SF 乱学講座 2006 年 3 月
コンピュータソフト/システムのできるまで。
もしくは「死の行軍」への道。

長 高弘

2006 年 3 月 5 日

1 タイトルの意味 − “死の行軍” とは何ぞや？ −

我々の業界、つまりコンピュータ業界の方や、コンピュータ業界に知合いがいらっしゃる方なら御存じとは思いますが、我々の業界では、仕事がやたらと忙しい人間や、仕事でトラブっている人間が山程居ます。

そして、昨今、良くコンピュータシステムのトラブルが報道されています。

つまり、コンピュータ業界は、トラブルが起こっているプロジェクトが、たくさん有る訳です。コンピュータ業界では、トラブルで、にっちも、さっちも行かなくなった状態の事を**「デスマーチ」**と呼びます。(元々は「Death March: The Complete Software Developers Guide to Surviving “Mission Impossible” Projects」と言う本で、混乱したソフト開発現場を “death march” と呼んだ事から)

この「デスマーチ」と言う言葉で、思い浮かぶものと言えば、**“Bataan Death March” (バターン 死の行軍)** でしょう。

何故か、国内の IT 関連の用語解説サイトでは、「“Death March” とは軍隊の苛酷な訓練の事」と言う説明がされている事が多いのですが、**「フルメタルジャケット」と「バターン 死の行軍」信は別物でしょう、どう考えたって。**

では、なぜ、コンピュータ業界では、「バターン 死の行軍」に例えられるような事が起きるのかを中心に、コンピュータソフトが出来るまでを解説します。

2 コンピュータソフト/システムとは何か

では、まず、基礎知識から。

当り前の事ですが、そもそも、コンピュータソフトウェアや、コンピュータを使ったシステムは、何らかの目的に基づいて作られ、導入されます。

例えば·····

- 会社の経理処理を行いたい。
- 何らかの装置の制御を行いたい。
- 工場での生産管理を行いたい。
- 顧客との間でオンライン取り引きを行いたい。

これらの、「ソフトやシステムを作成・導入する、そもそも目的」の事を、コンピュータ屋の符丁では「機能要件」と呼びます。

また、コンピュータソフトウェアや、コンピュータシステムを導入するにあたっては、様々な条件が有ります。

例えば·····

- 予算。
- 開発期間。

そして、本来の導入目的に付随する条件が有ります。

例えば·····

- ある処理を行う際の処理時間。
- あるデータの最大件数。
- 拡張性。
  - コンピュータの台数を、容易に増やせる。
  - プログラムの改修が容易。
- 障害耐性。
  - あるコンピュータが故障しても、他のコンピュータが自動的に処理を引き継げる。
  - 万が一、想定外の異常が起きても、原因を特定するための情報が、どこかに保存されている。

これらの事を、コンピュータ屋の符丁では「非機能要件」と呼びます。

私のような、コンピュータソフト/システムに関わる人間に求められる事は、必要な機能要件および非機能要件を満たすソフトウェアやシステムを、一定の予算・期間で作り上

げる事です。

ただし、これは非常に単純化した言い方です。例えば、複数の要件が有る場合、優先度が有りますし、ヤバい状態になった時には、品質と納期のどちらを優先するか、と言う選択をしなければいけない事もあります。

以降では、機能要件が比較的明確な「ある顧客から注文を受けて、その顧客が必要としているソフト/システムを作る」場合を例に、ソフト/システム出来るまでの解説をします。

3 開発の為の金と人

コンピュータ・ソフト/システムの開発が始まり、開発が完了し、寿命が来る (または改修の必要が生じる) まで、おおまかには、以下のようなプロセスを辿ります。

1. 要件の確認 (どう言うモノを作りたいかの詳細を確認する)
2. 仕様の決定 (作るプログラム/システムの詳細を決める)
3. コーディング/システム構築
4. テスト
5. 納品
6. 保守
7. システムの新規開発・改修 (1 へ戻る)

無論、このプロセス自体を、大きく変えるのは難しい (最近は、そうとも言いきれない、という考え方も出てきていますが) のですが、このプロセスの中にも、後々、色々な問題を発生させる要因が有ります。

