GeneXus™

ジェネクサスが生まれた背景と目的

－『業務知識に基づくシステム開発
(Knowledge-Based Development)』の注釈－

改修 平成 27 年 3 月
平成 26 年 5 月

ジェネクサス・ジャパン株式会社

技術士（情報工学部門）
代表取締役社長　大脇 文雄

１．ソフトウェア危機

ソフトウェア危機（Software Crisis）という言葉を覚えていらっしゃいますか。
WIKIPEDIA を検索すると、「**ソフトウェア工学が未発達な頃によく使われた言葉**」と書かれています。
この危機が叫ばれ始めたのは、1968 年の第 1 回 NATO ソフトウェア会議であり、40 年以上も昔の話です。

そして、WIKIPEDIA には、ソフトウェア危機が発生した原因は「**ソフトウェア開発工程の全体としての複雑性**」と「**ソフトウェア工学の相対的な未熟さ**」に関連しており、下記の様な様々な手法や方法論の開発によって解決されつつある、と書かれています。

- オブジェクト指向、カプセル化
- 構造化プログラミング
- ガベージコレクション
- アジャイルソフトウェア開発
- マルチスレッド・プログラミング
- ソフトウェアコンポーネント
- ソフトウェアプロトタイピング
- デザインパターン
- 統合開発環境、RAD
- バグ管理システム、バージョン管理システム
- 反復型開発
- Model View Controller

でも、本当にソフトウェア危機の問題は解決されつつあるのでしょうか。現在でも、当時と同様に以下の様な問題は頻繁に発生しています。

- 予算超過
- 開発期間の超過
- 低品質なソフトウェア
- 要求仕様を満たさないソフトウェア
- 保守困難なソフトウェア

ソフトウェアの黎明期に指摘された問題は、決して「**ソフトウェア工学の相対的な未熟さ**」に起因したものではなく、「**ソフトウェア開発工程全体としての複雑性**」に起因する本質的な問題だったのではありませんか。

現在でも、ソフトウェア危機の問題は全く解決していないのです。

２．元大学教授 Breogan Gonda 氏の理論

JICA の派遣専門家として主に東南アジアの仕事をしていた頃、突然ウルグアイでの調査の話が舞い込んできました。ウルグアイという国はウガンダと間違えるほど馴染みが無く、全く予想もしない話でした。さっそく、インターネットでウルグアイについて調べましたが、牛肉と牛革、乳製品が主力の牧畜産業の盛んな国でした。まさかソフトウェア産業が発達しているとは思いもしませんでした。

ウルグアイには 2002 年の９月から 12 月中旬まで約３ヶ月間滞在し、調査活動を行ないました。政府機関や学校、そして、企業を訪問し、約８０人の方々と面談をしましたが、多くの方々から元公立大学教授であるゴンダ氏と面談する様に勧められました。

2002 年 10 月 31 日午前 10 時に、モンテビデオの旧市街にオフィスを構えていたアルテッチ社を訪問し、ゴンダ社長と会いました。

最初の１０分間、彼は学者としての自分のキャリアとシステム開発に対する考え方、そして、自分が研究し開発した技術について説明されました。この１０分間の会話を通し、ゴンダ氏の科学者・技術者としての素晴らしさは理解できました。

しかし、彼の「**業務仕様を入力すると、推論により実現方法を含む設計情報を自動的に生成し、希望するＩＴ環境用のシステムを自動的に生成する**」という技術の説明は信じられませんでした。

眉唾だと感じ「そんな技術がこの世の中に存在する筈は無いよ」と疑問を投げかけると、彼は「私はこの技術で１０年以上ビジネスをしている。既に、ホンダ・ブラジルやトヨタ・ブラジルも利用している。更に、銀行パッケージさえ存在し、多くの銀行も使用している」と反論しました。その時、「もし、彼の技術が本物なら、長年悩み続けてきたソフトウェア危機の問題とソフトウェアに関するパラドックスが解決するかもしれない」と感じたのです。

３．もし、彼の言葉が本当なら….

