Chapter 1 ドメイン駆動設計とは

ドメイン駆動設計が目指すゴールと本書が目指すゴールを確認します。

ドメイン駆動設計はエリック・エヴァンス氏が提唱する設計手法です。書籍『エリック・エヴァンスのドメイン駆動設計』に端を発したこの考えは、いまやソフトウェア開発の世界に大きな影響を与えています。
本章ではまずドメイン駆動設計が何かを知ることから始め、本書が提示するゴールとそこへの道のりを確認します。各章で紹介するトピックの概要とそれらの関係性についての説明は本書を読み進めるあなたの道しるべとなるでしょう。

DDD 1.1 ドメイン駆動設計とは何か

ソフトウェアを開発するとき、私たちは新たに何かを学びます。たとえば会計システムを開発するのであれば経理について学ぶでしょう。物流システムを開発するのであれば輸送や配送について学びます。ソフトウェアシステムを開発する上で学んだ知識は汎用的な知識であることもあれば、ある組織特有の知識である場合もあります。

開発者はソフトウェアの利用者を取り巻く世界について基本的に無知です。彼らの問題を解決するために開発者が彼らの世界を学ぶことは当然ながら必要なことです。しかしながら、そのようにして学んだ知識がそのままソフトウェア開発の役に立つことは稀です。

たとえば物流システムを例に考えてみましょう。トラックの積載容量や燃費などの概念は物流システムにとって利用価値の高い知識です。しかし、トラックの語源がラテン語のtrochusで、その意味は「鉄の輪」であるといった知識は物流システムにとってほとんど無価値でしょう。知識は取捨選択される必要があります。

利用者にとって役に立つソフトウェアを開発するためには、価値ある知識と無価値な知識を慎重に選り分けて、選び抜かれた知識をコードに落とし込む必要があります。そうした手順を踏んで作り上げられたコードは有用な知識が込められたドキュメントの様相を呈してきます。

開発者がソフトウェアを開発するために必要な知識と不要な知識を選り分けるには何が必要でしょうか。当たり前のことですが、ソフトウェアの利用者を取り巻く世界について知る必要があります。ソフトウェアの利用者にとって重要な知識が何であるのかは、その世界の有り様に左右されるのです。価値あるソフトウェアを構築するためには利用者の問題を見極め、解決するための最善手を常に考えていく必要があります。ドメイン駆動設計はそういった洞察を繰り返しながら設計を行い、ソフトウェアの利用者を取り巻く世界と実装を結びつけることを目的としています。

あなたが学んだ知識はそれが何であろうと、貴重なあなたの人生の時間をいくばくか費やした、とても大切なものです。知識がコードに埋め込まれ、ソフトウェアとなって直接的に誰かを支援する。そこに喜びを覚えない開発者はいないでしょう。ドメイン駆動設計はまさに知識をコードへ埋め込むことを実現するのです（図1.1）。

図1.1：知識をコードへ

DDD 1.2 ドメインの知識に焦点をあてた設計手法

ドメイン駆動設計はその名のとおり、ドメインの知識に焦点をあてた設計手法です。この説明はすぐに次の疑問を呼び起こします。すなわち「ドメインとは何か」という疑問です。まずはしっかりとドメインという言葉が何なのかということから確認をしていきます。

ドメインは「領域」の意味をもった言葉です。ソフトウェア開発におけるドメインは、「プログラムを適用する対象となる領域」を指します。重要なのはドメインが何かではなく、ドメインに含まれるものが何かです。

たとえば会計システムを例にしてみましょう。会計の世界には金銭や帳票といった概念が登場します。これらは会計システムのドメインに含まれます。物流システムであればどうでしょうか。会計システムとは打って変わって貨物や倉庫、輸送手段などの概念が存在し、それらがそのまま物流システムのドメインに含まれます。このようにドメインに含まれるものはシステムが対象とするものや領域によって大きく変化します（図1.2）。

