15 디스틸레이션

폭넓은 일련의 모델을 구분하고 도메인 모델의 정수를 추출하는 방법

Domain-Driven Design, Modeling, Design

Distillation

- 증류(법), 증류액, 추출물
- **정수**(뼛속의 있는 물. 사물의 본질을 이룬 가장 뛰어난 부분. 가장 중요한 것)
- 혼합된 요소를 분리해서 **본질**을 좀더 값지고 유용한 형태로 뽑는 것

어떻게 중요 문제에 집중하고 부수적인 사항에 매몰되지 않는가?
LAYERED ARCHITECTURE는 컴퓨터 시스템을 동작하게 하는 기술적 로직으로부터 도메인 개념을 분리하지만 규모가 큰 시스템에서는 격리된 도메인조차도 관리할 수 없을 정도로 복잡해질지도 모른다.

도메인 모델에 대한 전략적 디스틸레이션에서는 아래와 같은 사항을 모두 수행한다.

1. 팀원들이 시스템의 전체 설계와 해당 설계가 어떻게 함께 **조화**될지 파악하게끔 돕는다.
2. UBIQUITOUS LANGUAGE의 일부가 될 수 있게 **관리 가능한 크기의 핵심 모델을 식별**해서 **의사소통**을 촉진한다.
3. **리팩터링**을 이끈다.
4. **가장 중요한 모델 영역의 업무**에 **초점**을 맞춘다.
5. 아웃소싱, 기성 컴포넌트의 활용, 할당에 관한 **의사결정**을 돕는다.

flowchart LR coreDomain["CORE DOMAIN"] genericSubdomain["GENERIC SUBDOMAIN"] subgraph useOfSupplementalDocuments["보조적인 문서로써 활용하는 지표/안내"] domainVisionStatement["DOMAIN VISION STATEMENT"] highlightedCore["HIGHLIGHTED CORE"] end cohesiveMechanism["COHESIVE MECHANISM"] declarativeStyle["Declarative Style"] segregatedCore["SEGREGATED CORE"] abstractCore["ABSTRACT CORE"] intentionRevealingInterface["INTENTION-REVEALING INTERFACE"] coreDomain --"벗어날 대상으로 인식 (덜 중요한 도메인 분리)"--> genericSubdomain coreDomain --"방향을 제시하는 데 활용"--> domainVisionStatement coreDomain --"내용물을 표시하는 데 활용"--> highlightedCore coreDomain --"벗어날 대상으로 인식
(계산 메커니즘 분리)"--> cohesiveMechanism coreDomain --"재패키지화의 대상으로 간주"--> segregatedCore segregatedCore --"디스틸레이션의 대상으로 사용"--> declarativeStyle genericSubdomain --"디스틸레이션의 대상으로 사용"--> declarativeStyle segregatedCore --"디스틸레이션의 대상으로 사용"--> abstractCore cohesiveMechanism --"기능을 공개하는 수단으로 활용"--> intentionRevealingInterface click declarativeStyle "/domain-driven_design/part_3_refactoring_toward_deeper_insight/10_supple_design#a_declarative_style_of_design" "10 유연한 설계 - 선언적 형식의 설계" click intentionRevealingInterface "/domain-driven_design/part_3_refactoring_toward_deeper_insight/10_supple_design#intention-revealing_interfaces" "10 유연한 설계 - 의도를 나타내는 인터페이스"

CORE DOMAIN

모델을 **요약**하라. CORE DOMAIN을 찾아 그것을 지원하는 다수의 모델과 코드로부터 쉽게 구별할 수 있는 수단을 제공하라. 가장 가치 있고 전문화된 **개념**을 부각시켜라. CORE를 작

게 만들어라.

CORE DOMAIN에 가장 재능 있는 **인력**을 할당하고 그에 따라 **인력**을 채용하라. 시스템의 비전을 수행하기에 충분한 **심층 모델**을 찾고 유연한 설계를 개발할 수 있게 CORE에 노력을 쏟아라. 다른 부분에 대한 투자는 해당 부분이 어떻게 정수가 추출된 CORE를 **보조할 수 있느냐**로 정당화하라.