3.1 外注さん いらっしゃい

前述のプロセスの中で、多量の人員が必要になるのは、上記の 2 から、4 ぐらいまで、1 や 6 では、それ程の人員は必要になりません。

つまり、コンピュータ・ソフト/システムの開発・保守を商売にする場合、現在、どのプロセスにあるかによって、適切な人員がどれ位かが、全然、違ってきます。

そして、かなり、経営に余力が有る会社でも、一番忙しい時に必要な人員を、**常時、養っていくのは、極めて困難です**。(と、言うか、そんな真似をしない会社が、この業界では、「経営に余力が有る会社」の必要条件の 1 つだったりします)

また、場合によっては、自分の会社に、必要な技術を持った人間が居ないような場合も、有り得ます。

そのような場合に必要となるのが、**外注**です。大きな規模のシステム/ソフトになると、2 次外注、3 次外注は当り前になります。

さらに、大きい会社ほど、社員の給与・福利厚生等が良く、採算に直接関係しない部門 (総務・経理・教育・管理) が大きくなる傾向が有ります (当然、この手の部門の人間の給与は、採算に直接関係する部門が稼いだ金から出てる)。言い方を変えると、大きな会社程、1 人あたりの人件費が高くなり、開発メンバーを、全て自社の人間にすると、採算が取れなくなってしまう傾向が有るのです。

こうして、この業界では、外注依存が、どんどん強まってきます。さらに、国内のみならず、インド (実は、以前から、インドのソフトウェア産業は優秀であると言われていた)、中国 (最近は、日本との取り引きが多い、沿岸部の企業のみならず、内陸部の企業とも取り引きが始まっている)、韓国、ベトナム (値段はインド、中国、韓国に比べるとお手軽だが、能力も、それ相応らしい) などの企業に外注するケースも多々有ります。

結果として、外注先が国内・海外を問わず、

- 外注との意志疎通が、ちゃんと出来ていない。(とんちんかんなモノが出来てくる)
- 本番で障害が発生した時、プログラムを作った人間と連絡が取れない。
- 特殊なプログラムの場合、外注先にテスト環境が無い。

などの問題が発生しがちです。

これらの問題を防ぐため、例えば、大手 A 社が、別の会社 B 社,C 社,D 社に、外注を行った場合、B〜D 社の人が、A 社内で作業をする、と言う事が有ります。この結果、A 社内の B〜D 社に外注をした部門のフロアには、A 社の人間は、ほとんどおらず、B〜D 社 (もしくは、さらに、その外注) の人ばかり、と言う事態が発生します。

3.2 世の中は金と時間が敵 (かたき) なり

例えば、家を建てる時、階数、部屋数、広さ、その他条件を、大工さんに言った際、
「建て終わる時期も、費用も、全く見当が付きません」
と言う答えが帰ってくれば、多分、その大工さんには、お帰り頂く事になるでしょう。

コンピュータソフト/システムでも同じ事です。

納期や、顧客が支払う金額は、かなり早い段階で決まる事が普通です。しかし、この時点は、設計の詳細も、開発にあたっての懸案 (何が決まっていて、何が決まっていないか？

発生が予想される問題は？) も、**ほとんど見えていません**。

しかも、何の因果か、コンピュータ業界では、技術というモノは、なかなか「枯れて」くれません。

似たようなソフト/システムを作った時に、必要であった時間・人員・予算を参考に出来る場合もあります。しかし、それも、数年で様々な前提が変わり、アテにならない代物に成り下がります。

数年前に発生した問題は、今回は発生しないかもしれません。その替わり、予想もしなかった、別の問題が起こるかもしれません。

コンピュータソフト/システムを作る際には、必要な予算・人員・期間の見積りが非常に難しいのが普通です。

そして、往々にして、予算は少なく (予算 ≒ 人件費 なので、人員も少なくなる)、開発期間は短めに見積もられます。

こうして、少なからぬプロジェクトが、始まった早々に、破滅への第一歩を踏み出すのです。

4 Road to Perdition

と言う訳で、プロジェクトは、開始早々、拝一刀・大五郎父子よろしく、冥府魔道へ迷い込みました。「子連れ狼」にも出てきた仏教の二河白道の喩ではありませんが、プロジェクトが、「予算・納期超過」と「納品物が低品質」のいずれにも陥る事なく、見事成功するか、御期待下さい。