次に「知識に焦点をあてる」というのはどういうことでしょうか。

ソフトウェアにはその利用者が必ず存在します。ソフトウェアの目的は利用者のドメインにおける何らかの問題の解決です。彼らが直面している問題を解決するには何が必要でしょうか。当たり前のことですが、「彼らが直面している問題」を正確に理解することが必要です。利用者が何に困っていて何を解決したいと考えているのかを知るには、彼らの考えや視点、取り巻く環境を真に理解する必要があります。つまりドメインと向き合う必要があるのです。

ドメインの概念や事象を理解し、その中から問題解決に役立つものを抽出して得

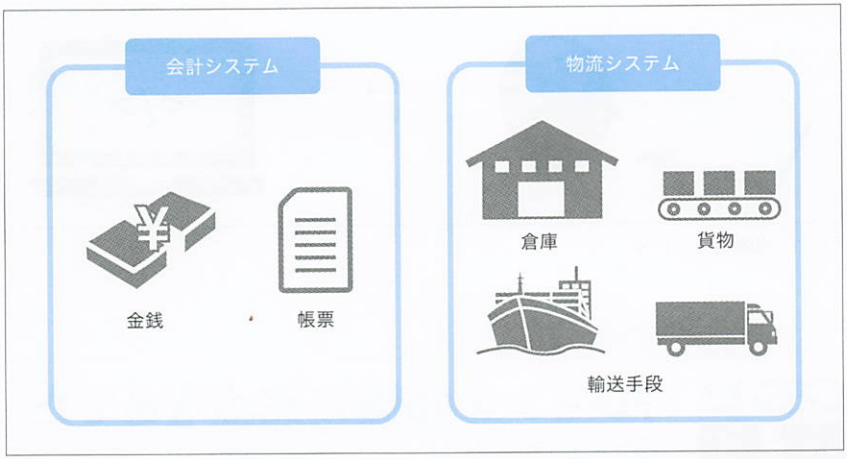

図1.2：システムごとのドメインに含まれる要素

られた知識をソフトウェアに反映する。こういったことはソフトウェアを開発する上で当たり前の行為です。しかし同時に技術指向の開発者であればあるほど疎かにしやすい工程でもあります。

たとえば最新のフレームワークや開発手法、最新技術といったワードは開発者の心を躍らせるものです。本来であれば問題を解決するにはその問題と向き合うことが求められますが、技術指向の開発者は技術的なアプローチで解決を図ろうとしてしまいがちです。結果としてできあがったものが的外れなソフトウェアでは目もあてられません。ピカピカのハンマーは開発者の目を曇らせ、見るものすべてを釘に変えてしまうのです。

こういった悲惨な結果を招かないためにも、ソフトウェアを適用する領域（ドメイン）と向き合い、そこに渦巻く知識に焦点をあてる必要があります。よく観察し、よく表現すること。ソフトウェアを構築する上で当たり前の行為です。しかし当たり前のことを実践することこそが難しいのです。ドメイン駆動設計のプラクティスはその実践を補佐するでしょう。ドメイン駆動設計はいわば当たり前を当たり前に実践するための開発手法なのです。

1.2.1 ドメインモデルとは何か

モデルという言葉は開発者にとって身近な言葉です。世に多く存在するソフトウェア開発に関する文献を紐解いてみれば、頻繁といっても差し支えないほどの頻度でモデルという言葉に出会えることでしょう。書籍『エリック・エヴァンスのド

メイン駆動設計』においてもこれは同様で、モデルという言葉は第1部の部題（第1部　ドメインモデルを機能させる）に登場しています。

開発者にとって身近な存在のモデルですが、さてモデルとは何でしょうか。改めて問われると、案外答えづらいものではないでしょうか。

モデルとは現実の事象あるいは概念を抽象化した概念です。抽象は抽出して象るという言葉のとおり、現実をすべて忠実に再現しません。必要に応じて取捨選択を行います。何を取捨選択するかはドメインによります。