설계의 일부 측면을 **경쟁우위**로서 비밀스럽게 유지해야 한다면 그것이 바로 CORE DOMAIN에 해당한다.

CORE 선택

어떤 CORE DOMAIN을 선택하느냐는 **본인의 관점**에 달렸다.

설계의 다른 모든 부분과 마찬가지로 CORE DOMAIN을 파악하는 일은 **반복주기**를 거쳐 발견할 것이다. 특정 관계의 중요성이 처음에는 분명하게 드러나지 않을지도 모른다. 반면 처음에 명백히 중요한 것으로 보였던 객체가 보조적인 역할에 머무른 것으로 판명될지도 모른다.

누가 그 일을 할 것인가?

프로젝트 팀에서 기술적으로 가장 숙달된 구성원이 도메인에 관한 지식을 많이 갖춘 경우는 거의 없다.
그 결과 도메인 지식의 유용함에 한계가 생기고 도메인 지식을 보조 컴포넌트로 배정하는 경향이 심화되며, 지식 부족으로 말미암아 프로젝트 구성원들이 도메인 지식을 축적할 기회를 얻지 못하는 잘못된 악순환이 계속된다.

이러한 악순환을 깨뜨리는 것이 중요하다. 이를 위해 오랜 기간 팀에 참여하고 도메인 지식의 보고가 되는 데 관심이 있는 능력 있는 **개발자**와 업무를 깊이 있게 알고 있는 **여러 도메인 전문가**
로 팀을 구성한다.

CORE DOMAIN을 만드는 기반을 닦고자 단기간적으로 외부 설계 전문가를 고용하는 것은 대개 도움이 되지 않는다.
그 이유는 팀에서는 도메인 지식을 축적할 필요가 있고 일시적으로 고용한 구성원은 양동이에 난 구멍에 해당하기 때문이다.

비슷한 이유로 CORE DOMAIN을 구입[1]할 수 있다는 것은 있음직한 일이 아니다. 그러나 구입한 CORE DOMAIN의 프레임워크가 더 제약을 가한다면 아래의 세 가지 가능성이 있다. (→ 좋지 않은 징조/사례)

1. 본질적인 소프트웨어 자산을 잃는다. CORE DOMAIN을 제약하는 프레임워크의 적용 범위를 줄인다.
2. 프레임워크에서 다루는 영역이 생각했던 것만큼 중요하지 않다. 모델을 실제로 구분되게 만드는 부분에 따라 CORE DOMAIN의 경계를 변경한다.
3. CORE DOMAIN에 특별한 요구사항이 없다. 애플리케이션에 통합할 소프트웨어를 구입하는 것과 같은 좀더 위험도가 낮은 해결책을 고려한다.

디스틸레이션의 단계적 확대

본 장의 나머지를 구성하는 여러 디스틸레이션의 기법은 거의 어떤 순서로도 적용할 수 있지만 **설계를 얼마나 급격히 수정하느냐**와 관련해서 정도의 차이가 있다.

- DOMAIN VISION STATEMENT : 최소한의 투자로 기본 개념과 가치를 전달한다.
- HIGHLIGHTED CORE : 의사소통을 향상시키고 의사결정을 내리는 데 도움되지만 그럼에도 설계를 거의 수정하지 않거나 아예 수정하지 않아도 된다.
- GENERIC SUBDOMAIN : 좀 더 적극적인 리팩터링과 리패키징을 거쳐 명확하게 GENERIC SUBDOMAIN을 구분하면 각 GENERIC SUBDOMAIN을 개별적으로 다루는 것이 가능해진다.
- COHESIVE MECHANISM : 용도가 다양하고, 의미전달이 용이하며, 유연한 설계를 통해 캡슐화할 수 있다. 이러한 부수적인 요소를 제거하면 CORE가 엉키는 것을 방지할 수 있다.
- SEGREGATED CORE : SEGREGATED CORE를 리패킹하면 CORE를 심지어 코드상에서도 바로 볼 수 있게 되고, CORE 모델에 대한 향후 업무가 쉬워진다.
- ABSTRACT CORE : 가장 야심찬 것으로, 이것은 가장 근본적인 개념과 관계를 순수한 형태로 표현한다 (그리고 모델을 대상으로 광범위한 재구성과 리팩터링이 필요하다).