では、各工程で行われる事を説明します。

4.1 要件の確認 −そもそも、モノを作るとは−

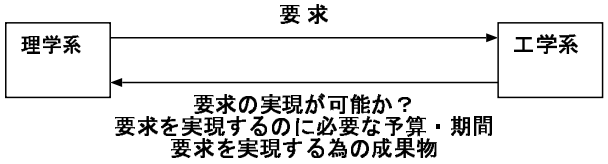
まず、本題に入る前に、我々の業界で、どうやってモノを作っているか、と言う事を説明するための、良質な参考資料を紹介します。

ただ、その「参考資料」は、技術関係の書籍では有りますが、**コンピュータ・IT 関連の書籍ではありません**。

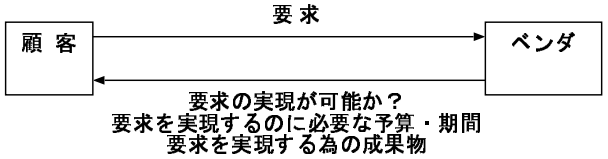
それは、日本の火星探査機「のぞみ」の開発から「失敗」に至るまでの経緯を書いた、松浦晋也さんの「恐るべき旅路」(朝日ソノラマ) です。

まず、この書籍の中で、「のぞみ」の開発は、理学系の研究者 (どう言う観測データが必要かを決める人達) と、工学系の研究者 (理学系の研究者の要求を実現する為の装置等の設計を行う人達) が協力して行った、と言う事が書かれています。

言わば、「のぞみ」の開発に関わった人達の関係を図にすると、次の様になります。



このような関係は、顧客と、ソフト/システムを開発する会社 (以下、「ベンダ」と呼びます) の間にも成り立ちます。



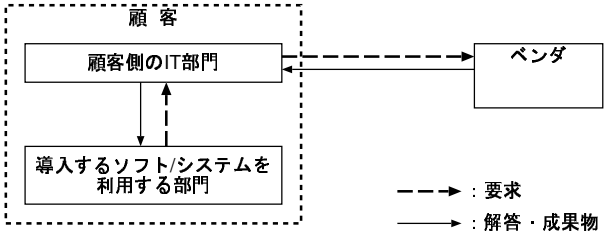
つまり、ベンダ側は、どう言うソフト/システムを必要としているかを、顧客自体が良く判っていなければ、作業のしようが有りません。

もっとも、顧客側の漠然とした要求を形にするのもベンダ側の役割の一つとも言えます。しかし、顧客側の漠然とした要求を形にする場合、どう言う「形」にするかは、顧客側に、どのような問題・必要性・思惑が有って、ソフト/システムを導入しようとしているかによって決めなければいけません。

また、顧客の会社や業界における「常識」を、ベンダ側が良く知らない事も多々有ります。

つまり、顧客の要求を明確化する事は、ある意味で、顧客とベンダの共同作業なのです。

しかし、顧客が規模の大きい会社だった場合、話が血塗られてきます。つまり、以下のような事が起き、ベンダと直接やりとりをする顧客側の担当者も、ソフト/システムを導入するに至った問題・必要性・思惑や、実際に使う人達が「常識」としている事をイマイチ理解していない場合が有るのです。



さらに、大規模なシステムの場合は、システムの一部分のみを、顧客側のある部門 A が利用し、システムの別の部分、別の部門 B が利用し‥‥と言う事も有ります。

そうなってくると、顧客とベンダの窓口も複数になってしまう事もあり、事態は、さらに複雑化 (言葉を変えれば、後々の火種になる要因) します。

4.2 仕様の決定 －最も頭を使う所－

顧客の要求は、不安要因は有りますが、一応、形には成りました。

次に、仕様の作成、または、設計と呼ばれている作業を行いますいわば、顧客の要求をプログラムにするために必要な設計図の作成です。これらの「設計図」は、「仕様書」や「設計書」と呼ばれるのが一般的です。

