レガシーシステム再構築に関する考察

修正　平成 30 年 6 月 27 日
修正　平成 29 年 8 月 28 日
修正　平成 29 年 7 月 6 日
平成 29 年 6 月 1 日

技術士（情報工学部門）
ジェネクサス・ジャパン株式会社
代表取締役社長　大脇 文雄

＜内　容＞

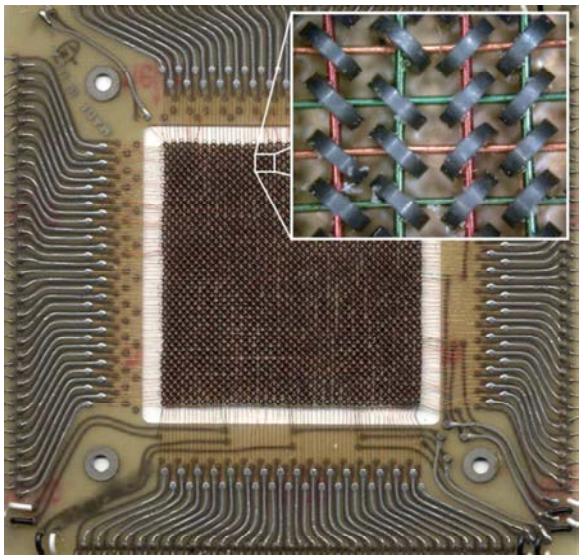
0．はじめに

この考察はレガシーシステム再構築という非常にデリケートな話題に挑戦したもので、以下の構成で整理をしてみました。なお、この考察を行うに際し、実際に幾つものレガシーシステム再構築を成功させているパートナー様の率直な御意見も参考にさせていただきました。厚くお礼申し上げます。

1．考察を始めるに当たって
：自分の経験から、この話題に目覚めていった経緯を述べます。
2．レガシーシステムとは？
：レガシーシステムとは何なのか。一部のシステムの問題なのか、それとも社会問題の一つなのか。
3．レガシーシステム再構築の何が問題なのか？
：システムの専門家でない方々には、レガシーシステム再構築の問題が理解できません。しかし、レガシーシステム再構築はシステム部門だけの話題ではありません。
4．レガシーシステム再構築方法の種類
：現在行われている様々なレガシーシステム再構築の方法の種類と問題点を考えます。
5．レガシーシステム再構築は可能なのか？
：レガシーシステム再構築の失敗事例を多く目にするとき、絶望的な心境に陥ります。しかし、可能な方法は有る筈です。
6．レガシーシステム再構築に利用できる情報
：現状でレガシーシステム再構築に利用できる情報を考えてみます。
7．レガシーシステム再構築のやり方
：GeneXus の世界には、レガシーシステム再構築の事例が数多く存在します。その手順を分かり易く説明します。
8．ツールが持つべきレガシーシステム再構築に必要な機能
：GeneXus 以外のツールを用いて、レガシーシステム再構築に挑戦する場合、ツールが持つべき機能を考えます。
9．GeneXus でレガシーシステム再構築が可能なのか？
：GeneXus を使えば、なぜレガシーシステム再構築が可能となるのか、そのポイントを洗い出します。

1．考察を始めるに当たって

数十年前に入社した頃、システム管理部には数十人の先輩技術者が働いておられました。丸の内のビルの５階に、特別に空調を効かせたマシンルームがあり、巨大なホストコンピュータ（IBM370-M155　と　M165）が稼働していました。マシンルームの奥の CE ルームには、IBM の CE（Customer Engineer)が常時待機されており、作業の合間に良く雑談をしました。ある日、CE ルームの中に磁気コアメモリーのボードが置かれていました。話には聞いていたのですが、実物を見るのは初めてでした。縦横 50　cm くらいのボードで、それで容量は 1 KB 程度だったそうです。



(参照：https://commons.wikimedia.org/wiki/File:Ferrite_core_memory.jpg)

配属されると、担当していたＩＢＭの技術者の方が我々新人のために BAL(Basic Assembler)のプログラミング教育を行ってくれました。この教育でプログラミングを覚えましたが、とても面白かったですね。新人教育の終了後に配属されたところが、情報系のシステム開発をする課でした。そこでは情報系ということでＰＬ／１を使っていました。

ＰＬ／１の Language Reference Manual や Execution Logic を渡され、自分で勉強するように言われました。夢中になって読みましたね。内部のメモリー構造や実行の仕方を理解できると、プログラミングの面白さが倍増しました。

そして、数年後に五反田のシステム子会社に移ると、二次オンラインの開発が始まり、プログラマーとして先輩たちの設計書を基に、BAL でプログラムを書きました。当時はプログラムを書くのが楽しくて、レガシーシステムの問題なんて考えもしませんでした。

バブルの時代が近づいてくると、本社からシステム化の要求が強くなってきました。初めて外部業者を使ってシステム開発をしたのもこの頃です。ある役員は、「全員がプロマネになって、外部業者を使ってたくさんのシステムを作ればいいのだ」とおっしゃいました。システム開発の素人がシステム子会社の役員であることに、すごく違和感を抱いたことを覚えています。その方は、システムの本質的な問題を全く理解していなかったのです。

オンライン端末の老朽化に伴い、この三次オンラインの改修プロジェクトが始まった頃から、レガシーシステムの問題を意識し始めたように思います。そして、パソコンの時代が始まり、ＩＴの技術革新の話題が出てくるようになりますと、レガシーシステム再構築の話題が広がってきました。

2003 年に会社を設立して以降も、レガシーシステム再構築の話題を忘れたわけではありません。GeneXus を有効に活用できれば、レガシーシステム再構築に光明が差すように感じていたのです。公開はしませんでしたが、レガシーシステム再構築に関するアイデアを何回も書き続けました。

そして今回、これまでの考えをまとめて、公開することにしました。

２．レガシーシステムとは？

(1).最初は誰も認識しなかった

日本に初めて商用コンピュータが入ってきたのは 1955 年でした。私が勤務していた会社がコンピュータを導入したのは 1956 年だったそうです。当初は、人海作戦だった売買報告書の計算処理等を、システムを導入すれば自動化できる、といった合理化の効果が経営者の説得材料として使われたそうです。多分、当時開発されたシステムが長年にわたって利用され、レガシーの問題を引き起こすなんて、誰も思わなかったのでしょう。

(2).設計書は手抜きをせずに作成し管理していた

当時のシステム開発は内製化が中心で、開発メンバーは業務部門の責任者の方たちと相談しながら設計を行いました。ワープロなどはまだなく、手書きで設計書を書いていました。字の上手下手はありますが、手抜きはしなかったと思います。

ですから、誰もが**「正しい設計書さえ残していれば、後日、機能変更や機能追加が発生しても、問題なく対応できる」**と考えていました。

(3).情報処理技術の進化のため、設計書が使い物にならなくなった

ところが、パソコンが世の中に広まってきますと、情報処理技術が急速に進化し始めました。ホストコンピュータの時代からパソコンの時代に移っていったのです。マイクロソフトが Windows95 を発表したときには、世界中が熱狂しました。しかし、その後 Windows 自体が、Windows98、XP、Vista、7、8、10 と進化し、古い OS は保守切れとなり、ウィルスの温床となってしまいました。

パワーユーザたちは WindowsXP 上で、Excel と VBA 等を使い、多くの業務を改善するためのツールを作ってきました。しかし、Excel と VBA で書かれた多くのツールは、新しい Windows の OS 上では動きませんでした。