GENERIC SUBDOMAIN

현재 진행 중인 프로젝트를 위한 것이 아닌 응집력 있는 하위 도메인을 식별하라. 이러한 하위 도메인에서 일반화된 모델 요소를 추출해서 별도 MODULE에 배치하라. 해당 MODULE에는 여러분의 전문성의 흔적을 남기지 마라.

일단 하위 도메인이 분리되고 나면 해당 하위 도메인의 계속되는 개발에 대해서는 CORE DOMAIN보다 **낮은 순위**를 부여하고
그 일에 핵**심 개발자를 배치하지 않는다**(개발자가 거기서 도메인 지식을 거의 얻지 못할 것이므로). 아울러 이러한 GENERIC SUBDOMAIN에 대해서는 기성솔루션이나 공표된 모델을 고려해본다.

이러한 패키지를 개발할 때는 몇 가지 추가적인 선택사항이 있다.

선택 1: 기성 솔루션

때로는 구현된 제품을 구입하거나 오픈소스를 이용할 수 있다.

유리한 점

- 개발할 코드가 적어진다.
- 유지보수 부담이 외부화된다.
- 코드가 좀더 성숙하고 다양한 곳에서 사용되므로 사내에서 개발된 코드에 비해 실수가 적고 완전하다.

불리한 점

- 사용하기 전에 평가하고 이해하는 시간이 필요하다.
- 품질관리가 업계에서 이뤄지므로 올바르고 안정적일 거라 확신할 수 없다.
- 용도에 비해 과도하게 만들어져 있을지도 모른다. 최소주의적인 사내 구현에 비해 통합에 드는 노력이 더 클수도 있다.
- 플랫폼 의존성, 컴파일러 버전 의존성 등을 야기할 수도 있다.

선택 2: 공표된 설계나 모델

유리한 점

- 사내 모델보다 더 성숙되고 많은 사람들의 통찰력을 반영한다.
- 즉각적이고 높은 품질의 문서화

불리한 점

- 요구사항에 딱 맞지 않거나 과도한 설계일 수 있다.

선택 3: 외주제작된 구현

유리한 점

- 대부분의 지식이 필요하고 축적되는 CORE DOMAIN과 관련된 업무에서 핵심 팀을 해방시켜준다.
- 팀을 영구히 확대하거나 CORE DOMAIN의 지식이 흩어져 없어지지 않고 더욱 많은 개발이 이뤄질 수 있다.
- 명세가 외부로 전달돼야 하므로 인터페이스 중심의 설계가 이뤄지고 하위 도메인을 일반화된 상태로 유지하는 데 도움이 된다.

불리한 점

- 인터페이스, 코딩 표준, 그리고 다른 모든 중요 측면을 대상으로 의사소통이 필요하므로 여전히 핵심 팀에서 시간을 들여 구현을 살펴봐야 한다.
- 코드를 이해해야 하기 때문에 소유권을 내부로 이전하는 데 상당한 부담이 야기된다. (그럼에도 특화된 하위 도메인보다 부담이 덜한데, 일반화된 모델은 아마도 이해하는 데 특별한 예비지식이 필요하지 않을 것이기 때문이다.)
- 코드 품질이 고르지 않다. 두 팀의 상대적 역량에 따라 코드의 품질이 좋거나 나쁠 수 있다.

선택 4: 사내 구현

유리한 점

- 통합하기 쉬움
- 정확히 원하는 것만을 얻음
- 임시 계약자를 할당할 수 있음

불리한 점

- 계속되는 유지보수와 교육 부담
- 패키지 개발에 필요한 시간과 비용을 과소평가하기 쉬움

일반화가 재사용 가능하다는 의미는 아니다

코드의 재사용에 신경 써서는 안 된다. 이는 디스틸레이션의 기본적인 동기에 어긋난다. 즉, **CORE DOMAIN에 가능한 한 많은 노력**을 기울이고 보조적인 성격의 GENERIC SUBDOMAIN에는 필요한 만큼만 투자해야 한다는 것이다.

완벽한 일반성을 갖춘 모델을 개발할 필요는 없다. 당면 업무에 필요한 부분에 대해서만 모델링하고 구현해도 된다.

