# 連載　企業および社会における情報システムの意味を考える
# 第８回　ソフトウェア開発とモノ作りとの違い

大島　正善（MBC:Method Based Consulting）

## 1. ソフトウェアとモノとの違い

ソフトウェアの開発は約半世紀の歴史があります。製造業の歴史が産業革命からとしても２５０年経ちました。モノ作りという意味では、そもそも人類の歴史が開始された直後から始まったと考えることもできます。そうすると、数１０万年という歴史があることになります。

ソフトウェア開発には混乱がつきものであり、知的産業だといわれながら労働集約的である面は否めません。ソフトウェア開発は製造業のモノ作りを真似ようとしていますが、そう簡単ではないようです。ソフトウェア開発は芸術であり画家や陶芸家あるいは建築家のように、天才にしか優れた設計はできないと考える人たちもいます。そうはいっても、陶芸、建築においても工業化され平準化された世界は存在します。ソフトウェアの世界では、そういった平準化はいまだにかなり困難です。

一般のモノ作りとソフトウェア開発は、何が違うのでしょうか?ソフトウェア開発を製造工程だと思うのが間違いで、モノ作りの設計工程あるいは研究開発工程に相当するのだという考え方もあります。その点は、後ほど触れたいと思いますが、最初は、そういった工程論ではなく、作る対象である「モノ」と「ソフトウェア」とは何が根本的に違うのか?という点に焦点をあててみます。

よく語られることですが、「モノは目に見えるが、ソフトウェアは目に見えない」という見解があります。確かに、ソフトウェアという構造物は、普通のモノとは違って形がありません。コードというソフトウェア言語による構造物です。たとえて見れば、推敲に推敲を重ねて破綻のない完璧な論理展開を見せる推理小説のようです。それが、機械を動かすというマジックを生み出すのですからわかりにくいのは確かです。

しかしながら、「目に見えないから一般のモノ作りとは違って難しい」というのは、その意味をもう少し掘り下げて考えてみる必要がある命題です。「目に見えない」とはどういうことでしょうか?一風変わった言語で記述されますが、記述されたコードを読むことはできます。何が目に見えないのでしょうか?ソフトウェアの構造を示す設計文書の表記法も、業界で統一されているわけではありませんが、ないわけではありません。

それに、「目に見えない」ことを困難な理由とすると、重力、分子、原子、素粒子、

遺伝子などを扱う学問は生まれなかったでしょう。目に見えないことを困難の理由にするのは、生産的ではないように思います。

ソフトウェアの設計・開発の困難さを生み出している「目に見えない」というのは単に現象にすぎず、本当の原因はその奥に隠されているのではないかと思っています。その原因として以下の二つのことを挙げたいと思います。

① **ソフトウェアが抽象物である**

② **単位がない**

2. 抽象物であるソフトウェア開発の難しさ

ソフトウェアは、「抽象化された概念」を設計図に表現したものです。抽象化されたものなのでわかりにくい（それを「目に見えない」と表現している）ということです。はじめに、そのことについて考えてみます。

ソフトウェアの機能は、現実を抽象化したものとして定義したものだということは、今までの連載でも述べてきました。それも、一度だけでなく何段階も抽象化を行った上で初めて定義されるものだということです。抽象物であるということを別の観点でみると、複雑な現実（たとえば、ビジネスの仕組み、モノ作りの仕組み、車の運転［組み込みソフトウェア］、人の動き［ロボット制御］など）を別の次元に写像したものだともいえます。基本的に社会の仕組み（人間そのものも含め）の多くは“情報システム”として理解することができますが、そういった複雑な現実の対象をソフトウェアで実現するときに抽象化が行われます。

製造物であるモノは抽象物ではありません。設計図上で「ネジ」と言葉で表現したときは、「ネジという種類」という意味で抽象概念となりますが、製造工程で部品となる「ネジ」は抽象物ではなく、まさに具体的な「ひとつのネジ」です。モノに取り付けるネジが１０個あったとすると、すべて別のネジです。一般の製造工程はこのようにひとつひとつが実態を持つモノを扱います。ソフトウェアの設計・開発は違います。設計も開発する対象も最初から最後まで、抽象物です。

ソフトウェアが抽象物であるということは、わかりにくさだけでなく、その組み立て（親、子、孫モジュールを結合してひとつの機能を実現すること）に一般の製造物とは全く違った様相が出現します。そのことが、ソフトウェアの設計・開発を複雑にします。それが根本的な問題となります。

3. 抽象物の開発の難しさ

モノの世界でも、“設計”は抽象化された対象を扱います。しかし製造工程は物理的な世界を扱います。ソフトウェアは、“設計”も“開発”も抽象化された世界を扱います。その意味では、コードを作ることを「開発」と表現していますが、それも「設計」といったほうがモノ作りと比較するうえでは適切なのではないかと思えます。