情報処理の技術者たちが一生懸命努力をして作成したシステムも、似たような状態に陥っています。情報処理技術が変化し、実現方法も変わったために、せっかく書いた設計書が陳腐化し、使い物にならなくなってしまったのです。

(4).ハードの世界とソフトの世界の違い？

「ハードの世界でも似たような話はあるのかな」とも考えましたが、そんな話は聞いたことがありません。レガシーの問題はソフトの世界だけのことなのでしょうか。

ハードは目で見て触ることができますから、多分感覚的に理解ができるのでしょう。しかし、ソフトは目で見えませんし、あるのは各コンピュータ言語で書かれたソースコードだけですから、ソースコードが実現している動きを感覚的に理解するのは難しいのでしょう。ソースコードを読んで、アルゴリズムを読み解き、実現している業務を理解することができなければ、分かったとは言えません。もしかすると、ソフトの世界の方が複雑なのかもしれません。

(5).日本のＩＴ技術は世界で３０位以下？

既に述べた様に、日本は世界でも早期にコンピュータを研究し実用化してきた国の一つです。例えば、富士通の沼津工場内の池田記念室を見学すると、池田敏雄氏という天才の豊かな発想と独創性を目にすることができます。

ですから、私が JICA の派遣専門家として活動しているとき、無意識に「指導してやる」という気持ちになっていたかもしれません。

ところが、先日のニュースを見て愕然としました。世界的なランキングでは毎年順位が下がり、３０位くらいに落ちているのですね。日本の企業は最新のＩＴ技術を採用せず、ＩＴ投資に対して消極的である、という批判を聞いたことがあります。

しかし、ＩＴ投資に消極的になっている一般企業の経営者を責める前に、その理由を理解すべきではないでしょうか。

日本でコンピュータを利用していない企業はほとんどありません。家内工業の様な小規模の企業でも、会計パッケージは利用しているでしょう。でも、これが日本のＩＴが遅れた最大の原因だと思います。つまり、全ての企業が何らかの形でレガシーシステムの問題に直面し、萎縮してしまっているのです。

業務システムを長年にわたって利用すると、再構築するのが極めて困難な状況に陥っていることに気付きます。しかも、毎年レガシーシステムの保守に莫大な支出を強要されています。この様な状況下で、最新のＩＴ技術を採用する、という決断を経営者は下せるでしょうか。

レガシーシステム再構築の問題を明確にし、経営者の不安を取り除かなければ、現状を打開することは無理ではないでしょうか。このままで、最新のＩＴ技術に投資する様に日本企業を説得しても、かなり難しいでしょう。

3．レガシーシステム再構築の何が問題なのか？

「**レガシーシステム再構築は新規のシステム開発よりもはるかに難しい**」とよく言われます。しかし、経営者やシステムを利用している担当者の感覚とは大きく違っています。その理由を考えてみます。

(1).経営者の理解を得られない

これは理解できます。経営者の立場では、「機能は変わらないのに、なぜ作り直す必要があるのか」という単純な疑問が出てくるのは当然です。「機能は変わりませんが、技術の基盤が変わったので作り替える必要があるのです」と言ったって納得してもらえません。「壊れるまで、そのまま使い続けろ」と言われるに決まっています。会社の収益向上に寄与しないのに、経営者を納得させるのは至難の業です。割に合わない仕事ですね。

(2).業務担当者の理解も得られない

これも理解できます。「機能がほとんど変わらないのに、なぜ作り直す必要があるのか。何もメリットがないではないか」と言われます。

しかも、「要件定義書を作成するのに協力してくれ」なんてお願いすると、「ふざけるな！」って怒られてしまいます。だって、「チャンと動いているシステムがあるのだから、業務の仕様は分かっているだろう」と言われます。

それでも協力してもらえれば幸運ですが、協力してくれる担当者は日々の業務に利用する前提で、使い勝手、画面レイアウト、振る舞い、機能の充実度等をチェックし、レガシーシステムとの違いを指摘し、同じになるように修正することを求めてきます。また、業務担当者は業務改善のチャンスと考え、機能の改善を期待しています。しかし、仕様は不明確なままであり、開発の攪乱要因ともなります。

結局、「**動くものができたらチェックしてやるよ**」と言われて、協力してもらえなくなるかもしれません。

(3).要件定義書を書くための業務知識の喪失

レガシー・システム再構築は、「**要求仕様が不明確なまま開始する業務システム開発**」と同じ、と感じています。実は、長年システムを利用している業務部門から、業務システムを開発するのに必要な、要件定義書を書くための業務知識が失われているのではありませんか。

例えば、業務担当者に「この帳票の作成方法を教えてください」と聞くと、「この画面のこのボタンを押すと印刷されます」といった返事が返ってきます。「いえ、そうではなくて、この数値の加工条件や計算方法を教えてください」と再度聞くと、「それは保守しているシステム部門に聞いてください」という返事が返ってきます。

そして、不明確な要件定義書に基づいて業務システムを開発しても、業務担当者がシステムテストに参加した段階で「こんなシステムは使えない。元のシステムに戻してくれ」、「元のシステムと寸分違わないように修正しろ」、「元のシステムというお手本が有るのだから簡単だろ」といった批判にさらされます。

もはや、業務部門から業務要件を抽出しようとする要件分析手法は時代遅れなのです。業務全体を理解して、要件定義書を書けるようなスーパーマンは存在しないのです。

(4).システム開発時の業務担当者とシステム技術者が残っていない

しかも、長年にわたって業務システムを運用しているうちに、法律改正、制度変更、組織変更、新規業務の追加、社外との情報交換、人事異動等が発生します。業務システムには、これらの変化に対応するために、様々な変更や追加が加えられてきました。

ところが、このような変更や追加を行うのは、ほとんどの場合システムを開発した技術者ではありません。年数が経過するに従って、変更や追加といった保守作業は後輩の技術者たちに引き継がれていきます。そして、開発を行った技術者たちは他の開発を担当したり、他部門に異動したり、更に年数が経過すると転職や退職してしまっているかもしれません。しかも、保守をする後輩の技術者たちに十分な引継ぎ時間は与えられませんでした。

結果として、要件定義書や設計書の変更がおろそかになり、「ソースコード以外は信用できない」という状況に陥ってしまいました。

(5).業務仕様はソースコード上にしか残っていない、ソースコードも存在しない?