ここで説明するパラドックスは、1970 年代の IBM Review に書かれていた記事の内容です。

パラドックス-1)「システムは完成した直後から陳腐化が始まる」

ＩＴ技術の進化は早く、「２年毎に新しい技術が生まれ、５年で総てが変わってしまう」と言われています。ですから、１～２年をかけてシステム開発をしていると、開発を始める時に採用した最新技術は、完成時には過去の産物となってしまいます。

特に、スマートディバイスを使うシステムの開発では、悲惨な状態が待っています。

パラドックス-2) 「ユーザは完成したシステムを見て、はじめて自分に必要な要件を理解する」

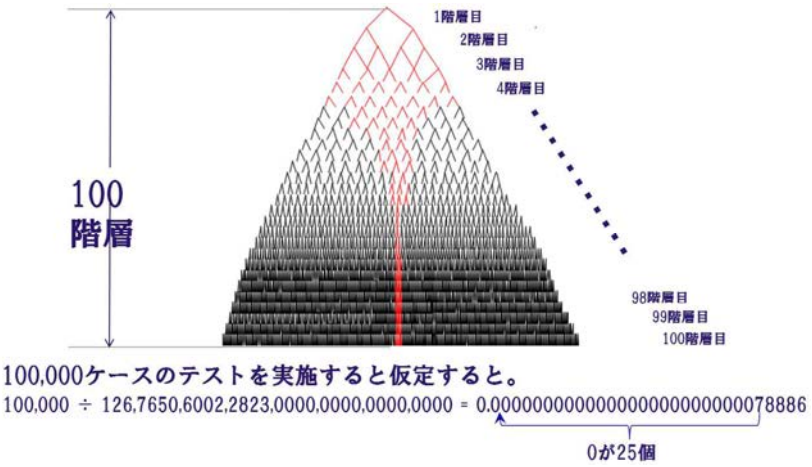
どんなに時間をかけて業務を分析し、要件を確定しても、ユーザによるシステム・テストが始まると、必ず多くの仕様変更の要請が上がってきます。これは、ユーザの責任でも仕様を分析した技術者の責任でもありません。ユーザとは「**完成したシステムを見て、はじめて自分が必要とするシステムの機能を理解できる**」ものなのです。ですから、要件分析の段階から実際に動くシステムをユーザに見てもらい、本当に必要な機能を確認してもらうしかありません。

パラドックス-3) 「ソフトウェアの正しさを、テストによって証明することはできない」

どれほど多くのテストケースを通しても、ソフトウェアの正しさをテストによって証明することはできません。これはソフトウェアのアルゴリズムの組み合わせを考えれば容易に理解できることです。

例えば、条件分岐文が１００個あるプログラムを想定しましょう。単純化するために二点分岐だけとします。条件分岐文の組み合わせによる総てのケース数は２の１００乗となります。これは１２６穣（じょう）という聞いたことも無い桁の数です。これだけのテストケース数を１秒間に１億ケース通せる環境で処理したとすると、総てのケースを通し終わるまでに４０１兆年かかることになります。ですから、テストによってソフトウェアの正しさを証明することは不可能です。

もし、仮に１０万ケースを用意し、正しく動くことを確認できたとしても、１２６穣（じょう）という分母が大きすぎるため、総てのテストケース数に対する割合を見ると、小数点以下に０が２５個も続いてしまいます。この値は統計学的に見て有意ではありません。即ち、全くテストを行っていない状態と同じ、ということになります。



一方、**GeneXus** は投入された業務の記述を推論機能によって解析し、実現方法を含む設計情報(Formal Mothods：形式仕様記述言語)を生成しますから、設計情報にバグは発生しません。また、この設計情報に基づいて、ソースコードが自動生成されていますから、アルゴリズムの総ての組み合わせを通すテストは不要です。

アルゴリズムの総ての組み合わせを通すことは不可能でも、業務で発生し得る総てのケースを洗い出すことは可能です。ですから、GeneXus で開発した場合は、業務で発生し得る総

てのケースをテストすれば、信頼性を担保することが可能となります。

４．システムを利用する組織の悩み

日本にコンピュータが導入されてから約６０年が経ちました。その間に、多くの組織で膨大な数のシステムが開発され、長年にわたって利用されてきました。

ところが、20～30 年前に開発されたシステムの保守（改善）作業に膨大な費用と時間がかかっています。例えば、帳票に一つの項目を追加するだけで、半年間の作業期間と数千万円の作業代金を請求された事例もあります。米国の某有名企業の例では、**「技術者は古いコードの調査に８０％の時間を費やし、その修正に１８％の時間を使い、残りの２％の時間で新たなコードを書くことができる」**そうです。それほどまでにシステムの保守（改善）作業にはお金と時間がかかります。

更に、事態を深刻化させているのがＩＴ技術の進化です。「ＩＴ技術は２年毎に新しい技術が出現し、５年で総てが変わってしまう」と言われています。そして、ＩＴ技術のＥＯＬ(End of Life)の問題も出てきています。

