03 모델과 구현의 연계

종이에 기록된 모델과 동작하는 소프트웨어 간의 괴리를 어떻게 좁힐것인가?

Domain Driven Design, Modeling, Design

Model-Driven Design

모델과 코드의 괴리를 좁히는데 여러 설계 방법론에서는 분석 모델의 필요성을 지지한다.

- 분석 모델 (Analysis model) : 소프트웨어 시스템에서 수행할 역할에 대해서는 전혀 고려하지 않은 채 **업무 도메인의 개념만**을 체계화하고자 해당 업무 도메인을 분석한 결과물
  - 설계와 또렷이 구분되고 다른 사람들이 만든다.
  - 오로지 **이해하기 위한 수단**으로만 간주되며, 구현 관심사와 섞일 경우 혼란만 초래하는 것으로 여겨진다.
    - 분석 모델은 설계상의 쟁점들을 염두에 두고 만들어진 것이 아니라서 모델과 설계의 연결을 분석 모델로 진행한다는 것은 매우 **비현실적**일 가능성이 높다.
- 순수하게 이론에만 치우친 분석 모델
  - **도메인의 이해**라는 가장 주된 목표에 미치지 못하기도 하는데, 중요한 발견은 언제나 설계/구현을 위해 노력하는 가운데 나타나기 때문이다. 매우 특이하고 미처 예상치 못한 문제는 늘 일어나기 때문이다.
  - 결과적으로 코딩이 시작되자마자 폐기되고 대부분의 문제를 다시 검토해야 한다.



설계 혹은 설계의 주된 부분이 도메인 모델과 대응하지 않는다면 그 모델은 그다지 가치가 없으며 소프트웨어의 정확함도 의심스러워진다. 동시에 모델과 설계 기능 사이의 복잡한 대응은 이해하기 힘들고, 실제로 설계가 변경되면 유지보수가 불가능해진다. 분석과 설계가 치명적으로 동떨어지고, 그에 따라 각자의 활동에서 얻는 통찰력이 서로에게 전해지지 않는다.



Model-Driven Design에서는 양쪽 모두의 목적을 달성하는 단일 모델을 찾기 위해 분석 모델과 설계를 나누는 **이분법은 채택하지 않는다.**
순수하게 기술적인 쟁점은 배제함으로써 설계상의 각 객체는 모델에서 기술한 **개념적 역할**을 수행한다. …

(서로) 연계하는 과정에서 기술적 고려사항 탓에 분석이 심각하게 타협된 상태에 놓여서는 안 된다. 마찬가지로 도메인 아이디어는 반영하지만 소프트웨어 설계 원칙은 따르지 않은 서툰 설계를 받아 들여서도 안 된다. 이 접근법에는 분석과 설계 관점에서 모두 효과적인 모델이 필요하다.

모델이 **구현에 대해 비현실적**이거나 **도메인 핵심 개념을 충실하게 표현하지 않을 때** 새로운 모델을 찾아내야만 한다.

- How to?
  - 소프트웨어 시스템의 일부를 설계할 때는 **도메인 모델을 있는 그대로 반영해서** 설계와 모델의

대응을 분명하게 하라
- (모델) 지식탐구 → (코드) 리팩터링 → …(반복)
  - **Ubiquitous Language**를 지원하는 것과 더불어 분석과 설계 두 가지 측면을 충분히 만족하는 **단 하나의 모델**을 만들어내야 한다.

모델링 패러다임과 도구 지원

Model-Driven Design이 성과를 내려면 인간의 오차 범위 내에서 정확하게 모델이 직접적으로 대응해야 하는데, 이를 만들어낼 수 있는 소프트웨어 도구가 뒷받침되는 모델링 패러다임 내에서 업무를 하는 것이 거의 필수적이다.

그 도구는 **OOP**. 객체 설계에서의 진정한 도약은 **코드가 모델의 개념을 표현**할 때 나온다.

반면, Procedural 언어에서는 프로그래머가 도메인의 개념을 생각할 수 있을지언정 프로그램 자체는 데이터를 기술적으로 조작하는 것에 지나지 않기에 의미를 많이 담지 못한다.

내부 드러내기: 왜 모델이 사용자에게 중요한가

- 실제로 사용자가 바라보는 것과 시스템이 불일치한다면 혼란을 일으킬 수도 있고, 최악의 경우 버그를 유발할 수 있다.
  - 왜 사용자가 새로운 모델을 배우게끔 만드는가?



설계가 **사용자**와 **도메인 전문가**의 **기본적인 관심사**를 반영하는 모델에 기반을 두면 설계의 골격이 다른 설계 접근법에 비해 더 큰 범위에 걸쳐 사용자에게 드러날 수 있다. 모델이 드러나면 사용자가 소프트웨어의 잠재력을 좀더 많이 접하게 되어 일관성 있고 예상 가능한 행위가 나타날 것이다.

Hands-on Modeler

실천적 모델러



결국 Model-Dirven Design을 코드로 만드는 과정의 미묘한 사항들은 **협업**을 통해 알 수 있는데, 설계자가 구현을 하지 못해 개발자가 업무와 단절이 생기면 숙력된 설계자의 지식과 솜씨는 결코 다른 개발자에게 전해지지 못할 것이다.



모든 팀원에게는 각기 전문화된 역할이 있지만 분석과 모델링, 설계, 프록래밍에 대한 책임을 **지나치게 구분**하는 것은 Model-Driven Design과 상충한다. …

XP를 비롯한 모든 애자일 프로세스에서는 팀원의 역할을 정의하면 비공식적인 분업화가 자연스럽게 이뤄지곤 한다. 문제는 Model-Driven Design 내에서 서로 긴밀하게 연결된 모델링과 구현이라는 두 가지 과업을 **분리**하는 데서 나타난다.

Model-Driven Design에서 **코드는 모델의 한 표현**. 코드의 변경이 곧 모델의 변경. **프로그래머가 곧 모델러**!



모델에 기여하는 **모든 기술자(Any technical person contributing)**는 프로젝트 내에서 수행하는 일차적 역할과는 상관없이 **코드를 접하는 데 어느 정도 시간을 투자해야만 한다.** 코드를 변경하는 책임이 있는 모든 이들은 **코드를 통해 모델을 표현하는 법을 반드시 배워야 한다.**

모든 개발자는 **모델에 관한 일정 수준의 토의**에 깊이 관여해야 하고 **도메인 전문가와도 접촉**해야 한다.

다른 방식으로 모델에 기여하는 사람들은 의식적으로 코드를 접하는 사람들과 **Ubiquitous Language를 토대로 모델의 아이디어를 나누는 데 적극 참여해야 한다.**

From:
https://wiki.ledyx.xyz/ - **Ledyx WIKI**

Permanent link:
https://wiki.ledyx.xyz/doku.php?id=domain_driven_design:03_binding_model_and_implementation&rev=1691496914

Last update: **2026/10/24 07:03**