この「ソースコードしか信用できない」という状況は、多くの会社が経験していることです。特に、バブルの時代を経て長年使われてきたレガシーシステムの状況は悲惨でしょう。まれなケースでしょうが、「ソースコードも存在しないのです」と言われたこともあります。

しかし、忘れていただきたくないことは、たとえソースコードが残っていても、それらのソースコードは**「システムが構築されたときの情報処理技術に基づいて書かれている」**ということです。つまり、これらのソースコードの実装の記述は再利用できない、ということです。

(6).レガシーシステム再構築は、大量の仕様変更と仕様追加との格闘

長年利用されてきた元のシステムには、本番運用開始後に多くの異例処理の追加、仕様変更、機能追加がされており、しかも、設計書の修正や追加が追い付きませんでした。

ですから、レガシーシステム再構築が終盤に差しかかり、業務部門の担当者がテストを始めると、これらの異例処理、仕様変更、機能追加の要求が五月雨式に発生してきます。

つまり、レガシーシステム再構築では、**「事前に完全な要件定義書を作成することは不可能に近く、膨大な数の仕様変更や機能追加要求が発生する」**という前提を受け入れなければなりません。極端な場合、業務部門の担当者から「こんなシステムは使えない」と受入れを拒否されるケースもあるのです。ですから、ウォーターフォール型の開発手法で挑戦した多くのレガシーシステム再構築が失敗しているのです。

(7).情報処理技術の進化

情報処理技術は急速に進化しています。パソコンが主流となり、インターネットの利用が拡大し、スマートディバイスの利用も一般的になってきました。レガシーシステム再構築で事前に考えておくべき点の一つが、この情報処理技術の進化への対応です。

例えば、今回システムの再構築を決断したとして、開発するのに１～２年必要で、しかも、システムが安定稼働をするまで２～３年かかるとします。

そして、システムが安定稼働する段階になると、ベンダーから「ハードの保守期限が来ますので、ハードを入れ替えてください」という話がくるかもしれません。ハードを入れ替える決断をすると、次は OS をバージョンアップするように言われるでしょう。さらに、アプリケーションサーバやフレームワークもバージョンアップや入替えが必要となります。そして最終的に、アプリケーションの再構築が必要となるかもしれません。「まだ使えるのに、まだ償却が終わっていないのに」とぼやいても後の祭りです。

同じシステムを何回作り直せば気が済むのか？経営者は呆れているでしょう。情報処理部門の評価は最低になっているかもしれません。

（参照：三上氏講演資料）

4．レガシー・システム再構築方法の種類

これまで採用されてきたレガシー・システム再構築方法には、以下のような種類があります。

(1).全面再構築方式

先ず、誰でも思い浮かぶのは、全面的な再構築かもしれません。しかし、成功するケースは少ないでしょう。

要件分析から始めて新規にシステムを開発しようとしても、システム開発に必要な要件定義書を書くための業務知識が、業務部門の担当者から失われています。ですから、システム開発の教科書通りに、業務担当者とのヒアリングから要件をまとめようとしても無理なのです。

そこで、失われた業務知識をカバーするために、「ERP パッケージを導入し、自社の業務に合致するようにカスタマイズする」という再構築方法が魅力的に見えてきます。

(2).ERP パッケージのカスタマイズ方式

ERP パッケージというのは、或る業務モデルを実現するために開発された業務システムであり、そのまま利用できるならば非常に有効です。

昔、或る外国の事業会社の CIO と話したことがあるのですが、ERP パッケージ導入に合わせて社内の組織を全面的に変更したそうです。そして、ERP パッケージ導入によって余分になった部門は、Redundancy（余剰）として全員解雇し

てしまったそうですが、日本の会社でこんな決断が下せるでしょうか。

ですから、ERP パッケージを導入しようとする日本の会社は、先ず、ERP パッケージと現実の業務との Fit & Gap を行い、現在の業務に合わせて、違っている箇所をカスタマイズしようとします。

しかし、ERP パッケージのカスタマイズは簡単ではありません。Fit & Gap を行ったとしても、現在の業務が完全に分析できている訳ではありません。業務の担当者が参加するシステムテストが始まると、必ず、膨大な数の仕様変更が発生し、最悪の場合「使い物にならない」と受入れを拒否されてしまう場合もあります。

日経コンピュータ 2016.10.13号に掲載されていた記事のように、「 ERP パッケージの導入失敗による訴訟が全体の６～７割に及び、ここ５～６年間で高止まりしている」そうです。

いずれにしても、要件定義の失敗が多くの訴訟の原因になっています。

ユーザー企業とITベンダーの間で、システム開発訴訟が続発

表1 2009年以降に提起され、請求額が1億円を超えるIT訴訟の例

提訴日	当事者（色付きは本訴原告） ユーザー企業	ITベンダー	請求
2009年1月	A社	V社	本訴●A社がV社に約5億1000万円（損害賠償）を請求 反訴●V社がA社に約2億1000万円（未払金）を請求
2009年9月	トクヤマ	TIS	本訴●トクヤマがTISに約18億円（賠償、原状回復請求等）を請求 反訴●TISがトクヤマに約2億4000万円（未払金）を請求
2009年11月	B社	W社	本訴●W社がB社に約14億6000万円（未払金等）を請求 反訴●B社がW社に約24億5000万円（賠償、原状回復請求等）を請求
2010年8月	C社	X社	本訴●C社がX社に約5億円（損害賠償）を請求 反訴●X社がC社に約4億5000万円（未払金）を請求
2010年8月	旭川医科大学	NTT東日本	本訴●NTT東日本が旭川医大に約22億8000万円（損害賠償）を請求 反訴●旭川医大がNTT東日本に約19億4000万円（損害賠償）を請求
2010年11月	読売新聞東京本社	アクセンチュア	本訴●読売新聞がアクセンチュアに約36億2000万円（代金返還）を請求 反訴●アクセンチュアが読売新聞に約2億7000万円（未払金等）を請求
2011年3月	D社	Y社	本訴●Y社がC社に約4億円（損害賠償等）を請求 反訴●C社がY社に約11億6000万円（支払済開発費返還）を請求
2012年7月	らでぃっしゅぼーや	大塚商会	本訴●大塚商会がらでぃっしゅぼーやに約1億円（未払金）を請求 反訴●らでぃっしゅぼーやが大塚商会に約3億円（賠償等）を請求
2013年11月	野村ホールディングス 野村証券	日本IBM	本訴●野村ホールディングス等が日本IBMに約33億円（賠償等）を請求
2014年8月	テルモ	アクセンチュア	本訴●テルモがアクセンチュアに約38億6000万円（損害賠償）を請求
2015年8月	E社	Z社	本訴●Z社がE社に約5億円（未払金等）を請求 反訴●E社がZ社に約4億4000万円（損害賠償）を請求

※当該企業またはその親会社の年間売上高が1000億円未満の民間企業が関わる訴訟については、当事者を匿名で表記した　ERP：統合基幹業務システム

026 日経コンピュータ 2016.10.13

（参照：日経コンピュータ 2016.10.13号）

(3).ソースコードの解析方式

要件定義書を書けないという状況に直面した再構築担当者は、**「信用できるのはソースコードしかなく、正しい業務仕様はソースコードの中にのみ存在する」**と感じているかもしれません。