更に、スマートディバイスを端末として使用する傾向も多くなってきました。「ＩＴ技術が経営を支えている」という認識の下で、常に新しい技術の導入を続けていけば、膨大な費用と時間が必要となります。ＩＴ技術の変化に追随するため、業務内容がほとんど変化しないのに、５～７年毎にシステムの再構築を繰り返し、度重なる高額な投資を行っていけば、経営を圧迫することになります。

しかも、20～30 年前のシステム開発に携わった技術者達は高齢化し、既に引退し、システムから遠ざかっているケースが多いのです。ですから、今のうちにシステムを再構築し、システムに関わる問題を解決する必要はありませんか。

「システムの TCO(Total Cost of Ownership)を半分以下にしたい」、「経営環境の変化に対するシステムの対応力とスピードを高めたい」、「業務の見える化と共に、システムのレガシー化を回避したい」、「IT 技術の進化に惑わされることなく、業務の保守（改善）に専念したい」等々、といった会社のニーズにどの様に答えるのですか。

５．プログラムの代わりに業務を記述

業務システムの構造は、「約 20%が業務内容に関する記述で、約 80%が各ＩＴ技術に基づく実現方法に関する記述」と言われています。

実現方法の記述は、業務システムが作られる際のＩＴ技術に大きく依存し、OS、DBMS、言語、ディバイスなどの種類によって異なります。

この 80%の実現方法を記述するために、内部（詳細）設計、プログラム設計、プログラミング等の作業に膨大な時間と人とコストが費やされています。しかし、厄介な問題は、この実現方法の記述がＩＴ技術の進化の影響を大きく受け、数年で陳腐化してしまうので、再構築するにはもう一度設計からやり直すしかないことです。

一方、業務内容の記述はＩＴ技術の進化の影響を受けません。

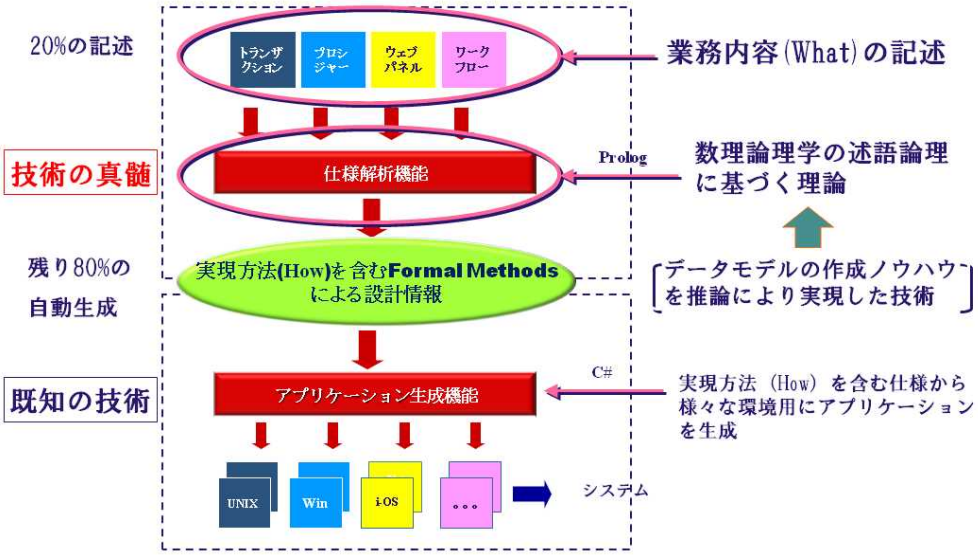
GeneXus は業務の記述（データの記述が 70%、業務手続きの記述が 30%）から実現方法の記述（設計、プログラム、データベース）を推論により自動的に生成します。**GeneXus** を利用すると、これらの実現方法の記述をする必要はありません。業務内容の記述をするだけで、業務システムの構築が可能となります。

６．業務の記述から自動生成する仕組み

業務の記述から実現方法の記述を自動生成する仕組みは以下の様になっています。

どの様なシステム開発でも、まず業務内容を整理し、開発を担当する技術者に理解して貰わねばなりません。

GeneXus には、この整理した業務内容の記述を投入します。



GeneXus は投入された業務の記述を解析し、実現方法を含む設計情報(Formal Mothods：形式仕様記述言語)を推論により自動的に生成します。この仕様解析機能は、過去に日本の第五世代コンピュータプロジェクトで採用された Prolog によって書かれています。