たとえばペンはどのような性質を抽出すべきでしょうか。小説家にとってペンは道具で、文字が書けることこそが大事な性質です。一方、文房具店にとってペンは商品です。文字が書けることよりも、その値段などが重要視されます。このことが指し示すのは、対象が同じものであっても何に重きを置くかは異なるということです（**図1.3**）。

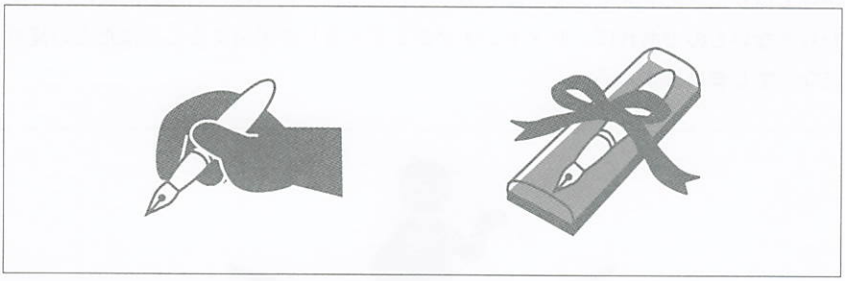

図1.3：道具としてのペンと商品としてのペン

人の営みは根本的に複雑です。ドメインの概念を完全に表現しきることはとても難しいです。何かと制約の多いコードで表現するとなれば尚のことです。しかし、ソフトウェアがその責務を全うするために必要な情報に限定をすれば、コードで表現することも現実的になります。たとえば物流システムにおいて、トラックは「荷運びできる」ことを表現すればそれで十分です。「エンジンキーを回すとエンジンがかかる」といったことまで表現する必要はありません。

こういった事象、あるいは概念を抽象化する作業がモデリングと呼ばれます。その結果として得られる結果がモデルです。ドメイン駆動設計では、ドメインの概念をモデリングして得られたモデルをドメインモデルと呼びます。

ドメインモデルはこの世界に初めから存在しているわけではありません。ドメインの世界の住人はドメインの概念についての知識はあっても、ソフトウェアにとって重要な知識がどれかはわかりません。反対に開発者はソフトウェアにとって重要な知識を判別できても、ドメインの概念についての知識がありません。ドメインの

問題を解決するソフトウェアを構築するために、両者は協力してドメインモデルを作り上げる必要があります。

1.2.2 知識をコードで表現するドメインオブジェクト

ドメインモデルはあくまでも概念を抽象化した知識にとどまります。残念ながら知識として抽出しただけでは問題を解決する力はありません。ドメインモデルは何かしらの媒体で表現されることで、問題解決の力を得ます。

ドメインモデルをソフトウェアで動作するモジュールとして表現したものがドメインオブジェクトです。

どのドメインモデルをドメインオブジェクトとして実装するかは重要な問題です。開発者は真に役立つモデルを選り分ける必要があります（図1.4）。長い時間をかけて作り上げたドメインモデルであっても、それが利用者の問題の解決に何ら関わりのないものであれば、ドメインオブジェクトとして実装することはただの徒労になってしまいます。

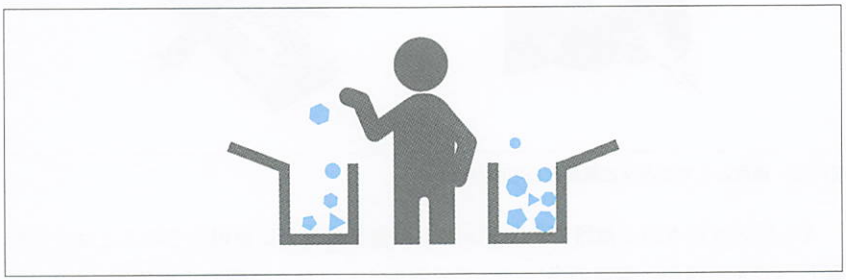

図1.4：ドメインモデルの取捨選択

ソフトウェアの利用者を取り巻く世界が常に凝り固まったものとは限りません。人の営みは移ろいやすく、ときの流れと共に変化します。変化の多くは軽微なものが積み重なるものですが、ときには常識すらも変わるでしょう。そんなときドメインオブジェクトがドメインモデルを忠実に表現していれば、ドメインの変化をコードに伝えることはたやすいです。