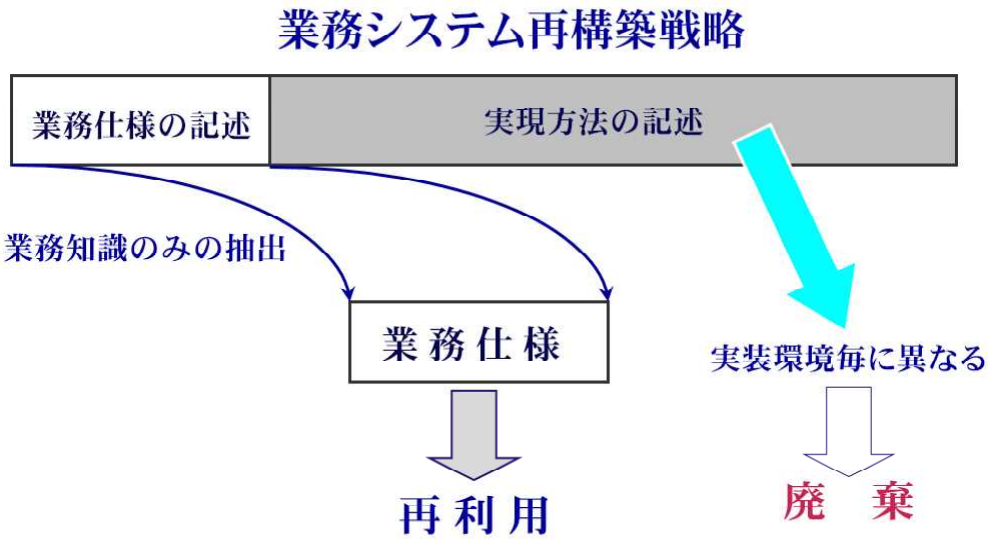
ですから、「人海作戦やツールを使ってソースコードを解析し、プログラム設

計書を作成して、そこから必要な業務要件を抽出しよう」と考えた方は多いと思います。

しかし、この発想には大きな落とし穴があります。業務システムは「**業務仕様の記述（What）**」と「**実現方法の記述（How）**」の二つの部分から構成されています。そして、業務システムの中で「業務仕様の記述」に費やしている部分は全体の約２０～３０％で，残りの約７０～８０％は「実現方法の記述」が占めています。

「業務仕様の記述」とは日々システムを利用している業務担当者が行っている業務の内容を記述したもので，「実現方法の記述」はその業務の内容を現実の実装環境上でシステムとして正しく稼働させるためのロジックを中心とした記述です。これまでの多くの自動化ツールは「実現方法の記述」に関する部分をソフトウェア部品の再利用によって自動的に生成しようとしています。

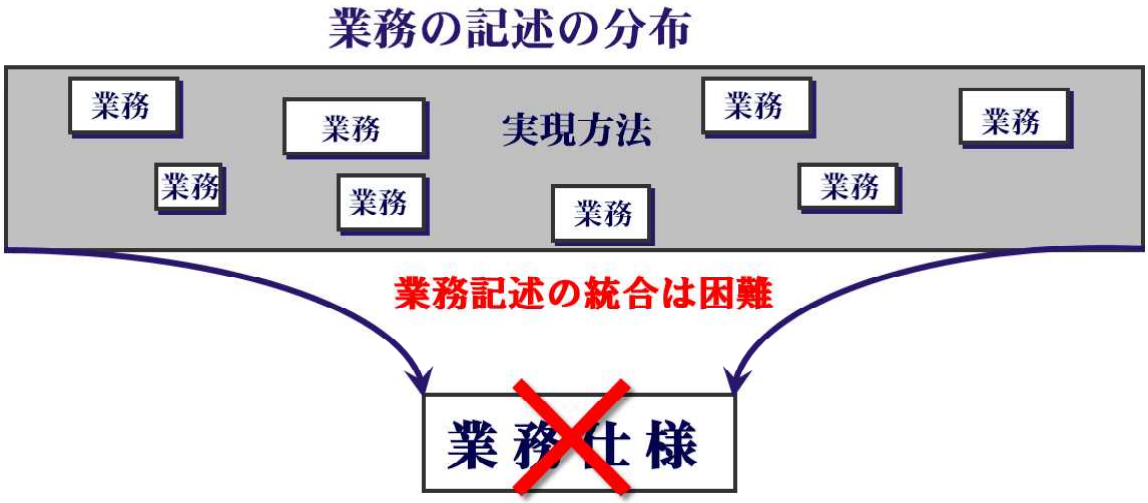
この「実現方法の記述」は各種の実装技術と密接に結びついているので，情報処理技術が進化すると作り直しが必要となります。一方，「業務仕様の記述」は実装技術とは関係がありません。どのように技術が進化しても，本来は作り直す必要はない部分です。



そこで、ソースコードを解析して「業務の記述」を抽出し、再利用しようとするのですが、残念ながらソースコードの中から「業務の記述」の部分のみを抽出するのは簡単な作業ではありません。

なぜなら、この「業務の記述」は，ソースコードの中では「実現方法の記述」と組み合わされ，バラバラに分解されて様々な箇所に分散しているのです。ですから，ソースコードの中から「実現方法の記述」を切り離し，バラバラに分解された「業務の記述」だけを抽出するのは至難の業であり、さらに、抽出された「業

務の記述」を整理し組み合わせて，要件定義書にまとめるのは不可能に近いでしょう。



さらに，日々の業務を推進している業務部門には，担当している業務に関する業務知識が蓄積されているはずですが，長年にわたるレガシーシステムの利用によって，業務部門ではシステムの利用知識が業務知識に置き換わっています。多くの会社が長年にわたって業務システムを利用し，業務の効率化を追求してきた裏で，組織の中では着実に要件定義書を書くための業務知識が失われてきたのです。

(4).強制コード変換方式

ツールを使い、COBOL 等のレガシーな言語を JAVA 等の最新の言語に変換するというやり方もあります。この方法は、言語を変えるだけなので、最もリスクが少ないように思われます。

しかし、情報処理技術は急速に変化していますので、言語変換だけでシステムが生まれ変わる筈はありません。特に、画面や振る舞い、帳票などは大幅に変化しています。

例えば、昔はグリーン画面や Win 画面でしたが、現在は Web 画面が標準で、スマートディバイスを利用するケースも増えてきました。しかも、言語や IT 環境の違いから、操作手順等単純に変換できないものが多いのです。

そして、強制的に言語変換されたソースコードを人が読み解くのは困難でしょう。つまり、運用開始以降の保守・改修作業が難しく、多額の保守費用が必要となります。

さらに、５～１０年後には、また再構築の話をしなければなりません。ほとんどブラックボックス化したソースコードを基に、どうやって再構築するのです

か。悪い夢を見ている心境になりませんか。

5．レガシー・システム再構築は可能なのか？

これまでの説明の通り、開発時に整備されていた設計書は、運用途中で様々な仕様変更や仕様追加が行われたため、ソースコードと乖離し、使い物になりません。

また、様々な変更が加えられたことにより、ソースコードはスパゲッティ状態となり、解析を困難にしています。ですから、ソースコードから詳細な業務仕様を抽出するのは、ほとんど不可能な作業です。

そして、情報処理技術が急速に進化しているため、レガシーシステムを開発した当時の実装の設計思想は再利用できません。全く新しい発想で取り組まねばなりません。

また、レガシーシステムを開発した当時のメンバーは、様々な理由で呼び戻すことは困難です。つまり、要件定義書を正しく書き、技術者に説明できるスーパーマンは存在しないのです。

しかも、経営層の理解は得られず、業務担当者の協力も得られません。業務担当者からは、「**出来上がったものを見せてくれれば、良いか悪いかの評価はしてやるよ**」と言われます。これではレガシー・システム再構築など不可能に思えます。

しかし、「**出来上がったものを見せてくれれば.....**」という業務担当者の一言が、ヒントになっているのではないでしょうか。

6．レガシーシステム再構築に利用できる情報

まず、レガシーシステム再構築のために使える情報を整理してみました。現状で入手可能な情報には以下のものがあります。もし、他にもあれば、喜んで集めておきます。

(1).業務担当者の利用している業務マニュアル

業務担当者は、日常業務の遂行や業務引継ぎ等のために、必ず、業務マニュアルを作成しています。この業務マニュアルは、レガシーシステムを開発した技術者が作成した利用者マニュアルを元に、業務担当者が日常業務を遂行している際の気付きや、注意事項を追加記述したものです。そこには、システムが対応していないので、仕方なく業務担当者が行っている処理も書かれています。

