1장. 소프트웨어모델링이란?

(2023)

사고기술 – 思考 技術- Thinking Art

시작하기 전에

이 장은 소프트웨어모델링이 무엇인지에 대해 아는 것을 목표로 합니다.

좀 더 구체적으로는 다음과 같은 질문들에 답할 수 있어야 합니다.

1. 왜 소프트웨어모델링을 해야 하는가?
2. 소프트웨어모델링이란 무엇이고 무엇을 하는 것인가?
3. 소프트웨어모델링을 하려면 무엇을 알아야 하는가?
4. 소프트웨어개발주기의 어디에서 소프트웨어모델링 하는가?

1절. 소프트웨어모델링이란?

1.1 소프트웨어 개발은 한마디로 '복잡한 문제해결'입니다.

소프트웨어 개발은 문제해결입니다. 고객의 문제를 소프트웨어로 해결하는 것입니다.
소프트웨어 개발은 문제도 복잡하고 해결책도 복잡합니다. 또한 문제영역과 해결책영역이 달라 그 복잡도가 더 커집니다.

문제는 트러블의 의미도 있지만 오늘날의 상황에서는 기회, 도전거리의 의미에 더 가깝습니다.

소프트웨어의 또 다른 큰 특징 중 하나는 변경입니다. 대상이 단순하면 변경도 쉽습니다. 변경이 어렵다는 것은 복잡하기 때문입니다. 따라서 변경 또한 복잡성에 대한 것입니다.

1.2 소프트웨어개발은 복잡성을 다룰 수 있어야 합니다.

1.3 모델을 사용하는 것은 복잡성을 다루는 최고의, 보편적인 방법입니다.

모델은 복잡성을 다루기 위해 등장한 것으로, 이것보다 더 나은 것은 아직까지 등장하지 않고 있습니다.

모델은 복잡한 대상을 다루는 대부분의 산업 분야에서 보편적으로 사용되고 있습니다.

인류가 만들어내는 가장 복잡한 것들 중에 하나인 소프트웨어를 개발하기 위해 모델을 사용하는 것은 너무나 당연한 것이고 필수적인 것입니다.

1.4 모델은 복잡한 대상을 다수의 관점을 가지고 추상화를 통해 얻어지는 개념을 언어적으로 표현한 것입니다.

사람은 한 번에 5개에서 9개 정도를 이해하고 다룰 수 있다고 합니다. 한꺼번에 다루어야 하는 것이 9개를 넘어서면 이해의 한계에 의해 복잡하게 느끼는 것입니다.

복잡한 것을 다루기 위해서는 먼저 복잡한 것을 다룰 때의 한계를 인정해야 합니다. 그리고 그 한계를 넘지 않을 정도만을 남기고 나머지는 잠시 숨기거나 버려야 합니다.

한꺼번에 드러난 세부사항들 중에서 어떤 것은 드러내고 어떤 것은 감추거나 버릴 것인가를 결정하기 위해서는 그렇게 하는 목적이 있어야 합니다. 목적에 따라 어떻게 대상을 바라보아야 하는지에 대한 **관점(view)**이 결정됩니다.

구체적인 세부사항들 중에서 목적(관점)에 맞지 않는 세부사항을 감추거나 버리는 것을 **추상화(abstraction)**라 합니다.

추상화를 통해 얻어진 **개념(concept)**은 커뮤니케이션을 위해 **언어적으로 표현**되어야 합니다.

1.5 모델은 효과적이고 효율적으로 대상에 대해 알게 하고, 학습하게 하며, 커뮤니케이션하게 하며, 다룰 수 있게 합니다.

모델은 복잡한 대상을 추상화를 통해 단순화시켜 줌으로 효율적이고, 목적에 맞는 부분만 추출해 냄으로 효과적입니다.
모델은 대상에 대한 현재의 이해의 결과를 제공함으로 모르는 것을 드러내어 학습하게 합니다.
모델은 이해의 결과를 기반으로 커뮤니케이션 하며, 대상을 다룰 수 있게 합니다.

1.6 모델링은 모델을 만들어 가면서 복잡한 대상을 다루는 사고법(思考法) 입니다.

뭔가를 다루려면 그 대상에 대해 이리저리 따져 깊이 생각해야 합니다. 즉 사고해야 합니다.

복잡한 것은 단순한 것을 다루는 사고법으로는 다룰 수 없습니다. 복잡한 것을 다루는 사고법이 필요합니다.

모델은 복잡한 것을 다루는 최고의, 보편적 방법이니 모델을 사용해서 사고할 필요가 있습니다.

1.7 모델링은 모델을 만드는 사고법(思考法)이며, 모델을 사용하는 사고법(思考法)입니다.

모델을 사용하려면 모델을 만들어야 합니다. 모델을 만들기 위해서도 사고가 필요하고 만들어진 모델을 사용하기 위해서도 사고가 필요합니다.
모델링은 문제를 이해하는 과정이고, 모델은 이해의 결과입니다.

1.8 메타모델은 사고법에 대한 모델입니다.