ドメインで起こった変化はまずドメインモデルに伝えられます。

ドメインの概念の射影であるドメインモデルは、変化を忠実に受け止めます。ドメインオブジェクトはドメインモデルの実装表現ですから、変化したドメインモデルと変化していないドメインオブジェクトの両者を見比べてみれば、自ずと修正点は浮き彫りになるでしょう。ドメインの変化はドメインモデルを媒介にして連鎖的

にドメインオブジェクトまで伝えられるのです。

また反対にドメインオブジェクトがドメインに対する態度を変化させることもあります。プログラムは人の曖昧さを受け入れられません。ドメインに対する曖昧な理解は実装の障害となります。それを解決するにはドメインモデルを見直し、ひいてはドメインの概念に対する捉え方を変える必要があるでしょう。ドメインに対する鋭い洞察は実装時にも得られるものなのです。

かくしてドメインの概念とドメインオブジェクトはドメインモデルを媒介に繋がり、お互いに影響し合うイテレーティブ（反復的）な開発が実現されます（図1.5）。

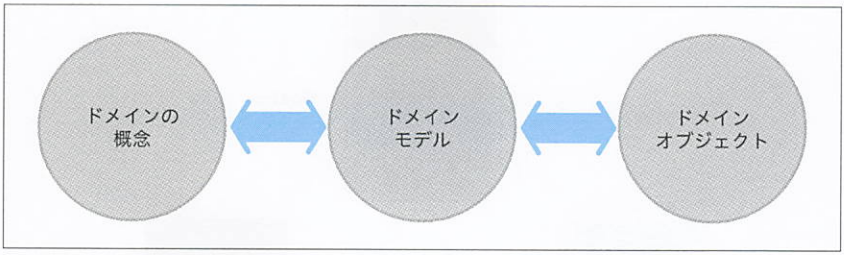

図1.5：イテレーティブな開発

DDD 1.3 本書解説事項と目指すゴール

ドメイン駆動設計を理解するのは難しいといわざるを得ません。ドメイン駆動設計の学習を進めるとまるで翻弄するかのように多くの概念や用語が出てきます。これは多くの初学者を混乱させ、恐怖に陥れます。

知識は連鎖するものです。ある知識を得るために、前提として異なる知識が求められることは多くあります。ドメイン駆動設計で語られる概念や用語を理解するためには、その結論に至る過程で得られる多くの前提知識を要求します。ひとつひとつは些細な知識であったとしても、それがいくつもとなると対応するのは難しいものです。

もうひとつ重大な問題があります。ドメイン駆動設計のプラクティスにはそもそも実践の難しいものが存在するということです。百聞は一見に如かずという言葉もあるとおり、知識を理解に落とし込むための最善の手段は実践です。残念ながらドメイン駆動設計のプラクティスには、実践するためにある程度の環境を要求するも

のも存在します。

とはいえ、理解が難しく実践も難しいとあってはいつまで経ってもドメイン駆動設計の世界に踏み込めません。そこで本書では概念の理解や実践が難しいものを一旦棚上げし、理解と実践がしやすい実装に関するパターンに集中してボトムアップで解説していきます。概念の前提となる知識も、その都度必要に応じて解説します。そうして少しずつ理解の領地を広げていって、ドメイン駆動設計の本質に立ち向かう準備を完了することをゴールとします（図1.6）。