ただし、業務担当者には自分の担当業務しか見えていませんから、この業務マニュアルでシステム開発に必要な要件定義書を書くことは無理です。

(2).レガシーシステムのデータベースと設計資料等

レガシーシステムでは、Oracle 等のデータベースを利用している場合と VSAM 等のファイルを利用している場合がありますが、項目定義書だけは正しく保守されているのではないでしょうか。データベースの各テーブルやファイルの項目

定義書は再構築の際に重要な情報源の一つです。

これらのデータベースやファイルには、様々な呼び方はあるかもしれませんが、顧客、社員、商品等のマスター系と注文、発注、在庫といったトランザクション系があります。この違いを明確にしておくことも重要でしょう。

(3).実業務で使われているデータベースの生のデータ

レガシーシステムで使われているデータベースやファイルに蓄積されている生のデータは重要な情報です。例えば、「個別の注文データ」なのか、「集約した注文データ」なのか、「月次集計データ」なのかは、生のデータを見れば判断できます。

(4).レガシーシステムの画面、帳票上の項目、属性

レガシーシステムで使われている画面や帳票上の項目は、業務で使用されている項目として、また、業務の流れを解析する上で重要な情報です。

これらの項目を画面や帳票単位で整理し、キーとなる項目を識別します。それに、現在使われている属性や桁数も付け加えておきます。

(5).業務の流れ（業務フロー、又は、追跡調査）

整理された業務フロー図が残っている会社は素晴らしいのですが、業務フロー図が存在しない会社もあります。レガシーシステムの設計書が陳腐化しているのと同様に、業務フロー図も陳腐化してしまって、使い物にならなくなっているかもしれません。その場合は、画面やデータ項目に基づいた追跡調査が必要となります。

7．レガシーシステム再構築のやり方

既に述べたように、レガシーシステム全体を理解し、要件定義書を書けるスーパーマンは存在しません。しかし、現実に業務を遂行している業務担当者は存在します。ですから、業務担当者の持っている業務遂行上のノウハウは活用すべきです。

もちろん、彼らのノウハウだけで要件定義書を完成させることは不可能です。しかし、動くシステムを見てもらえば、その良否の判断と不足部分の指摘は可能でしょう。

ここで重要なことは、「**レガシーシステム再構築に当たって完璧な要件定義書を作成しようとは思わない**」ことです。現状で明らかになっている情報だけで再構築に着手します。ここで不明確な点や判断できない事項は、後日明らかになった時点で取り込めば良いのです。

各業務担当者の知識を全て組み合わせれば、全体像が見えてくるのでは？

（参照：GeneXus 社資料）

(1).現状分析と調査

どの様な会社にも、業務の現場には担当者が日々の業務を遂行するための、メモや業務マニュアル（手順書）があります。そして、担当者が入手する情報(Input、入力伝票)と他の担当者や部門に渡すための出力情報(Output、出力伝票、帳票)があります。業務は、複数の担当者が分担して遂行するので、複数の担当者や部門を通した「Input と Output」の連続によって構成されていると言えます。

業務の流れ図（業務フロー図）があればよいのですが、存在しない場合もこの Input と Output を時系列的に並べていけば、業務の流れが見えてきます。さらに、各業務の現場が一つの業務だけを行っているわけではありません。複数の業務について同様に業務の流れが見えてくれば、それらの業務の関連図も書くことができます。

レガシーシステムが作られた時代と今とでは、会社の実態も大きく変化しているでしょう。大切なことは、今の業務の実態を理解することで、使われていない過去の業務を分析しても仕方ありません。つまり、誰も使っていない画面や帳票は無視すべきです。

業務の分析を始めたときには処理内容が不明であっても、分析の後半で前の処理で行っておくべき内容が判明してくることもあります。必要な前処理が分かった時点で追加すれば良いのです。

或る複数のレガシーシステム再構築を成功させている方は、「**ブラックボックスになっている業務処理など存在しない**」と言い切っています。業務処理の内容や必要性が誰にも分からなければ、今は何かに記録しておきましょう。後で

誰かが困った状態に陥れば、必要な処理内容が明らかになってきます。処理内容が明確になれば、そのときに追加すればよいのです。もし、分析をしていて分からなくなれば、業務の現場に入って聞いてくるのです。

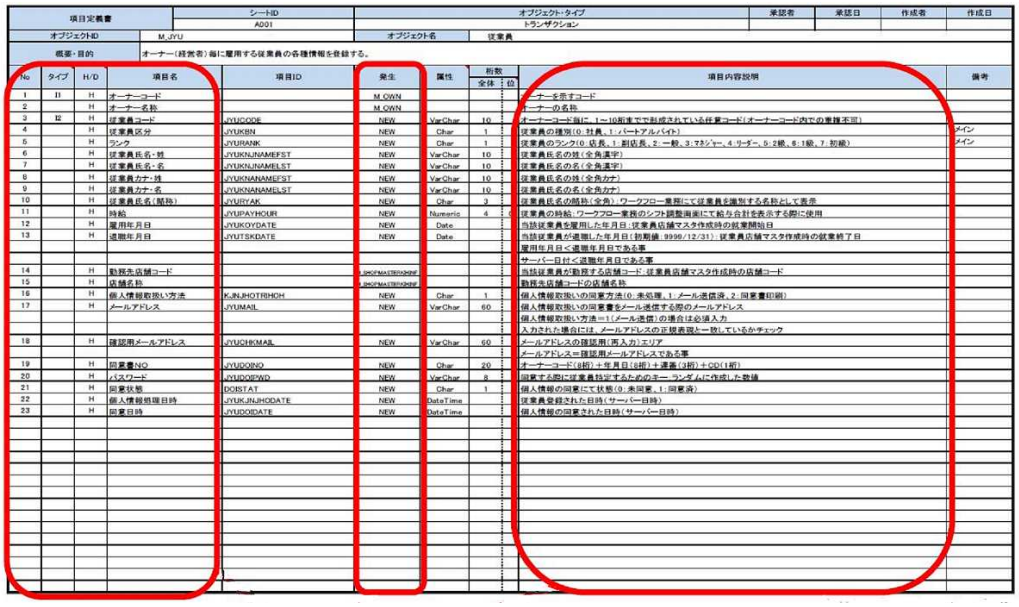
「現在、分かっている情報しか採用しない。情報が明確になったときに採用する」という方針が重要です。

(2).業務担当者からの情報収集

いずれにしても、キーマンは現実に業務を行っている業務担当者です。彼らから、現在使用している画面や帳票のコピーを入手します。さらに、帳票等から抜き出した項目を使って、手作業で加工し作成した報告書や伝票類のコピーも入手します。また、手作業で行っている作業のやり方と加工のルールも聞き出します。そのコピーには、生のデータが入っていなければなりません。生のデータが入っていない画面や帳票を使って分析するのは困難です。

そして、この作業を関係する全部門の全ての担当者に対して行います。この様にして入手した情報は脈絡のない情報に見えますが、それで良いのです。Input と Output を時系列的に並べれば、業務の流れは見えてきます。

次に、部門別に集めた資料から、業務の言葉で書かれた項目を集めて、項目定義書を作成します。この項目定義書に複数のデータが存在する場合の並べ方、データの識別方法に注目してください。例えば、顧客コードだとか、商品コードといったものです。これらはキー項目になるかもしれません。