복잡한 대상에 대한 사고법이 그 자체로 복잡하다면, 사고법에 대해서도 모델링을 해야 합니다. 사고법에 대한 모델링 결과가 사고법 모델입니다.

사고법에 따라 사고하여 모델을 만들기 때문에, 모델은 사고법 모델에 따라 만들어진다고 할 수 있습니다. 모델을 만들 수 있도록 하는 모델을 **메타모델(meta model)**이라고 합니다. 모델보다는 한 차원 위에 있다는 의미를 반영한 것입니다. 높은 차원의 모델이 낮은 차원의 모델을 지배합니다.

1.9 객체모델은 소프트웨어 개발에 있어 최고의 사고법입니다.

객체모델은 소프트웨어 분야에서 갑자기 등장한 것이 아닙니다. 사람이 어떻게 사물에 대해서 인식하는지에 대한 축적된 인류의 지적 자산을 집적한 것입니다. 우리가 대상에 대해 사고할 때 객체모델을 벗어나지 못하는 것도 이러한 이유에서입니다.

객체모델은 모델이 기본이 되는 모델링 사고법입니다. 객체모델은 지금도 발전하고 있으며, UML 스펙을 통해 반영되고 있습니다.

모델은 표현하는 언어를 모델링 언어라고 합니다. UML은 다이어그램으로 표현하는 그래픽 모델링 언어입니다.

1.10 객체모델은 문제와 해결책 이해에 동일한 사고법을 사용합니다.

소프트웨어 개발은 문제도 복잡하고 해결책도 복잡합니다. 만약 문제에 대한 사고법과 해결책에 대한 사고법이 다르다면 둘 사이의 매핑을 위한 사고법 또한 필요하게 됩니다. 문제와 해결책만으로도 충분히 복잡한데 매핑이라는 복잡도까지 더하게 됩니다.

객체모델은 이러한 문제를 충분히 인식하고 문제와 해결책에 동일한 사고법을 사용하도록 합니다.

오늘날 소프트웨어 산업에서 널리 사용되고 있는 객체기술은 객체모델을 근간으로 하고 있습니다. 따라서 객체모델로 문제를 이해하고 객체기술로 해결책을 구축한다면, 문제와 해결책에 대해 동일한 사고법을 사용 할 수 있게 됩니다.

1.11 소프트웨어모델링은 객체모델이라는 메타모델에 따라 소프트웨어 모델을 만들어 가면서 소프트웨어 개발이라는 복잡한 문제해결을 다루는 사고법입니다.

객체모델링은 객체모델에 따라 사고하고, 사고의 결과를 객체모델로 표현합니다.

2절. 무엇을 알아야 하는가?

2.1 객체모델을 구성하는 구성요소의 정확한 개념과 정확한 표현을 알아야 합니다.

객체모델에 따라 사고하고 사고의 결과를 표현한다는 것은 객체모델을 사고와 표현을 위한 언어로 사용한다는 것입니다.

객체모델을 구성하는 구성요소 하나하나는 어휘를 구성하는 단어에 해당하기 때문에 어떤 의미를 갖는지, 어떻게 읽고 쓰는지를 정확히 알아야 합니다.

2.2 객체모델은 UML 명세서로 명세 되고 있습니다.

오늘날 대부분의 객체모델링 방법들은 객체모델을 구성하는 구성요소들이 갖고 있는 개념을 정확히 설명하지 못합니다. 10년이 훨씬 더 지난 객체모델링 초기의 불명확한 개념을 그대로 설명하거나, 설명이 어려운 것은 몰라도 된다는 식으로 넘어갑니다.

객체모델은 UML 등장 이후, UML 명세서로 명세 되고 있습니다. 따라서 객체모델을 구성하는 구성요소들에 대한 개념과 표현은 UML 명세서를 따를 경우 좀 더 분명해 질 수 있습니다.

2.3. 객체모델링 기법을 알아야 합니다.

UML 명세서는 객체모델을 구성하는 구성요소들과 그들 사이의 관계를 분명하게 명세하고 있지만, 어떻게 객체모델링을 해야 하는지에 대해서 명시적으로 기술하지 않고 있습니다. 따라서 별도의 객체모델링 기법을 알아야 합니다.

2.4 객체모델링 기법은 개념에 투명하고 논리적이어야 합니다.

오늘날 대부분의 객체모델링 방법들은 개념 따로 기법 따로 움직입니다. 개념과 기법이 따로따로인 이러한 엉터리 객체모델링 방법을 익힌 사람들에게 나타나는 공통된 특징이 있습니다.
모델링을 하고 나서 불안해합니다. 제대로 했는지 자신이 없기 때문에, 자기보다 경험이 좀 더 많거나 잘 알 것 같은 사람에게 자신의 모델이 제대로 된 것인지 묻습니다.
그런 경험의 횟수가 늘면서 모델링은 답이 없다고 생각하게 됩니다. 답이 없기 때문에 경험이 중요하다고 생각합니다.

