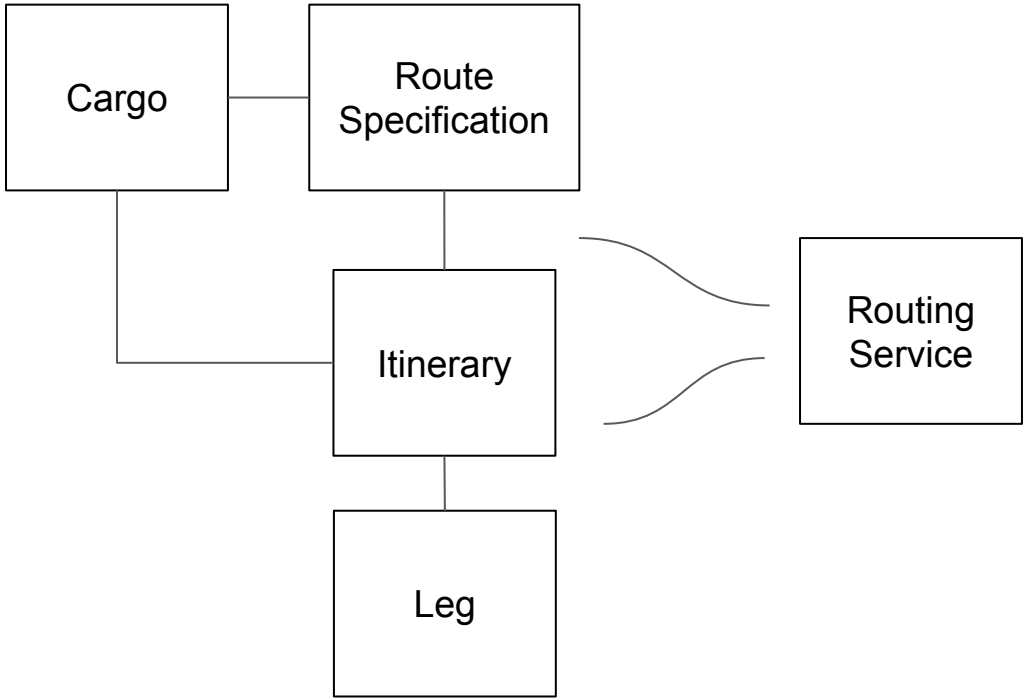
- 추상화된 모델 기반 언어 사용
- 팀 내 모든 의사소통, 코드, 다이어그램, 문서에서 동일한 언어 사용
- UBIQUITOUS LANGUAGE의 변화 → 모델의 변화

화물의 운송 항로 고안하기



- Routing Service에 출발지, 목적지, 도착 시간을 전달하면 화물이 멈춰야 할 지점을 찾고 그것을 데이터베이스에 삽입한다.

화물의 운송 항로 고안하기



- Routing Service는 Routing Specification을 만족하는 Itinerary를 찾는다.

화물의 운송 고안하기 결론

- 두 번째 대화의 경우 첫 번째 대화보다 도메인 모델 기반의 언어를 사용해서 대화가 더욱 정확해짐

- 운항 일정 (Itinerary) 가 객체로 표현되어 있어서 더 명확하고 구체적으로 논의 가능

도메인 전문가들에게는 도메인 객체 모델이 추상적이고, 이해하지 못하기 때문에 도메인 전문가들에게 도메일 모델을 보여줄 필요가 없다 ?

→ 도메인 전문가도 해당 모델을 이해하지 못한다면 모델이 뭔가 잘못된 것

다이어그램

- 설계의 핵심적인 부분을 표현하고 객체의 행위, 제약조건들을 잘 나타낼 수 있음

→ 하지만 다이어그램은 모델이 아니고, 보조적인 설명 수단이라는 것을 인식해야 함

문서

- 구두로 매번 전달하기 힘든 상황에서 안정감을 줌

→ 하지만 프로젝트가 진행되면서 바쁜 상황이 되면 코드보다 뒤쳐질 수 밖에 없음

→ 코드를 자세하게 설명하기 보다 의미를 설명하고, 대규모 구조, 핵심 요소에 집중할 필요가 있음

결론

- 서로가 이해할 수 있는 객체 모델을 가지고 지식탐구를 통해 계속 개선해나가야 한다.

- 객체 모델 기반의 UBIQUITOUS LANGUAGE를 의사소통, 코드, 문서 등 다양한 곳에 사용하여 의사소통 비용을 최소화 하여야 한다.