項目定義書	シートID	オブジェクト・タイプ	承認者	承認日	作成者	作成日
	A001	トランザクション				
オブジェクトID	M_JYU	オブジェクト名	従業員			
概要・目的	オーナー（経営者）毎に雇用する従業員の各種情報を登録する。					

No	タイプ	H/D	項目名	項目ID	発生	属性	桁数 全体	桁数 小数	項目内容説明	備考
1	II	H	オーナーコード		M_OWN				オーナーを示すコード	
2		H	オーナー名称		M_OWN				オーナーの名称	
3	II	H	従業員コード	JYUCODE	NEW	VarChar	10		オーナーコード毎に、1～10桁までで形成される任意コード（オーナーコード内での重複不可）	
4		H	従業員区分	JYUKBN	NEW	Char	1		従業員の種別（0：社員、1：パートアルバイト）	メイン
5		H	ランク	JYURANK	NEW	Char	1		従業員のランク（0：店長、1：副店長、2：一般、3：マネジャー、4：リーダー、5：2級、6：1級、7：初級）	メイン
6		H	従業員氏名・姓	JYUKNJNAMEFST	NEW	VarChar	10		従業員氏名の姓（全角漢字）	
7		H	従業員氏名・名	JYUKNJNAMELST	NEW	VarChar	10		従業員氏名の名（全角漢字）	
8		H	従業員カナ・姓	JYUKNANAMEFST	NEW	VarChar	10		従業員氏名の姓（全角カナ）	
9		H	従業員カナ・名	JYUKNANAMELST	NEW	VarChar	10		従業員氏名の名（全角カナ）	
10		H	従業員氏名（略称）	JYUIRYAK	NEW	Char	2		従業員氏名の略称（全角）：ワークフロー業務にて従業員を識別する名称として表示	
11		H	時給	JYUPAYHOUR	NEW	Numeric	4	0	従業員の時給：ワークフロー業務のシフト調整画面にて給与合計を表示する際に使用	
12		H	雇用年月日	JYUKOYDATE	NEW	Date			当該従業員を雇用した年月日：従業員店舗マスタ作成時の就業開始日	
13		H	退職年月日	JYUTSKDATE	NEW	Date			当該従業員が退職した年月日（初期値：9999/12/31）：従業員店舗マスタ作成時の就業終了日	
									雇用年月日＜退職年月日である事	
									サーバー日付＜退職年月日である事	
14		H	勤務先店舗コード		JUKOFMASTERKANR				当該従業員が勤務する店舗コード：従業員店舗マスタ作成時の店舗コード	
15		H	店舗名称		JUKOFMASTERKANR				勤務先店舗コードの店舗名称	
16		H	個人情報取扱い方法	KJNJHOTRIHOH	NEW	Char	1		個人情報取扱いの同意方法（0：未処理、1：メール送信済、2：同意書印刷）	
17		H	メールアドレス	JYUMAIL	NEW	VarChar	60		個人情報取扱いの同意書をメール送信する際のメールアドレス	
									個人情報取扱い方法＝1（メール送信）の場合は必須入力	
									入力された場合には、メールアドレスの正規表現と一致しているかチェック	
18		H	確認用メールアドレス	JYUQHKMAIL	NEW	VarChar	60		メールアドレスの確認用（再入力）エリア	
									メールアドレス＝確認用メールアドレスである事	
19		H	同意書NO	JYUDOINO	NEW	Char	20		オーナーコード（8桁）＋年月日（8桁）＋連番（3桁）＋DD（1桁）	
20		H	パスワード	JYUDOPWD	NEW	VarChar	8		同意する際に従業員特定するためのキー ランダムに作成した数値	
21		H	同意状態	DOISTAT	NEW	Char	1		個人情報の同意にて状態（0：未同意、1：同意済）	
22		H	個人情報処理日時	JYUKJNJHODATE	NEW	DateTime			従業員登録された日時（サーバー日時）	
23		H	同意日時	JYUDOIDATE	NEW	DateTime			個人情報の同意された日時（サーバー日時）	

（参照：A&A 安藤氏講演資料より抜粋）

当初は、赤く囲った部分しか記入できませんが気にしないでください。また、この段階では、部門毎に同じ項目の呼び名が違っていることがあります。この

ような場合は、業務担当者と話し合い、同じ項目であることを確認します（異音同義語）。さらに、同じ呼び名ですが、生のデータを見ると明らかに内容の違う項目も出てきます（同音異義語）。この場合は、項目名を変更します。

そして、この段階で業務の担当者が知っているビジネスルールは書き留めておきます。後で、このルールを入力することになります。

(3).既存データベースやファイルの情報収集

レガシーシステムが利用しているデータベースは、後ほど取り込むことになります。一方、既存のファイルについては、ファイル定義書に基づいて別途項目定義書を書いておきます。そして、そのデータはCSVやテキストデータに変換し、新たな環境でデータベース化することになります。

先ほど、これらのデータベースには、顧客、社員、商品等のマスター系と注文、発注、在庫といったトランザクション系があると書きました。

マスター系のデータベースは、業務の枠を超えて別のシステムでも利用されていますので、安易に変更することはできません。ですから、マスター系データベースはそのまま利用することになるでしょう。後日、チャンスがあれば、その時点で作り直せば良いのです。

一方、トランザクション系のデータベースは、その業務内に閉ざされたものが多いので、再構築する方向で話を進めます。もちろん、他のシステムでこれらのトランザクション系のデータベースを利用しているケースもありますから、事前に影響調査を行う必要があります。

(4).項目辞書の作成

次に、これまでに作成した項目定義書から項目辞書を作成します。

ユーザビュー	データ項目名	主体	カテゴリー	クオリファイア	補足	アトリビュート名
受注登録	受注番号	受注			番号	SONo
受注登録	受注合計金額	受注	合計		金額	SOTotalPrice
受注登録	受注明細番号	受注	明細		番号	SODetailNo
受注登録	受注明細商品数量	受注	明細	商品	数量	SODetailPrdQty
受注登録	受注明細小計金額	受注	明細	小計	金額	SODetailSubtPrice
顧客管理	顧客名称	顧客			名称	CustNm
顧客管理	顧客住所	顧客			住所	CustAdr
顧客管理	顧客受注限度金額	顧客	受注	限度	金額	CustSOLimitPrice

（参照：A&A 安藤氏講演資料より抜粋）

項目名で並び替えられた一覧表に GeneXus 社の提唱している GIK(GeneXus Incremental Knowledge Base)という命名法を用いて、英文名を付与します。

GIK命名法

主体 + カテゴリ + クオリファイア + 補足

主体　　:アプリケーションの実体(エンティティ)やオブジェクトを表す部分
カテゴリー　:属性の持つ意味のカテゴリーを表す部分
クオリファイア　:形容詞や副詞で項目属性の意味をユニークにする
　例:Start, End, Due, その他
補足　　:自由なテキスト(注釈や属性を含める)
　例:Id, Code, Name, Date, Description, Price, Stock, その他

(参照：A&A 安藤氏講演資料より抜粋)

全ての項目定義書の項目に対し、ここで自動的に生成された英文項目名と項目の属性、桁数を付与します。

(5).既存データベースの取り込み

既に存在し、該当する業務で使用するマスター系データベースを、GeneXus のデータベース・リバース・エンジニアリング・ツール(DBRET)を用いて、GeneXus に取り込みます。GeneXus は取り込んだデータベース用のトランザクションとデータビューを作成します。このトランザクションは正規化対象外ですが、他のトランザクションと同じように使用できます。