구성요소에 대한 개념을 분명히 하고, 그 개념의 범위를 벗어나지 않도록 구성된 기법을 익혔다면 검증은 필요 없게 됩니다. 개념 자체가 기법의 올바름을 정당화해줍니다. 기법이 개념에 투명해집니다. 이러한 방법을 익힌 사람이라면 동일한 대상에 대해 동일한 결과를 낼 것이기 때문에 경험은 검증에 있어서 중요하지 않게 됩니다. 경험은 속도를 높여줄 뿐입니다.
구성요소의 개념과 기법이 서로 논리적인 연결고리를 갖고, 그 연결고리가 분명하게 설명된다면 모델링은 경험적인 것이 아니라 논리적인 것이 됩니다.

우리는 개념에 투명하고 논리적인 객체모델링 기법으로 행위형식화를 제시하고, 행위형식화를 통해 모델을 만듭니다.

2.6 모델을 만들고 모델을 사용할 때 사용할 수 있는 사고법들을 알아야 합니다.

우리는 행위형식화를 위한 다양한 사고법들을 제시합니다.

2.7 소프트웨어모델이 어떻게 소프트웨어로 구현되는지 알아야 합니다.

소프트웨어모델링은 소프트웨어를 개발하기 위한 것으로 궁극적으로 소프트웨어로 구현되어야 합니다.

3절. 소프트웨어개발주기와 소프트웨어모델링

3.1 문제해결 과정은 분석, 설계, 검증 구현 단계를 거칩니다.

문제를 해결하기 위해서 문제를 이해하고 해결책을 설계합니다. 문제를 이해하기 위해 문제를 분석합니다. 설계된 해결책은 검증 후 구현됩니다. 소프트웨어 개발에 있어 설계 검증 구현은 동시적으로 일어나기도 합니다.

3.2 소프트웨어모델링은 복잡한 대상을 다루는 사고법입니다. 특정한 개발 단계가 복잡하면 모델링을 할 수 있습니다.

어떤 소프트웨어 개발 조직들을 보면 모델링 직군이 나누어져 있는 것을 볼 수 있습니다. 이것은 모델링이 복잡성을 다루기 위한 사고법이라는 것을 모르기 때문에 발생하는 것입니다.
모델링은 복잡성을 다루기 위한 사고법입니다. 사고법을 다루어 메타모델을 만드는 조직이 아니라면 모델링 팀은 있을 수 없습니다.

소프트웨어개발주기의 각각의 단계들이 복잡하면 어느 단계든 모델링을 할 수 있습니다.
분석단계가 복잡하면 분석모델을 만들고, 아키텍처단계가 복잡하면 아키텍처모델을 만들고, 상세설계나 시험이나 구현이 복잡하면 이들에 대해서도 각각 모델을 만들 수 있는 것입니다.
특정한 직군이 모델링을 하는 것이 아니라 소프트웨어 개발 단계의 어떤 역할을 수행하든 복잡하면 모델링을 할 수 있는 것입니다.

도메인 전문가는 도메인의 문제를 도메인의 개념들을 사용해서 설명합니다. 소프트웨어 개발자는 도메인에 대한 이해의 결과를 도메인 모델로 작성합니다. 소프트웨어 개발자는 도메인 모델을 가지고 도메인 문제를 이해하고 또 다시 이해의 결과를 모델(문제 모델)로 작성한다.

소프트웨어 개발자는 모델을 검증하기 위해 시뮬레이션 합니다. 시뮬레이션의 결과는 도메인 전문가의 설명을 재현하는 것입니다.

프로그래밍 또한 모델링입니다. 프로그래밍을 통해 문제를 해결할 수 있는 해결책을 모델링합니다. 도메인의 세계를 소프트웨어 세계에 모델링한 것입니다.

마치기 전에

이번 장을 통해 우리는 기존의 설명들과는 다른 '사고법으로서의 소프트웨어 모델링'을 이야기했습니다. 뭐가 뭔지도 명확히 설명할 수 없는 설명들은 이제 버리고 '사고법으로서의 소프트웨어 모델링'을 취할 때입니다.

소프트웨어모델링은 소프트웨어 개발을 위한 어떤 한 과정이 아니라, 소프트웨어 개발이 갖고 있는 복잡성을 다루기 위해 요구될 수 밖에 없는 사고과정입니다.

소프트웨어 개발에 있어 가장 강력한 힘은 생각하는 힘입니다. 소프트웨어모델링은 모델을 만들어 가면서 또 만들어진 모델을 가지고 생각하는 힘을 극대화하는 것입니다.

소프트웨어모델링은 아무 때나 하는 것이 아니라 대상이 복잡할 때 하는 것입니다.

복잡한 대상인데도 불구하고 소프트웨어모델링을 하지 않는 것은 인류의 지적 자산을 무시하는 것이며 전쟁터에 총을 들고 가지 않는 것과 같습니다.

소프트웨어모델링을 복잡한 대상을 다루는 사고법으로 패러다임을 전환하면, 소프트웨어모델링은 특정 팀/역할이 하는 것이 아니라 소프트웨어개발주기의 어떤 단계라도 복잡하면 할 수 있는 것이 됩니다. 복잡성을 다루는 누구에게나 필요한 사고 기술이 되는 것입니다.