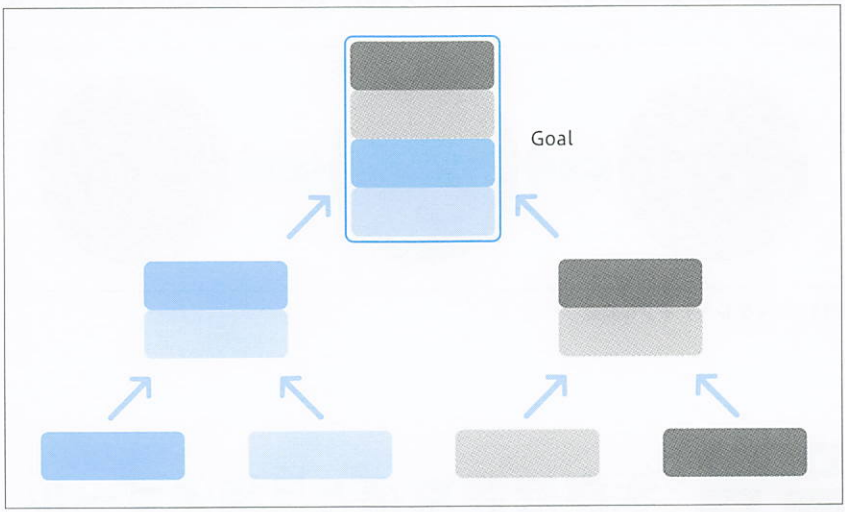

図1.6：ボトムアップでゴールに向かう

COLUMN

ドメイン駆動設計の実践を難しくするもの

ドメイン駆動設計はソフトウェア開発をテーマにしています。ソフトウェア開発の現場には実際に手を動かすかの違いがあるものの、関係者が複数存在します。したがって、ドメイン駆動設計のトピックは実装だけに留まらず、関係者とのコミュニケーションやチームビルディングと密接に関わるものが存在します。つまり開発者個人に収まらず、多くの関係者を巻き込む必要があるのです。

たとえばドメインモデルを形作る作業を開発者だけで完結することは不可能です。ドメインの概念に対する捉え方はドメインの実践者の視点が欠かせません。開発者はドメインの実践者の助力を得る必要があります。しかし、残念ながらそれが叶わない現場も多いでしょう。

もしあなたがそういった環境に置かれていても悲観することはありません。ドメイン駆動設計のすべてのトピックがそういった類のものとは限りません。ドメイン駆動設計はソフトウェア開発を包括的に取り扱った設計手法です。そこには開発者個人の裁量で実践できるプラクティスも多くあります。本書で学ぶパターンはまさに今すぐ実践できる類のものです。

ドメイン駆動設計に関わらず、設計にはある種の理想としての側面があります。重要なのは理想を無理やり現実に当て込むことではなく、現実に適合させるために取るべき手段を考え選択することです。理想を掲げ、妥協しなくてはならないときに、どこを妥協するのか悩むのもまたソフトウェア開発の楽しみ方の1つです。

ドメイン駆動設計は当たり前のことを当たり前にやるためのプラクティスです。当たり前とはどういうことなのか。それを実現するための手法を知っているのと知らないのでは大きな違いです。もし、あなたがドメイン駆動設計のすべてを実践できない環境にいたとしても、あなたが開発者として学びを得るためにドメイン駆動設計を選んだのは決して間違いではありません。

DDD 1.4 本書で解説するパターンについて

目的地を知らされずに道を辿ることは、さながら迷路を彷徨うようなものです。ドメイン駆動設計に関わらず、同じ道のりであっても、目的地までどれくらいなのか、いま自分がどこにいるのかさえわかっていればペース配分を考えられます。ここで一度、本書で学ぶドメイン駆動設計のパターンを俯瞰しておきましょう。

- 知識を表現するパターン
  - 値オブジェクト
  - エンティティ
  - ドメインサービス
- アプリケーションを実現するためのパターン
  - リポジトリ
  - アプリケーションサービス
  - ファクトリ
- 知識を表現する、より発展的なパターン
  - 集約
  - 仕様