抽象化されたモノを設計し開発することの困難さは、単体のモジュールの開発ではなくそれを組み合わせるときに発生します。一般のモノの場合も同じですが、ある部品の「設計」に間違いがあれば、その部品を使っている製品すべてに影響が出ます。ソフトウェア開発では、単体部品は、それ自体が完成しても進捗は１００％完了とはなりません。結合したら動作しないことは頻繁に発生し、その時点で進捗はゼロになる可能性があります。一般のモノ作りで製造工程に入ってから設計ミスが頻繁に発生したらその部品は別の部品に取り替えられるでしょう。

ソフトウェアでは、設計変更は開発工程でも発生することが多々あります。設計変更の理由は、その上流である業務要件の変更が発生したり、サブシステム間やサービス間に設計変更が発生したりすることが大きな理由です。現代はビジネスの変化が速い時代ですから、設計や開発の途中でも正当な要求の変化に基づく仕様変更も頻繁に発生すると考えておく必要があります。設計が紙ベースで行われ、多くの技術者が個々に自分の守備範囲を確実に守ることを理由に、仕様変更の受け入れに難色を示すのは間違いです。

ソフトウェアの開発が厄介なのは、設計変更を止めることができない上に、１つの部品（モジュール）が他の多くの部品と関連をもっていることに大きな原因があります。プログラムの作成作業を“開発”という言葉で説明していますが、モノの“製造”とは異なり、“設計”を行っているのと同じです。物理的なモノの世界の製造では、ひとつの物理的なモノは、特定のモノとしか関係を持ちません。１本のネジは特定のモノ同士をつなぐためにしか用いられません。したがって、そのモノに問題があれば（設計の問題ではなく）、取り換えれば済むことです。自動車、家電製品や建造物のみならず、服の生産、弁当の生産、陶器の生産、椅子、机の生産などみな同じです。ある部品や材料に問題があっても設計上の問題でなければ、それを捨てて別のモノを使えば済みます。ソフトウェアはそのようにはできていません。捨てることはできず、それが完治するまで待たなければなりません。トヨタ流であれば、ムダな在庫を作るだけとなるので、工程をストップさせるということになります。今のやり方でソフトウェアを設計し開発をすると、とてもトヨタ流でやるわけにはいきません。そんなことをしたら、まったく前に進まなくなるでしょう。

ソフトウェア開発のやり方を一般のモノの製造のやり方と比較することがよく行われ、なぜ、一般の製造物のようにできないのか?と疑問をなげかけることがよくあります。私は、「モノ作りとは基本が違う」と考えています。モノと比較するのであれば、製造工程ではなく、設計や研究開発のやり方・仕組みと比較するほうが適切であるのではないでしょうか。

ソフトウェアは、概念の集まりである抽象物を扱いながら設計部品表（BOM：Bill Of Materials）も持たずに設計し開発を行っているという問題多き産業です。労働集約的になり、品質にバラツキが生じるのは当然です。いくらCMMIでレベル５の認証を受けたといっても、たかが知れています。抜本的に設計・開発の考え方、見方、やり方を変革することが必要です。抽象物であるソフトウェアであればこそ、本来は部品表の作成が必須です。ソフトウェアの設計ツールはありますが、多くは単なる図の描画ツールであり、構造物の属性を管理するようには考えられていません。実態は紙に設計図を描いているのと同等です。ソフトウェアの構造全体を見渡せるような部品管理（構造管理）ツールが求められているといえるでしょう。

３月号の記事でも触れましたが、ソフトウェアが抽象物の部品でありながら、多くの他の部品と関連をもっているということが、機能のくくり方を難しいものにし、変化に対して対応の困難さを引き起こします。ひとつの機能の単位は、凝集度を高め、結合度を弱くするというのが基本原則ですが、それを集めた大きな機能群である、サービスやサブシステムをどう括っても、変更が及ぼす影響範囲を狭めることはできません。

現在は、経営環境の変化は数十年前にくらべてきわめて速く、情報システムはそういった変化に素早い対応ができることが求められています。ＳＯＡでは、「サービスは業務の視点で括った機能の単位」と言われていますが、業務の単位とソフトウェアの単位は今まで述べてきたようにそもそも違うので、そのようにするだけで変更に強く、拡張性や保守性が本当に向上するのか私は疑問を持っています。

サブシステムというのは、ソフトウェアのインターフェースを最小限に（しようとの意図で!）括った集まりですが、インターフェースの変更には弱い集まりでしかないという点では変わりがありません。サブクラス化が可能な言語を採用しても、全く異なるインターフェースになることもあり、破綻しないとは言いきれません。インターフェースという部品の設計を間違えたら、複数のサブシステムに影響が出ます。プログラミングでＡＰＩ（Application Programming Interface）にコーディング間違いがあり結合テストで発見された場合、それを在庫部品に取り換えてすぐ製造を続けるなんていう芸当は使えません。