これらの設計書や仕様書も、顧客の要求を文書化したもの、基本方針、プログラムに必要な機能を記述した物、その機能を実現する為のプログラムの詳細、と何段階かに分けて作るのが理想でしょう。

何故ならば、顧客の要求だか、プログラムの詳細だか判らないようなドキュメントを元にプログラムを作成すれば、結局、そのプログラムは、顧客の要求を、より良く満たすものにならない可能性が出てきます。

また、プログラムで不良が発生した場合、どの段階で、不良の原因が紛れ込んだか、不良をどう直すのが正解か、と言う事が判りにくくなります。

しかし、問題は、顧客によって、納品を要求される設計書・仕様書の種類やフォーマットが異なっている事です。そして、往々にして、タイトな人員・期間で作業をしているベンダの作業者は、顧客に要求された以外のドキュメントを作ろうとはしない事が多くなります。

そのようなケースの中でも、顧客に納品する設計書・仕様書の体系やフォーマットに問題が有ると、悲惨な結果になる事が有ります。

まず、不良が発生した場合、不良をどう直すのが正解なのかが判りにくくなります。(「正解」となる直し方の必要条件の一つは、顧客の要件を満たす事ですが、その「要件」がドキュメントに残されていなかったり、ドキュメントに書かれている内容の、どこから、どこまでが顧客の要求なのかが判りにくければ、「正解」に辿り着くのは難しいでしょう)

また、ドキュメントを見ただけでは、プログラムの詳細や、システムの全体像が良く理解できない可能性が出てきます。つまり、ドキュメントを通した、作業者内での知識の共有が出来ないのです。(一体、何の為のドキュメントなのか、判らなくなってくる事も多々有ります)

この工程こそが、ソフト/システムを作り上げる場合の、一番、重要な部分と言えますし、作業も大変です。ですが、往々にして、出来てくる成果物は、この工程の重要さや、この工程に投入された労力に見合ったものかは、別問題と言う場合も少なからず有ります。

4.3 コーディング/システム構築

コーディングとは、プログラムを書くこと。実は、ここは、案外簡単です。人間の言葉で書かれた仕様書や設計書を、プログラミング言語に翻訳するだけです。

例外 (技術的に難しい点がある、など) は有りますが、案外、単調です。

出来上がったモノの品質も、単純ミスなどを除けば、仕様書や設計書の品質や、前段階での作業の質と、と正の相関が有ると考えて良いと思います。無論、プログラマによって、品質に差は出ます。しかし、システム全体の設計がタコだと、そのシステムを作る為の部品である各プログラムの質がいくら良くても、限界が有ります。

問題は、プログラムを動かす為のコンピュータを設定するシステム構築です。

多数のマシンからなる大規模なシステムの場合、コンピュータの設定をするだけでも、結構な作業量になります。また、マシン間での通信が上手くいかない (ネットワーク系のトラブル)、とか、ここに来て、OS その他の設定が、当初、想定していたものでは、まずい点が有る事が判明するなどの問題が起こります。

また、システムの規模が大きくなると、プログラマ部隊とは、別の部隊によって、この作業が行われます。

言葉を変えれば、プログラムを作った部隊と、システム構築をやる部隊の間で、意志疎通や情報の共有が不十分なケースが有ると言う事です。

しかも、大規模なシステムの場合、プログラマ部隊も何チームかに分かれています。そして、意志疎通や情報の共有が不十分と言う問題は、各プログラマ部隊間でも起こり得ます。

この結果、起こる事は (次に説明するテスト段階にも関係しますが)、プログラマが使っていたマシンでは、ちゃんと動いていた筈のプログラムが、システム構築部隊が本番に近い設定をしたマシンの上では、ちゃんと動かなかったり、他の部隊が作ったプログラムとの連携が上手くいかなかったり、と間抜けな事が起こります。

4.4 1 にテスト 2 にテスト 3,4 が無くて 5 にテスト

そして、最終工程のテストです。と、言っても、テストの一部は、システム構築より前に行われます。

まず、テストと一口に言っても、次のようなものが有ります。

1. 各プログラムや、そのプログラムで使用する「部品」毎のテスト (単体テスト)