재사용을 목표로 설계할 일은 거의 없더라도 **일반 개념의 범위 내에서 설계를 유지하는 것과 관련해서는 엄격해야 한다.**

프로젝트 위험 관리

팀원들이 검증된 기술을 보유하고 도메인에 매우 친숙한 경우를 제외하면 최초로 만들어진 시스템은 GENERIC SUBDOMAIN이 아닌 **CORE DOMAIN의 특정 부분에 기반을 둬야 하고 그럼에도 단순해야 한다.**

DOMAIN VISION STATEMENT

"도메인 모델의 본질과 해당 도메인 모델이 얼마나 기업에 가치 있는가"에 초점을 맞춘 문서. 디스틸레이션 과정에서 방향성을 공유하게 만드는 이정표.

(약 한 페이지 분량으로) CORE DOMAIN을 짧게 기술하고 그것이 가져올 가치에 해당하는 **"가치 제안"**을 작성하라. 이 도메인 모델과 다른 것과 구별하는 데 도움되지 않는 측면은 무시하라. 도메인 모델이 어떻게 다양한 관심사를 **충족**하고 **균형**을 이루는지 보여라. **한정된 범위**에서 내용을 유지하라. 초기에 이 선언문을 작성하고 새로운 통찰력을 얻을 때마다 선언문을 개정하라.

예시

이것은 **DOMAIN VISION STATEMENT**의 일부다	이것이 중요하긴 하지만 **DOMAIN VISION STATEMENT**의 일부는 아니다
항공 예약 시스템	**항공 예약 시스템**
(내용)	(내용)

HIGHLIGHTED CORE

CORE DOMAIN을 쉽게 파악하기 위해 특별히 중요한 부분을 강조하는 기법

DOMAIN VISION STATEMENT에서는 대체로 CORE DOMAIN을 파악하지만 특정 CORE 모델의 구성요소를 식별하는 것은 전혀 예측할 수 없는 개별 해석의 몫으로 남는다. 예외적으로 팀의 의사소통 수준이 높지 않다면 VISION STATEMENT만으로는 거의 효과가 없을 것이다.

디스틸레이션 문서

디스틸레이션 문서는 완전한 설계 문서가 아니다

최소주의적인 진입점으로서 **CORE의 윤곽**을 드러내고 설명하며, **특정 부분**을 좀더 면밀하게 조사하는 이유를 제기한다.

(표현 기법에 상관 없이) CORE DOMAIN과 CORE의 구성요소 사이에서 일어나는 **상호작용**을 기술하는 (3~7 페이지 가량의) **매우 간결한** 문서를 작성하라.

비기술 관련 팀원도 이해할 수 있는 문서를 작성한다.

문서를 별도로 유지했을 때 발생하는 통상적인 위험 요소는 다음과 같다.

1. 문서가 관리되지 않을 수도 있다.
2. 문서를 아무도 읽지 않을지도 모른다.
3. 정보의 출처가 늘어남으로써 복잡성을 관통하는 문서 본연의 목적이 무의미해질 수도 있다.

표시된 CORE

방대한 문서 혹은 도메인 속에서 **어디서부터 시작해야 하나?**

(표현 기법에 상관없이) 모델의 주요 저장소 안에 있는 CORE DOMAIN의 구성요소에 대해 그것을 **설명하지 말고 표시**하라. 개발자가 힘들이지 않고도 CORE의 안과 밖을 알 수 있게 하라.

프로세스 도구로서의 디스틸레이션 문서

XP를 실행하는데 가이드

디스틸레이션 문서가 CORE DOMAIN의 본질적인 면면의 윤곽을 드러낸다면 모델 변경의 중요성을 나타내는 실질적인 **지표**로 작용한다. 모델이나 코드 변경이 디스틸레이션 문서에 영향을 주면 다른 팀원과의 **상의**가 필요하다. 변경이 일어나면 모든 팀원에게 이러한 사실을 즉시 알리고 새로운 버번의 문서를 배포해야 한다.