これらはドメイン駆動設計に登場する代表的なパターンです。本書ではこの一覧の順序で解説していきます。

パターンはドメインの知識を表現するためのパターンとアプリケーションを実現するためのパターンに分けられます。これらの関係性を図に表したのが**図1.7**です。

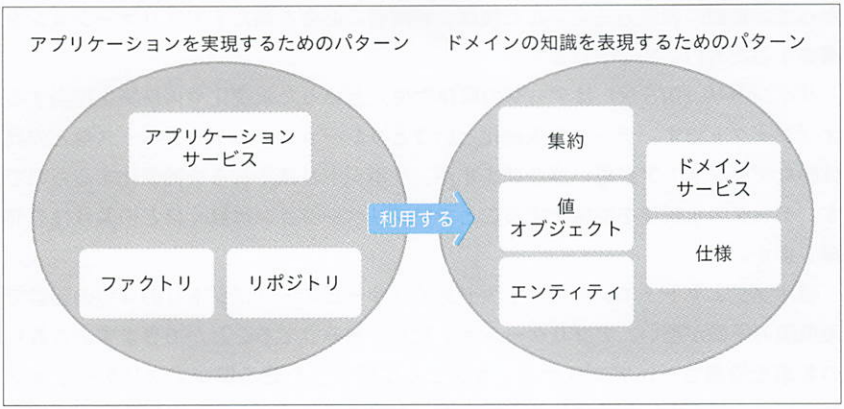

図1.7：用語の関連性

1.4.1 知識を表現するパターン

最初に解説するのは知識を表現するパターンです。まずはドメインの知識をオブジェクトとして表現するドメインオブジェクトを理解することから始めます。

値オブジェクト（第2章）はドメイン固有の概念（金銭や製造番号など）を値として表現するパターンです。その概念や目的はとても理解しやすいものです。まさにドメイン駆動設計を学ぶ足掛かりとして最適な概念です。

次に学ぶのはエンティティ（第3章）です。エンティティは値オブジェクトと同じくドメインの概念を表現するオブジェクトですが、値オブジェクトと対を為すような性質があります。値オブジェクトで学んだことはそのままエンティティを理解するのに役に立ちます。エンティティを学ぶのは値オブジェクトを学んだ直後が最適です。

ドメインサービス（第4章）は値オブジェクトやエンティティではうまく表現できない知識を取り扱うためのパターンです。値オブジェクトとエンティティがどういったもので、どういったことができるのかを確認してから学習に臨むと理解がしやすいため、値オブジェクトとエンティティを把握した後に学びます。

1.4.2 アプリケーションを実現するためのパターン

ドメインの知識を表現しただけではドメインの写しがコードとして表現されているだけに過ぎません。ソフトウェアに求められる必要は満たされていません。した

がって、知識の表現方法を学んだ後は、利用者の必要を満たすアプリケーションを構築するための手法を学びます。

リポジトリ（第5章）はデータの保存や復元といった永続化や再構築を担当するオブジェクトです。データの永続化というとリレーショナルデータベースなどの具体的なデータストアが思い浮かびますが、リポジトリはそれらを抽象化するものです。データの永続化を抽象化することでアプリケーションは驚くほどの柔軟性を発揮します。

値オブジェクト・エンティティ・ドメインサービス・リポジトリの４つの要素で最低限の準備が整い、アプリケーションとして組み立てることができます。これらの要素を協調させ、アプリケーションとして成り立たせる場がアプリケーションサービス（第6章）と呼ばれるものです。本書ではアプリケーションサービスについて学んだのちに、実際に動作可能なWebアプリケーションへ組み込みを行います。

Webアプリケーションとして動作させるところまで確認した次はファクトリ(第９章) を学びます。ファクトリはオブジェクトを作る知識に特化したオブジェクトです。複雑な機構をもつものは得てしてその生成方法も複雑になりがちで、それはドメインオブジェクトであっても同じです。複雑なオブジェクトの生成方法はある種の知識となります。オブジェクトの生成は至るところで発生します。対策せずにいると煩雑な手順はコードの至る所にばらまかれ、処理の趣旨をぼやけさせるでしょう。ファクトリを利用して生成に関する知識を一箇所にまとめあげることは、ドメインオブジェクトを際立たせ、処理の趣旨を明確にするために必要なことです。