製造業が理想とする「平準化した生産性を保った一個流し」で作りこむことは、抽象物を扱うソフトウェアの開発では無理があります。物理的なモノを扱う製造業の生産工程、管理のやり方、見積もり精度、生産性の基準、品質管理手法などを適用するのは困難が伴います。製造業と比べるのであれば、むしろ研究開発部門の研究の進め方や“設計”のやりかた（完成品のモックアップを必ず作るというやり方など）が参考になるのではないかと思います。

## 4. 単位がないことがソフトウェアの設計にもたらすこと

日常生活において単位がない世界は考えられません。目的地までどのくらいの距離があるのか、どの程度時間がかかるのか、歩いて行けるのかタクシーに乗るべきか、荷物の重さはどのくらいなのか、ツァー旅行の人数はどのくらいなのかなど、何か行動しようとしたときに必ず何らかの単位を考えています。モノ作りでも同じで、設計では材料や部品の量、長さ、厚み、高さ、重さなどを決めなければ設計とは言えません。

一方、ソフトウェアの世界では単位を決めることはありません。長さ、重量といった、標準化され合意された単位がソフトウェアの世界にはありません。このことは、ソフトウェア作りを困難なものにしているもう一つの大きな原因の一つだと思います。１機能の大きさ、１画面に表現する情報量、１帳票に表現する情報量などについて、「この程度の重量にすべし」というような設計基準を決めているという話は聞いたことがありません。

ステップ数というのがサイズに該当するように思えますが、それは、絶対基準ではない（単なるガイドにしかならない）ですし、複数のソフトウェアを比較するのに利用できないことはご存じの通りです。

ファンクションポイントというのがありますが、それも結局のところステップ換算と同等ですから比較には使うのは困難です。重さという単位は、鉄、砂、食物、飲料水などあらゆるモノの重さの比較に使えます。

ソフトウェアの世界に客観的指標がないのは、単位が存在しないということが理由ではないかと思っています。

単位がなく重量やサイズの測定ができないということは、設計・開発という作業に次のような困難さを生じさせています。

### ① 適切な機能の大きさと範囲の評価が困難

② インターフェースの適切さと品質確認が困難

③ 要件があいまいになりやすい（重量やサイズといった客観的指標で要件が規定できない）

④ 生産性の基準と数値が恣意的になりやすい（投入工数もあてにならないところで、客観的な産出量をどう測定するのか?）

⑤ 見積もりが困難

複雑な条件判断が必要な大きな機能を、どのような重さにすれば製造に耐えられるのか、ということを数値で客観的に議論できるようになればどんなに便利でしょう。開発規模もステップ数というあいまいなものではなく、重さで測れるようになれば、生産性の数値も意味のあるものになるでしょう。見積もりも、その数値を基準にできるようになる可能性があります。

“ソフトウェアの重量”という考え方は存在していないので、その定義があるわけはありませんが、そのような概念を導出し分野の異なるソフトウェアを比較できる指標を策定することは有効だと考えます。その中には、構造化定理が示す三つの基準（順序、条件分岐、繰り返し）という要素、および、複雑度や規模（たとえば、扱うデータ項目数や実行行数など）も含めた指標であることが望ましいと考えています。また、特定のデータに対する命令文の出現のバラツキの程度は、ソフトウェアの解読のしやすさに影響するので、重さの指標には小説の“文体”にも相当する“データに対する処理の凝集度”といった考え方も必要かも知れません。

複雑度に関しては、MaCabeの循環的複雑度というのがあります。比較的知られていますが、プログラムの独立した経路の数で示されます。この値は、プログラムに条件文がなければ、行数にかかわらず“1”ということになり、複合条件が指定されていても、経路は一つと評価され、直観的に考えるプログラムの複雑度からすると、物足りない印象を受けます。

もう一点、ソフトウェアの重量という考え方には、その対象（たとえば業務そのもの）の重量の違いということも反映されるのが望ましいといえます。製品としてのソフトウェアの場合はともかく、ビジネス・アプリケーションの開発では、見積もり時点ではソフトウェアは存在しないのですから、見積もりのインプットとしては業務そのものの重量を評価し、それに基づいてソフトウェアの重量や工数を見積もれるようにすることが必要でしょう。情報システムを使う、使わないにかかわらず、業務処理そのものも“情報処理”だと考えれば、そこには“情報量”という概念が存

在してもおかしくはありません。それを数値化し、業務の重量を測定し見積もりに利用するようなことができれば、見積もりの確からしさが評価できるようになるのではないかと考えています。

ソフトウェアに単位がないということが話題になることは今までになかったのかもしれませんが、今後研究する価値のあるテーマではないかと思っています。

今回はここまでとします。次回は、こういった特徴を備えたソフトウェアをどのように開発すべきかという観点から開発工程モデル（ウォーターフォールとアジャイルなど）について考察してみる予定です。

以上