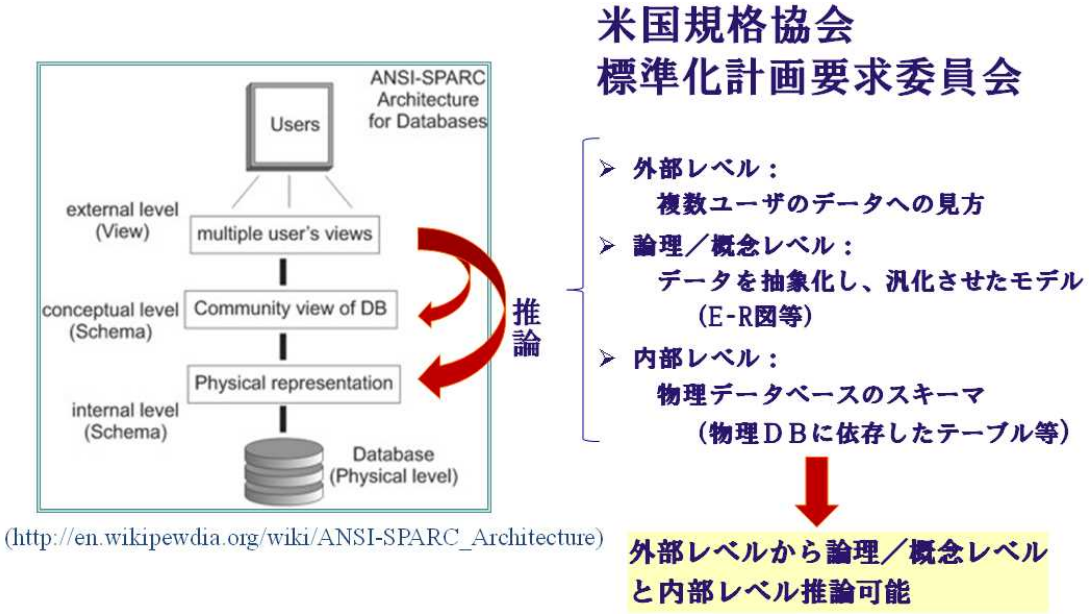
そして、ここで生成された実現方法を含む設計情報に基づき、アプリケーション生成機能が、要求された実装環境用のアプリケーションと物理データベースを自動生成します。

上図の一番下右端の箱には「・・・」と書かれています。これは、今は存在していない OS・DBMS

・言語、そして、ディバイスにも将来対応できる事を示しています。

ところで、 **GeneXus** と他のツールとの違いの説明を頻繁に求められます。
違いは簡単です。他のツールは、この絵の上部の機能（仕様解析機能）を持っていません。世の中には、設計情報としてのデータモデルや UML 等を投入すると、ソースコードを自動生成してくれるツールは無数に存在します。しかし、業務の記述からアプリケーションを自動的に生成してくれる技術は、現在のところ **GeneXus** しか持っていません。この違いの影響は、開発保守（改善）の工程で顕著に現れるでしょう。

7．ANSI-SPARC Architecture（三層構造）



(http://en.wikipewdia.org/wiki/ANSI-SPARC_Architecture)

これは ANSI（American National Standards Institute：米国規格協会)-SPARC（Standards Planning and Requirements Committee：標準化計画要求委員会）が 1975 年に提案したデータベースの三層構造です。

① 外部レベル(external level)：
multiple user`s view（複数ユーザのデータの見方）を単純に集めたものです。

② 概念（論理）レベル(conceptual level)：
Community view of DB（同じ業務を分担し遂行している業務担当者が共通に使うデータベース）で、正規化されたデータモデルのことです。