2. 関連する部品やプログラムを組み合わせた際のテスト (組合せテスト)
3. 本番に近い状態でのテスト

さらには、テスト専門の部隊が開発者とは別の観点からテストを行ったり、顧客に実際に使ってもらうようなテストも有ります。

顧客が実際に使って、「何か、思ってたのと違うぞ」と言う話になり、最初からやりなおしなどと言う悲惨なケースも有ります。

また、何の為にテストを行うかについても、複数の考え方が有ります。

1. プログラマだって、所詮は人間。完璧なものが作れる筈が無い。テストをちゃんと行う事で、バグを摘出しなければいけない。
2. テスト段階、それも、後の方になってバグが見付かれば、出戻り作業が発生する。重篤なバグを作りこみにくい方針を立てて設計や開発を行い、テストはあくまでも、品質の確認とすべき。
3. コーディングの一部としてのテスト。(最近、広まっている「テストファースト」と言う考え方)

上記の１の方針で開発・テストを行っていた場合、テスト期間は、十分長く取る必要が有ります。しかし、納期に余裕が無いような場合は、テスト期間が短くなり、十分なテストが出来ない場合が有ります。その様な場合、何が起きるかは、容易に予想が付くと思います。

4.5 ふりだしに戻る

数々の困難を乗り越え、開発とテストが終了し、顧客にソフト/システムを納めました。しかし、これで終わりでは有りません。

テスト時に見付からなかった、バグが本番で見付かり、プログラムを修正する必要が出てくるかもしれません。

顧客が、納品されたソフト/システムを使っている内に、こう言う機能も欲しいと思うかもしれません。

顧客の要求の前提事項が替わり、別の要求が出てくるかもしれません。

一度、作ったソフト/システムは、遅かれ早かれ、修正する必要が出てきます。

ここで、話は、要件の決定や、仕様やコーディングに戻ります。要件の確認の段階で注意すべき事は、顧客の要件の内、将来的に変わる可能性があるものが、どれかを見極める事であり、仕様やコーディングの段階で注意すべき事は、変更しやすいプログラムを作る事です。そうしないと、後で泣きを見る事になります。

5 万能薬を信じるのは愚か者だけ

では、結論として、なぜ、コンピュータ業界ではデスマーチが多いのか、と言う事をまとめます。

1. ソフト/システムを作る際には、複数の企業・部署の人間が関係する場合が多い。(関係者間の意志疎通や情報の共有がうまくいかない上に、意志決定もおくれる)
2. ソフト/システムを作るために必要な費用・人員の見積りが困難。(人員・期間が最初から不足した状態で、開発を行う羽目になる)
3. コンピュータ技術自体が、ある意味で、まだ、発展途上の技術である。(せっかく学んだ技術も、数年後には時代後れになってしまう)

でも、はっきり言って、理由は判っても、対策の立てようが無いようなものばかりです。それに、コンピュータ業界で、良く言われている言葉に、こう言うものが有ります。

「狼男を殺せる銀の弾丸は無い」

6 今後のこの業界

ついでとして、この業界の、最近の動向について述べます。

私が説明したのは、開発工程を、いくつかに分け、各工程でドキュメントをきちんと残す事を理想とする方法です。

しかし、これに対して、最近は「アジャイル」と呼ばれる、ドキュメントよりも、動くプログラムを作る事を最優先とする考え方が出てきました。

この考え方については、個人的には、見るべき点も有り面白いと考えている半面、「銀の弾丸」には成り得ないとも思っています。これまでの開発方法が、大規模システム向け立ったのに対し、「アジャイル」は、小規模だが、変更が多いシステムを少ない費用で手早く作ったり改修したりするのに向いています。

また、テストを、仕様の決定やコーディングの後工程としてだけではなく、コーディングの一部としてテストを採り入れる (まず、テスト用のプログラムを作り、そのテストをパスするようなプログラムを作る)、テストファーストと言う考え方が出てきました。このテストファーストを採用する事で、「動いているプログラム下手に改修しない」と言う、今までの常識に真向から反する、「プログラムの外面的な動

作が同じなら、どんどん、改修しても良い」と言う開発方針が採用される事も出てきます。

　こう言った、最近の新しい考え方が、本当に有効なのかは、もう少し様子を見る必要が有るように思います。