(参照：GeneXus 社資料)

これから開発するに際して、業務担当者が良否を判断するために生のデータが必要となります。業務部門の責任者の方と相談し、実際に動いているシステムから或る条件でデータを抽出し、テスト用の小さなデータベースを作成しておきます。

(6).項目定義書を GeneXus に投入

これまで作成してきた項目定義書に英文名を付与し、業務の流れに従ってトランザクションとして GeneXus に投入します。この際、明確になっている計算式やビジネスルールも入力します。

該当する業務担当者が扱う機能に関するトランザクションの投入が終わったら、実際にビルドをし、生成された画面を業務担当者に触ってもらうことになります。

(7).画面の標準形式の作成

業務の機能を確認するために自動生成された画面を業務担当者の方に触ってもらいますが、その際には事前にマスターページや CSS(Cascading Style Sheets)を使って、標準的なレイアウトや配色等を決めておくと、業務担当者の受けが良くなります。

早い段階で、業務担当者の方々に画面のデザインを刷り込んでおく方が良いでしょう。**絶対に、業務担当者に画面のレイアウト作成を依頼してはいけません。**

(8).業務担当者に触ってもらう

「動くものができたらチェックしてやるよ」と言っていた業務担当者に、実際に動くシステムを触ってもらいます。この際、画面レイアウトを気にしては駄目です。業務で使用するデータ項目と機能を先ず確認してもらいます。

実際に動くシステムを触ると、業務担当者は様々な意見を述べてくれます。例えば、「この数値は違っている」、「表示の桁数が足りない」、「合計金額も表示してほしい」、「操作画面の表示順が変だ」、「ここに表示するのは合計値で、個々の値ではない」等です。

業務担当者からの要望はできるだけ反映し、業務担当者が納得するまで繰り返します。ただし、業務担当者の気持ちが揺れ動くのに付き合っていては収束しませんから、そこはプロジェクト管理者が繰り返し回数に制限を設ける、といった牽制が必要です。実現した機能に納得してもらえたら、業務担当者から承諾書にサインをもらいます。

(9).バッチ処理の作成

ところで、業務担当者に動く画面を触ってもらうと、年初以来の累積値、平均値、高安値、去年の値からの伸び率、大量なデータを処理しないと得られない値、締め処理等といった、バッチ処理でないと算出しづらい値を要求されることがあります。

このような場合は、業務担当者の要望に従って、プロシジャーを用いてバッチ処理を作成します。なお、バッチ処理の駆動タイミングは、Web 画面から起動したり、リアルバッチ処理であったり、締め処理のようにスケジューラで起動をかける場合もあります。

(10).関係する全ての業務担当者に触ってもらう

そして、次の業務担当者にも同様に、業務の流れに従って、実際に動くシステムを触ってもらいます。この際、生のデータから作成したテスト用のデータを使用すれば、誤解等による項目の不整合を発見できます。

そして、全ての業務担当者の方々に触ってもらい、要望や気付きを受け入れて、当初契約で約束していた全ての機能を開発したら、業務部門の責任者の方に承諾書のサインをもらいます。

(11).UI のレイアウトのブラシュアップとパフォーマンスのチューニング

業務で使用するデータ項目と機能の確認作業が終了したら、次は、見栄えを直していきます。

開発に参加されていた業務担当者の方々は、既に、自動生成された画面に慣れていますから、大幅な直しは発生しづらくなります。できるだけ、画面作成ツールを使って、見栄えの良い画面を作成します。帳票については、プロシジャーを用いて作成します。

そして、実環境と同じデータ量を用いたパフォーマンステストを実施します。チューニングの方法については、別途説明しています。

(12).データベースの移行

データベースには、商品マスターや顧客マスターといったマスター系データベースと注文・約定データ、在庫データといったトランザクション系データベースがあります。

マスター系データベースは他のシステムも利用していますから、移行を行わないケースが多いと思います。問題となるのはトランザクション系データベースです。

基本的には、トランザクション系データベースは移行したくはなく、過去の取引データが必要なら、並行ランの期間をのばし、過去のデータを別途保存して、取り出して使えるようにしておきたいのですが、そうもいきません。データベースの移行が必要になった場合には、データの中身を疑わなくてはなりません。

例えば、数値データと文字データが混在していたり、特殊なキー項目を作って、意味の違うデータを格納していたり、といった例外処理の残骸が多く含まれています。ですから、***データのクレンジング*（Data Cleansing）が必要となります。このデータのクレンジングは非常に手間のかかる作業です。最初にシステムを作った会社に任せるのが正解でしょうが、そうもいきません。

ですから、データベースの移行は請負契約とせず、ユーザ責任の下で準委任

契約とすべきです。場合によっては、レガシーシステム再構築の費用よりも多額の費用が発生することもあります。

(13).システムテストや運用テストの準備と実施

全ての機能や画面・帳票ができた段階で、業務部門が主体のシステムテストや運用テストを実施します。事前にテストケースを洗い出し、シナリオを作成しておくとよいでしょう。

業務担当者が一緒になって開発していますので、この段階で大幅な仕様変更はほとんど発生しません。

(14).並行ランの準備と実施

運用テストが終わった段階で、本番環境と同じデータを用いた並行ランを実施します。並行ランというのは、実質的に本番運用と同じです。並行ランの期間に、全てのイベントが発生していることを確認してください。

この並行ランを実施していると、現在のレガシーシステムの出力結果とは違った結果が出てくることがあります。この原因は、これまで誰も気付かなかった過去の仕様変更や隠れ仕様等によるものが多いでしょう。

これらの出力結果の違いは、業務部門と一緒になって調査し、原因を明らかにします。そして、修正を加え、再度処理を行います。

もし、調査をしても原因が判明しない場合、業務上の支障がなければ、メモを残して、そのままにしておきます。後日、原因が明確になった段階で修正を加えれば良いのです。

(15).本番運用の開始

並行ランというのは、実質的に本番運用と同じです。一定期間の並行ランで致命的な間違いが出現せず、実用に耐えるレベルと判断できれば、業務部門と相談の上で、レガシーシステムの運用を中止します。この際、年間を通した全てのイベントを、正しく処理できていることを確認してください。

8．ツールが持つべきレガシーシステム再構築に必要な機能

レガシーシステム再構築では、完全な要件定義書を作成することは期待できません。そして、完全な要件定義書が作れることを前提としたウォーターフォール型開発手法ではうまくいきません。

さらに、ハードウェアの保守期限の問題やＯＳＳ(Open-source software)のＥＯＬ(End of Life)の問題等が迫ってきますので、レガシーシステム再構築にそれほど時間はかけられません。ですから、有効な開発ツールを利用する、という選択肢は考えておくべきでしょう。では、レガシーシステム再構築に利用する開発ツールに求めるべき機能と

は何でしょうか。

(1).仕様変更や仕様追加に柔軟に耐えられる機能

レガシーシステム再構築では、完璧な要件定義書を作ることは困難ですから、曖昧な業務仕様の下で再構築に取り組むことになります。業務担当者がテストに参加し、また、現在のレガシーシステムとの並行ランを実施すると、膨大な量の仕様変更が発生し、隠れ仕様の追加要求が出てきます。