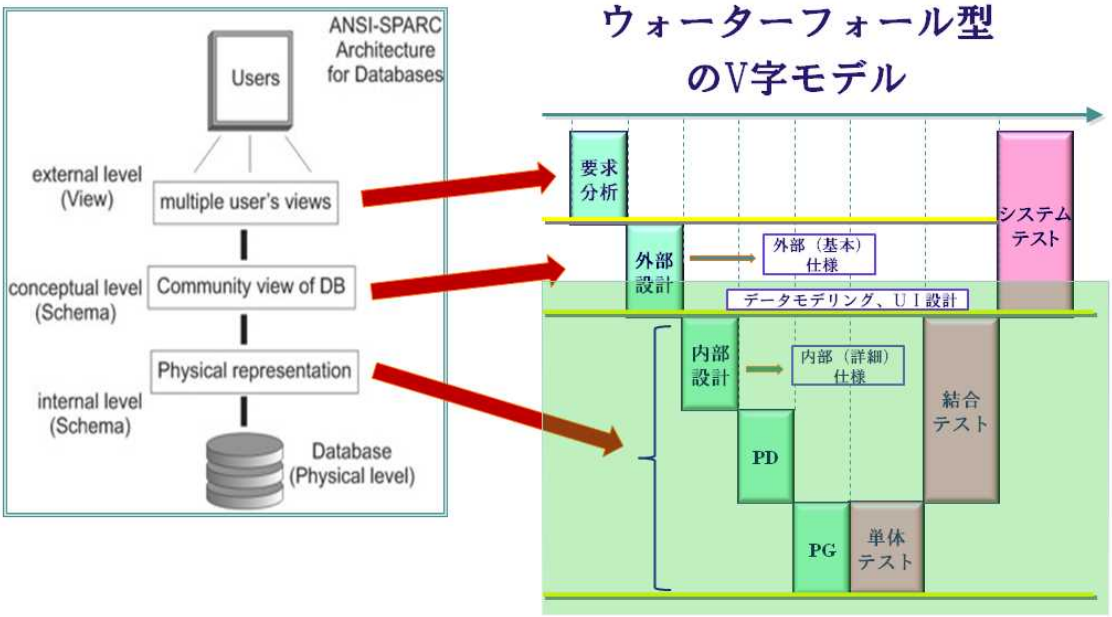
③ 内部レベル(internal level)：
Physical representation（物理表現）で、物理データベースのスキーマ情報（テーブルや項目等）のことです。

GeneXus は、「外部レベルの情報から、推論により概念（論理）レベルと内部レベルを自動的に生成する」という技術です。

では、なぜこの様な三層構造が提案されたのでしょうか。それは「**プログラムと物理データベースが密結合されていると、プログラムの改修がいずれ破綻する**」という危機感があったからです。つまり、プログラムは外部レベルの情報を使って書き、物理データベースとのマッピングは概念（論理）レベルの情報を使って行えば、物理データベースの構造が変化してもプログラムに影響を及ぼさなくて済むというアイデアです。

8．ウォーターフォール型開発工程との対比

この ANSI-SPARC の三層構造と、これまでのシステム開発で使用されているウォーターフォール型開発工程を対比させてみましょう。



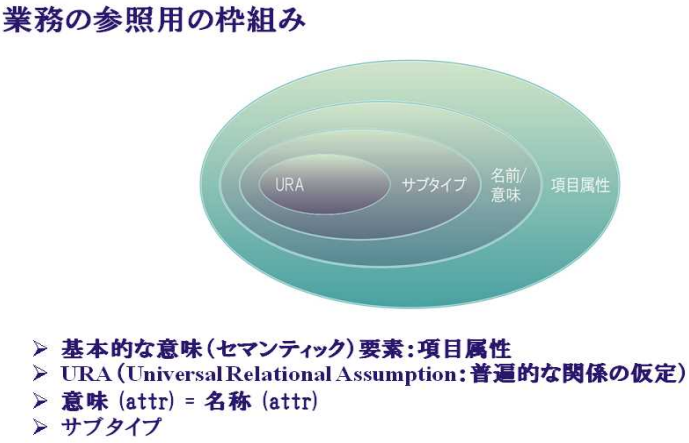
ウォーターフォール型開発工程で、外部レベル（外部モデル）に相当するのは要求分析で、概念レベル（概念／論理モデル）に相当するのは外部（基本）設計です。そして、内部レベル（内部モデル）に相当するのは内部（詳細）設計、プログラム設計、プログラミングの各工程です。

GeneXus は外部レベルの情報を使って、推論により概念（論理）レベルの設計仕様と内部レベルを自動的に生成することができる技術です。つまり、網掛けをした外部設計のデータモデリングやＵＩ設計以下の工程は不要となります。

9．業務知識の表現方法（GeneXus の理論の原点）

推論を正しく動かすためには、業務を参照する枠組みが必要となります。彼等はその枠組みの中心に項目属性（attribute）を置き、そして、この項目属性に基本的な意味（セマンティック）の要