COHESIVE MECHANISM

계산 메커니즘을 간결하게 만드는 모델을 찾아라. 응집력 있는 부분을 찾아내면 나머지 메커니즘을 이해하기가 좀더 쉬워질 것이다.

개념적으로 COHESIVE MACHANISM을 **별도의 경량 프레임워크로 분할**하라. 특히 형식주의[2]나 잘 문서화된 알고리즘 카테고리를 주의 깊게 살펴라.

INTENTION-REVEALING INTERFACE로 프레임워크의 기능을 노출하라. 이제 도메인의 다른 요소들은 해법의 복잡성("어떻게")을 프레임워크에 위임해서 **문제("무엇")을 표현하는 데 집중할 수 있다.**

새로 만들어지는 것이 **계산에만 집중**하게 하고 표현력 있는 모델과 섞이지 않게 한다. 그러면 책임의 분리가 일어나고 CORE DOMAIN이나 GENERIC SUBDOMAIN의 모델이 사실이나 규칙, 또는 문제를 **정형화**하게 된다. 그리고 COHESIVE MECHANISM은 모델에 구체화돼 있는 대로 **규칙을 결정하거나 계산을 완료한다.**

예시 : 그래프 알고리즘, SPECIFICATION

GENERIC SUBDOMAIN Versus COHESIVE MECHANISM

공통점으로는 CORE DOMAIN의 부담을 더는 데 목적이 있다. 차이점은 각자 맡고 있는 책임의 본질에 있다.

- GENERIC SUBDOMAIN : 팀이 도메인을 어떻게 바라보는지와 관련된 일부 측면을 나타내는 표현력 있는 모델이 토대를 둔다.
  단지 CORE DOMAIN에 비교하여 덜 중심적이고, 덜 중요하고, 덜 특화되었다는 점을 제외하면 전혀 다르지 않다.
- COHESIVE MECHANISM : **도메인을 나타내지 않는다.** 표현력 있는 모델에서 제기하는 일부 성가신 **계산** 문제를 해결해줄 뿐이다.

모델이 제안하면, COHESIVE MECHANISM은 **처리**한다.

MECHANISM이 CORE DOMAIN의 일부인 경우

예외적으로 MECHANISM 자체가 기업 소유이고 소프트웨어 가치의 **핵심 부분**을 차지하는 경우, 간혹 **대단히 특화된 알고리즘을 보유**한 경우 비용 편익 분석 기반으로 분리를 할지 말지 결정될 것이다.

선언적 형식의 디스틸레이션

디스틸레이션의 가치는 현재 여러분이 무슨 일을 하고 있는지 볼 수 있다는 것이다. 즉, 관련성 없는 세부사항 탓에 혼란을 겪지 않고도 **본질**에 직접적으로 도달하는 것이다.

선언적 형식을 취하게 되면 더 정수에 가까운 상태가 된다.

SEGREGATED CORE

CORE DOMAIN을 구조적으로 구별하는 데 직접적인 접근법

> GENERIC SUBDOMAIN을 추출하는 식으로
> 도메인에서 일부 CORE를 불분명하게 만드는 세부사항을 제거해 CORE를 눈에 띄게 만들 수 있지만 모든 하위 도메인을 식별하고 분명하게 하는 작업은 쉽지 않고, 이 중 일부는 그럴 만한 가치가 없는 듯 하다.

보조적인 역할(잘못 정의된 것을 비롯해)로부터 CORE 개념을 분리되게끔 모델을 리팩터링하고 CORE와 다른 코드와의 결합은 줄이면서 CORE의 응집력은 강화하라. 모든 일반적이거나 보조적인 역할을 하는 구성요소를 다른 객체로 추출해서 다른 패키지에 배치하라. 심지어 이러한 과정이 매우 긴밀하게 결합돼 있는 요소를 분리하는 식으로 모델을 리팩터링하는 것을 의미하더라도 말이다.

> AOP와 비슷한 의미인듯.

SEGREGATED CORE로 리팩터링하는 단계는 대체로 아래와 같다.