1.4.3 知識を表現する、より発展的なパターン

集約と仕様は知識を表現するオブジェクトですが、より発展的なパターンです。ここまでの内容にひとしきり慣れた上で学習に臨むとよいでしょう。

集約（第12章）は整合性を保つ境界です。値オブジェクトやエンティティといったドメインオブジェクトを束ねて複雑なドメインの概念を表現します。前提知識をいくらか要求されるため理解することが難しく、同時に正しく実践することが難しい代物です。もちろん前提知識として必要なものはここまでの内容で揃っています。ここまでたどり着けたのであればきっと乗り越えることができる壁です。

集約の後に学ぶ仕様（第13章）はオブジェクトの評価をします。オブジェクトがある特定の条件下にあるかを判定する評価のふるまいをそのオブジェクト自身に実

装すると、オブジェクトが多くの評価のふるまいに塗れて煩雑になってしまうことがあります。仕様を適用することで、オブジェクトの評価をモジュールとしてうまく表現できます。

本書は構成上、前後の章が繋がっています。各章は単独でも理解できる内容ですが、順序に沿って読み進めると、より理解に繋げやすくなることでしょう。

COLUMN

なぜいま、ドメイン駆動設計か

ドメイン駆動設計が提唱されたのは2003年頃です。ITという分野の進化は目覚ましく、最新の技術も十年過ぎれば陳腐化してしまうといったことは往々にして発生します。にもかかわらず昨今のシステム開発の現場において、ドメイン駆動設計という言葉を耳にする機会が増えたのには、いったいどのような背景があるのでしょうか。

ひと昔前はサービスをいち早く世に出すことこそがもっとも重要なこととされていたように感じます。そのためモデリングに重きを置き、開発の最初期にコストを支払うドメイン駆動設計は重厚で鈍重なものであると誤解され、敬遠されていました。

とにかく早くサービスを打ち出すことはほとんどミサイルのような片道ロケットに乗ることに似ています。打ち上げた後に帰ってこれないという欠点に目を瞑れば、システム開発の過酷な生存競争に勝つために取れる最良の選択肢でしょう。それに対してモデリングをしっかりと行い、長期的な運用を視野に入れた設計手法は飛行機に運用するようなことです。飛行機は片道ロケットと違って往復することは可能ですが、その速度は圧倒的に見劣りします。それにもかかわらず、なぜ私たちは片道ロケットの打ち上げ競争を辞めて、飛行機を安定運用したいと願うようになったのでしょうか。

ソフトウェアは変化するものです。ごく最初期の局所的な開発速度を優先したソフトウェアは、柔軟性に乏しく、変化を吸収しきれません。ソフトウェアに求められる変化に対応するために、開発者は継ぎはぎのような修正を重ねます。数年もすればソフトウェアは複雑怪奇な進化を遂げるでしょう。それでも時代の変化についていくために、開発者は辟易しながら継ぎはぎだらけの修正を積み重ねるのです。場当たり的な対応に嫌気の差した開発者たちが片道ロケットの打ち上げ競争ではなく、飛行機の安定運用を願うようになるのも想像に難くありません。救いを求めて手にしたものの中にドメイン駆動設計がありました。

ドメイン駆動設計はドメインと向き合うことで分析から設計、そして開発までが相互作用的に影響し合うよう努力を重ねることを求めます。ソフトウェアを構築する最初期においても一定の効果はありますが、その真価は変化に対応するときにこそ表れます。ドメイン駆動設計を取り入れてみた当時はさほど効果が見られなかったでしょう。ときが流れ、段々とドメイン駆動設計が認められてきたのは、偉大なる先人たちによって撒かれた種が芽吹いてきたからに他なりません。

プログラムは動かすだけなら簡単で、しかし動かし続けることは難しい代物です。システムを長期的に運用したいと願うのならば、安定的な飛行機の運用を願うのならば、ドメイン駆動設計をいまこそ学ぶべきでしょう。