素を持たせようとしました。

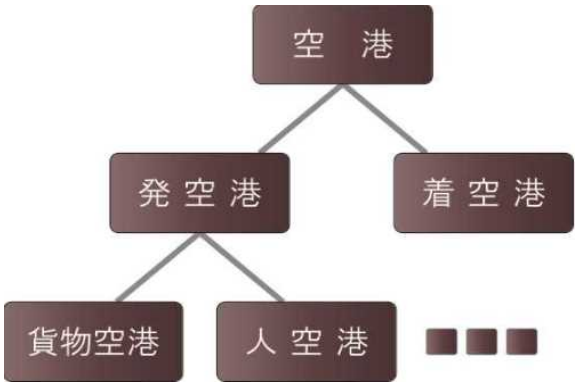


その中心にあるのが URA(Universal Relational Assumption)という仮定です。つまり、「一つの業務の世界には、同じ名称を持つ異なった内容のデータは存在しない」という仮定です。この様な仮定を置くと、データの意味は名称で表すことが可能となります。

しかし現実には、異なった部門で同じ内容のデータを違った名称で呼んでいる場合（synonym：異名同義語）があります。そこで、ここにサブタイプという考え方を付け加えます。つまり、異なった名称であっても、実体が同じであれば、サブタイプを設定することで、同じ項目属性とみなしてくれます。

逆に、異なった部門で異なった内容のデータを同じ名称で呼んでいる場合（homonym：同名異義語）もあります。この場合は名称を変更し、異なった項目属性として登録します。

１０．業務知識の表現方法（補足）



ここで、もう少し補足しましょう。

業務分析の過程で、右の図の様な構造が明らかになったとします。業務の観点から、「空港」は「発空港」と「着空港」に分かれ、しかも、「貨物空港」と「人空港」等に分かれるとします。これらは総て業務を遂行している担当者の視点から見えてくる構造と項目属性です。

GeneXus では、担当者の視点から見えてくるこれらの項目属性を URA (Universal Relational Assumption) として扱い、**GeneXus** に投入します。

しかし、更に業務を分析していく過程で、「空港」が「発空港」や「着空港」と同じであることが判明したとします。そこで、サブタイプを用い 「空港」が「発空港」や「着空港」と同じであることを示します。

GeneXus は、URA である「空港」、「発空港」、「着空港」、「貨物空港」、「人空港」等を使ってプログラムを自動生成し、サブタイプの指定に従って正規化されたデータベースを自動生成します。

１１．他ツールや言語との決定的な違い

システム開発の生産性向上を目的としたソースコード自動生成ツールが数多く存在します。それらの多くは「**ソフトウェア部品**」の考え方に基づいています。

現在は、ツールの開発環境を著しく進化させ、「**部品の組み合わせ技術**」の優劣を競い合っています。最新のツールは「**開発者にソフトウェア部品の存在を意識させない**」という方向に向かっている様にも見えます。

しかし、この「**ソフトウェア部品**」という方向性には、以下の様な幾つもの欠点が存在します。

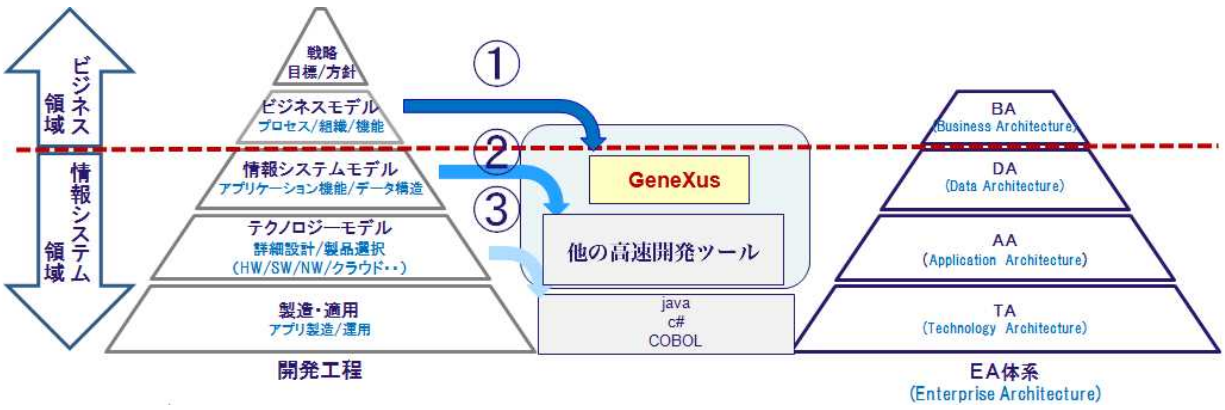
① 部品は実装技術の塊（開発当時の）
② 部品の充実と部品の保守が相反（幾何級数的にツールの保守工数が増加）
③ 実装技術に飛躍が起こると対応困難（例．新言語の出現、ディバイスの出現）
④ ソフトウェア部品は工学のレベルに達していない（製品の構造を過小評価）