1. CORE 하위 도메인을 식별한다(디스틸레이션 문서에서 도출해야 할 수도 있음).
2. 새로운 MODULE로 관련 클래스를 옮긴다. 이 때 MODULE의 이름은 관련 개념에 따라 짓는다.
3. 개념을 직접적으로 표현하지 않는 서버 데이터와 기능으로 코드를 리팩터링한다.
   제거된 측면을 (아마도 새로 만들어진) 다른 패키지에 있는 클래스에 배치한다. 그러한 측면은 개념상 관련 작업과 함께 둬야 하지만 완벽하게 하려고 너무 많은 시간을 들여서는 안 된다.
   CORE 하위 도메인의 불순물을 제거하고 그곳에서 다른 패키지를 참조하는 바를 명시적이 그 자체로도 이해할 수 있는 상태로 만드는 데 집중한다.
4. 새로 생긴 SEGREGATED CORE MODULE의 관계와 상호작용을 더욱 단순하고 전달력 있게 만들고, 다른 MODULE과의 관계가 최소화되고 분명해지게끔 리팩터링한다. (이 것은 계속 진행되는 리팩터링의 목표가 된다.)

5. SEGREGATED CORE가 완전해질 때까지 또 다른 CORE 하위 도메인을 대상으로 위 단계를 반복한다.

SEGREGATED CORE를 만드는 데 드는 비용

- 특정 MODULE의 CORE DOMAIN의 응집도를 이끌어내기 위해 Cohesion이 희생될 수도 있는데, 엔터프라이즈 소프트웨어의 가장 큰 부가가치는 모델의 기업별 측면에서 나오기 때문에, 이를 CORE DOMAIN으로 이끌어내면 오히려 이득이다.
- CORE를 분리하려면 해야 할 일이 많기 때문에 개발자가 많은 시간을 보낸다.

SEGREGATED CORE를 노출시켜야 할 때는 규모가 큰 BOUNDED CONTEXT가 시스템에 결정적인 역할을 하지만 많은 양의 보조적인 기능 탓에 **모델의 근본적인 부분이 가려지는 경우다.**

발전하는 팀의 의사결정

의사소통은 모든 사람이 CORE라는 하나의 시각을 함께 유지할 수 있을 만큼 효과적으로 이뤄져야 한다.

ABSTRACT CORE

모델의 가장 근본적인 개념을 식별해서 그것을 별도의 클래스나 추상 클래스, 또는 인터페이스로 추출하라. 이 추상 모델이 중요 컴포넌트 간에 발생하는 **상호작용**을 대부분 표현할 수 있게끔 설계하라. 특화되고 세부적인 구현 클래스는 하위 도메인을 기준으로 정의된 자체적인 MODULE에 남겨둔 상태에서 이 추상적이면서 전체적인 모델을 자체적인 MODULE에 배치하라.

ABSTRACT CORE를 모델링하려면 핵심 개념과 해당 개념이 시스템의 주요 **상호작용**에서 수행하는 역할을 심층적으로 이해해야 한다. 다시 말해, 이것은 더 심층적인 통찰력을 향한 리팩터링의 한 사례에 해당한다. 그리고 대개 여기에는 상당한 양의 재설계가 필요하다.

심층 모델의 디스틸레이션

flowchart subgraph subCoreDomain["CORE DOMAIN의 정제 → Project 전체"] deepModel["Deep Model의 정제 → 한 도메인"] end

리팩터링의대상 선택

CORE DOMAIN 분리에 집중.

1. 고통 주도적(pain-driven) 리팩터링에서는
   문제의 근원에 CORE DOMAIN이나, CORE와 지원 요소와의 관계가 관련돼 있는지 살핀다.

만약 그렇다면, 이를 악물고 그 부분을 가장 먼저 고쳐야 한다.

2. 마음껏 리팩터링할 수 있는 상황이라면
   제일 먼저 CORE DOMAIN을 더 잘 분해하고, CORE의 격리를 개선하며, 보조적인 하위 도메인이 GENERIC하게 만드는 데 집중한다.

---

[1] 소프트웨어, 프레임워크 등

[2] formalism, 목표 달성을 위한 실질적인 내용보다 의식이나 절차 선례 관습등에 집착하는 성향을 형식주의라고 한다. 실질적인 목적보다 표면적인 현상에 행정력을 낭비하여 행정 목적에 어긋나는 것이라고 할 수 있다. 출처 : 나무위키