これは開発要員を増やせば対応できる問題ではありません。開発ツールがこれらに柔軟に対応できないと、開発費用が増大し、再構築が失敗してしまう危険性が高いのです。

(2).データベースの変化にも耐えられる機能

さらに、レガシーシステム再構築では、データベース構造が大幅に変更になる可能性も考慮に入れておかねばなりません。

事前に明確な要件定義書を作成できる開発では、事前に確固としたデータモデルを設計し、実装に入ることができます。しかし、不明確な業務仕様の下で着手しなければならないレガシーシステム再構築では、確固としたデータモデルは期待できません。開発の途中でデータベースの構造が変化してしまう可能性もあるのです。

つまり、データモデルの構造が変更になっても、ツールがその変更を受け入れ、自動的に対応してくれる機能が必要です。

(3).ソースコード（実装のアルゴリズム）のテストが不要

仕様変更や仕様追加が当然のように大量発生し、しかも、データモデルが変更になる場合もあるという前提でレガシーシステム再構築を考えた場合、変更の都度ソースコードのテストを行っていたのでは、テストの工数が膨大となり、テスト期間も長くなり、常に予算超過の危険性に怯えていなければなりません。

つまり、大幅な変更が発生しても、ソースコード（実装のアルゴリズム）のテストを行わなくても済むようなツールが必要です。

(4).システム開発ツールの自動生成の割合

システム開発ツールの中には、70 ～ 80 %を自動生成し、残りは JAVA 等を書き加えるというものもあります。しかし、手作業でプログラムを書き加えると、ソフトウェアの信頼性や柔軟性に問題を残します。当然、仕様変更や仕様追加には対応しづらくなります。

多くの高速開発ツールはソフトウェア部品を効率的に組み合わせて生産性を上げようとしていますが、これらのツールには本質的な弱点があります。それ

は「**ユーザの希望する全ての機能を、ソフトウェア部品として提供することはできない**」ということです。提供できない機能は、独自に JAVA 等を書き加えねばなりません。これでは仕様変更や仕様追加に対して柔軟に対応できません。

それに、ソフトウェア部品は、ツールが開発された当時の最新の技術を使って作られています。多くのツールは JAVA を採用していますが、「JAVA だけは変化しない」と保証されているわけではありません。近日中に、新しい言語に変わってしまうとは思いませんが、いずれ廃れてしまうかもしれません。

ですから、「**システム開発ツールは実装技術から独立しているべき**」なのです。情報処理技術が進化しても、生成をやり直せば最新の技術を使ったシステムに生まれ変わるべきなのです。そうでなければ、業務システムを守ることはできないでしょう。

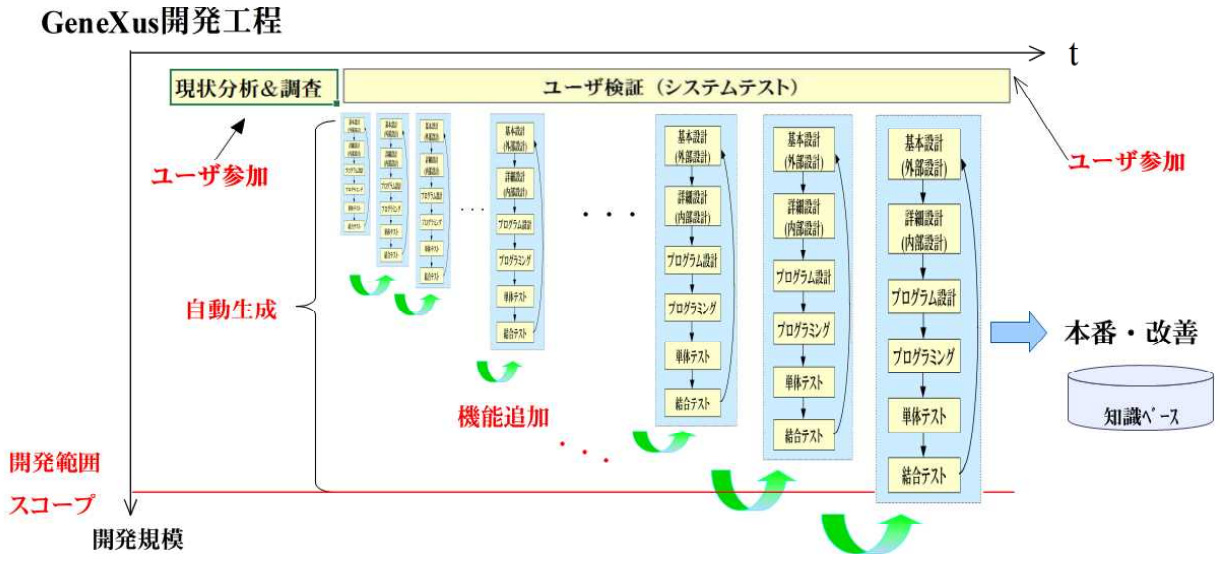
9．GeneXus でレガシーシステム再構築が可能なのか？

実は、日本での GeneXus 開発実績の中で、比較的大規模(例えば、JAVA ベースで 6M ステップ以上)なものは、ほとんどレガシーシステム再構築の事例です。GeneXus でレガシーシステム再構築が成功する理由は、既に御理解されたと思います。

要するに、

①「ユーザは完成したシステムを見て、始めて自分に必要な要件を理解する」というユーザの特性に合わせ、本物のシステムをユーザに見せながら開発可能
②業務要件が不明確でも、分かっている箇所から開発可能
③仕様変更や隠れ仕様の追加に短期間で柔軟に対応可能
④データベース構造が変化しても対応可能
⑤ソースコード（実装のアルゴリズム）のテストは不要
⑥実装技術から独立し、100 %の自動生成が可能

といったレガシーシステム再構築に必要な機能を、GeneXus は全て備えているのです。



10. まとめ

この「レガシーシステム再構築に関する考察」では、できるだけ一般的な分析を行うように心掛けました。難しさや問題点を洗い出すまでは良かったのですが、再構築を可能とする解決策をまとめているうちに、どうしても GeneXus を使わなければ解決できない、と思い至りました。

現実に、他の方法を用いて挑戦したレガシーシステム再構築プロジェクトの多くが失敗しています。

例えば、「**ソースコードから業務知識を抽出し、それをまとめて要件定義書を書き、プログラムを再作成する**」という方法は、限りなく不可能です。ソースコードに書かれている実装のためのアルゴリズムは解析できても、その業務上の意味を理解することはできません。

また、「**ERP を採用し、自社の業務フローに合わせてカスタマイズする**」という方法も失敗事例が多いです。ERP パッケージをそのまま使えば良いのですが、結局カスタマイズを試みる時点で、曖昧な業務知識の問題に陥っているのです。

しかも、全ての他の方法に共通するのは、「**情報処理技術の進化への対応**」に関する課題です。たとえ、今回レガシーシステム再構築に成功したとしても、5 ～ 10 年後には再構築の話を蒸し返さなければなりません。そのとき、状況は今以上に深刻になっているかもしれません。ですから、「**二度とレガシーシステムにしてはならない**」のです。

ユーザ企業が目指すべきは、新たな収益機会の獲得やビジネスモデルへの挑戦であって、「**貴重な経営資源を、レガシーシステム再構築やその保守といった非生産的なものに浪費すべきではない**」と考えるのですが、いかがでしょうか。

以上