ですから、現在の様な急激な IT 技術の進化の下では、いずれ破綻することは明白です。

一方、ソフトウェアの品質向上のために、「**数学的に厳密に意味付けられた言語を用いて情報システムの仕様や設計書を記述しよう**」という Formal Methods（形式仕様記述言語）を推進する流れも有ります。しかし、「一般の技術者が使うには敷居が高い」と言われています。

既にご説明した通り、GeneXus は、外部レベルの業務担当者によるデータの見方を解析し、概念レベルと内部レベルの情報を生成します。その際、GeneXus は数理論理学の述語論理による理論に基づいた Prolog による推論を行い、実装のための設計書を生成します。そして、この設計書は Formal Methods（形式仕様記述言語）によって記述されています。

GeneXus と他ツールを使い方の面で比較すると、大きな違いが有ります。ANSI-SPARC Architecture（三層構造）で説明した通り、GeneXus と他ツールとでは開発時に使用する情報に違いが有ります。



（参照）ジェネクサス・ソリューションズ・ジャパン(株)講演資料

・GeneXus　　：外部レベル（ビジネス領域）の情報（項目属性等）①

・他のツールや言語：内部レベル（情報システム領域）のスキーマ情報（テーブルや項目等）②

他のツールや言語でシステムを構築すると、データベースの構造が変化したり、項目が他のテーブルに移動したりする時は、再度設計からやり直す必要があります。

しかし、GeneXus では外部レベル（ビジネス領域）の情報（項目属性）①を使っているので、データベースの構造が変化しても、自動生成をやり直すだけで書き直す必要はありません。

１２．業務手続きを書けるオブジェクト（プロシジャー・オブジェクト）の追加

1989 年にゴンダ教授の理論に基づいた GeneXus の第１版が完成したとき、彼等の目標は「プログラムの７０％を自動生成し、保守（改善）できる様にする」ことでした。

残りの３０％は手作業でプログラムを書き、保守（改善）しなければなりませんでした。彼等も、当初は「プログラムの全体を自動的に保守（改善）できる」か否か半信半疑だったのです。

しかし、GeneXus を最初に利用したお客様は、開発生産性が著しく向上したことには満足されましたが、むしろ、「DB を自動的に保守（改善）できる」という点に GeneXus の価値を見出したのです。そして、次第にお客様からの「業務システムの全てを自動的に保守（改善）可能とする様にしてくれ」という要求に晒されることになりました。

そこで、彼等はプロジェクトチームを作り、研究を重ねました。そして、残り３０％の自動化を実現するためには、「業務手続きを書けるオブジェクトを追加するしかない」という結論に達しました。そして、GeneXus に外部レベルの項目属性を使って業務手続きを書ける機能を実現するために、プロシジャー・オブジェクトを加えたのです。

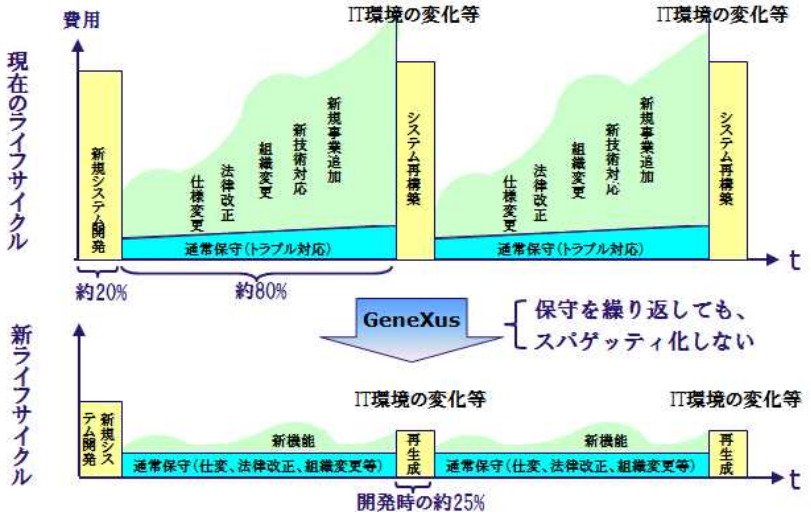
プロシジャー・オブジェクトに書く業務手続きには、実装に関わる記述は含まれていません。そこが物理ＤＢのスキーマ情報を使ってプログラムを書くことと決定的に違う点です。

１３．ユーザが見いだした価値

ここで、ユーザが GeneXus に見出した価値について少々補足しましょう。

ユーザには「完成したシステムを見て、始めて自分が本当に必要とする要件を理解する」という傾向があります。ですから、最初から完璧な仕様に基づいてシステムを構築することなど不可能で、仕様変更の山となります。また、システム完成後も社会の変化、法制度の変化、会社組織の変更、新たなビジネスモデル等にも対応していかねばなりません。しかも、ＩＴ技術の進化によって、５～７年毎に再構築を要求される場合もあります。

その様なシステム開発の終了後に必要となる費用を総て加味すると、開発に要する費用はライフサイクル全体の約 20%程度で、保守（改善）に関わる費用が約 80%と言われています。ですから、1 円入札をしても採算が取れるのです。ライフサイクル全体をベースに考えれば､僅か約 20%の値引きをしているにしかすぎません。



現在では、様々な開発ツールや開発技法も提案されていますが、それらはライフサイクル全体の約 20%を占めるシステム開発の効率化を狙っているだけで、「ビジネスモデルの変更」や「ＩＴ技術の変化」といったシステムの保守（改善）に関しては全く無力です。

ですから、全体の約 80%を占める保守（改善）の費用を削減できなければ、その効果は微々たるものです。**GeneXus** を使って開発すれば、この費用を大幅に削減することが可能となります。

１４．誰が業務知識を持っているのですか？

20～30 年前に開発された基幹業務システムには、長年の間に、法律改正、制度変更、組織変更、ビジネスモデルの変化等の理由で、多くの改修が加えられています。そして、時間に追われて改修作業を行ったため、プログラムの改修とテストに重点が置かれ、設計書の修正はお座なりでした。しかも、改修を繰り返すうちに、ソースコードはスパゲッティ化し、判読が困難になりました。

その当時のシステムを開発した技術者やユーザの多くは定年を迎えたり、別の会社に転職したり、別の部門に異動になっています。ですから、今ではプログラムの改修が行われた理由さえも分からなくなっているケースが多いのです。

そして、そのシステムを使って仕事をしている現場の担当者からも業務知識は薄れてきました。長年システムを使っているうちに、システムの利用知識が担当者の業務知識となり、ＩＴ技術者に説明するために必要な「業務仕様」さえも書けなくなっているのです。これでは、システムを再構築したくても難しいでしょう。

１５．各ユーザのビューから全体が見える

元々、組織の中に業務全体を理解しているスーパーマンなどは存在しません。でも、個々の業務担当者は日々システムにデータを入力し、出力された帳票を使って業務を遂行しています。

ですから、自分達が扱っている情報のことは良く知っています。たとえ業務全体を理解し、仕様を書けるスーパーマンがいなくても、個々の業務担当者が持っている業務知識を集めれば、全体像が見えてくるのではありませんか。

スーパーマンが書く様な業務仕様が無くても、要件分析の段階から **GeneXus** を利用し、各業務担当者の方々から得られた断片的な業務知識を投入していけば、業務の全体像を作り出すことは可能です。

後は、この方法で新たに構築したシステムと元のレガシーシステムを並行ランし、違いが発生した際に、その原因を調査し修正を加えていけば良いのです。だから、レガシーシステムの再構築が困難だと、悩む必要はありません。時間を掛けて取り組めば、必ず再構築は可能なのです。

１６．GeneXus とは

「**システム開発**」という言葉の中に「開発が終われば、仕事は終了。後は、勝手にしてください」というニアンスを感じるのは筆者だけでしょうか。
ですから、**GeneXus** を説明するのに「**開発ツール**」という表現は使いたくはありません。

GeneXus によって構築された業務システムは、業務の変化、社会制度の変化、**IT** 技術の進化等が発生しても、未来にわたって保守（改善）を続けていくことが可能です。

つまり、**GeneXus** の目標は「**貴社のビジネスの未来を守る**」ことなのです。

参照：” KNOWLEDGE-BASED DEVELOPMENT (PHILOSOPHY AND THEORETICAL FOUNDATION OF GENEXUS)”
BY BREOGÁN GONDA AND NICOLÁS JODAL, MAY 2007

謝辞：この資料作成にあたり株式会社ジェネクサス・ソリューションズ・ジャパンの佐々木敏一氏、内山東平氏、四居雅章氏の助言を頂戴しました。厚く御礼申し上げます。

GeneXus™

ジェネクサス・ジャパン株式会社
〒141-0031
東京都品川区西五反田 2 丁目 27 番 3 号
Tel. (03)6303-9381　　Fax. (03)6303-9980