貴方の組織の未来を守る

(*Futureproofing Your Organization*)

Ken Orr

September 2005

（2015 年 3 月　加筆修正版）

Copyright © ARTech Consultores S. R. L. 1988-2007.

Todos los derechos reservados. Este documento no puede ser reproducido en cualquier medio sin el consentimiento explícito de ARTech Consultores S.R.L. La información contenida en este documento es para uso personal únicamente.

Marcas Registradas

ARTech y GeneXus son marcas o marcas registradas de ARTech Consultores S.R.L. Todas las demás marcas mencionadas en este documento son propiedad de sus respectivos dueños.

もし，自分の過去を知りたいのなら，現在の状況を見なさい。

もし，自分の将来を知りたいのなら，現在の行動を振り返りなさい。

仏教の教え

未来を守る事と変化

Futureproofing（未来を守る）, 定義：“今日の決定が，将来でも効果が有ることを保証する行為”

“Futureproofing” – なんと素晴らしい響きを持ったコンセプトでしょう。もし，今日の絶え間ない変化の波の下で為されるビジネスや技術分野での意志決定が，未来にわたって効果を生み出すとするならば，素晴らしいことではありませんか。もし，ビジネスや技術が変化しても，直ぐに，そして，簡単にそれらに適応可能な柔軟なシステムを構築できるとすると，どうでしょうか。もし，現実に未来の変化から貴方のビジネスを守る（Futureproof）ことができるとすると，どうでしょうか。多分、そんなことは想像するだけでも無理かもしれませんね。しかし，この論文は，既に証明されているソフトウェア開発と保守の技術を使って，貴方の組織の未来を守ることができる，という点について述べています。

未来を守るというアイデアを考えてみましょう。未来を守るということは，現実に発生する変化とその変化への対処方法を理解するということに尽きます。この現実の世界では，明らかに変化はビジネス（ビジネスが公共か民間か，大きいか小さいかにかかわらず）に於ける最も重要な要素です。どの様な分野のビジネスであろうとも，総てのビジネスは今日発生する変化を切り抜けていかねばなりません。そして，責任のレベルにかかわらず，総てのマネージャは加速度を上げている変化に対処するために，重要な決定を下さねばなりません。そして，IT の分野ほどその影響を受けているところはありません。

毎週，そして，ほとんど毎日，新しい装置やソフトウェアが市場に出現します。それらは新たに重要なビジネス機会を作り出すと同時に，既存のディバイスやソフトウェアへの巨額な投資を時代遅れなものにしてしまいます。ほとんど毎週，ビジネスの情景を一変させる様なシステムの開発が行われています。まさに，今後数年間にわたる多くの組織の成功は，この論文に書いている通り，変化をマネージする彼らの能力（組織の未来を守る）に強く依存することになるでしょう。

メガトレンド：これら総ての変化はどこから来るのでしょうか。

しかし，変化は，それがたとえ大きな変化（Megatrend）だとしても，決して希に起こるものではありません。前の世紀を振り返るだけでも，幾つもの例を挙げることができます。例えば，ラジオ，飛行機，テレビ，核エネルギー，集積回路，宇宙旅行，遺伝子工学，そして特に，コンピュータです。２０世紀初頭では，多くの旅人は自らの足や馬の背に乗って旅をしていました。２０世紀の終わる頃には，彼らは高速な自動車や飛行機で旅行をしています。当時，電話機は希にしか見られませんでしたが，２０世紀の終わる頃には，携帯電話が財布や鞄と同じように一般化していました。

しかし，２１世紀に起こっている変化の種類には新しい局面が見られます。つまり，世の中の速い動きを後押しする幾つものメガトレンドがあり，そして，最も日常的な意志決定にさえ，より多くのリスクが含まれています。これらのメガトレンドの中で最も重要なものは，以下の通りです。

- メガトレンド 1:　歴史の流れが加速しています。
- メガトレンド 2:　未来はより一層予測困難となっています。
- メガトレンド 3:　未来は現在の延長線上にはありません。

メガトレンド１：歴史の流れが加速しています

ビジネスと技術がもたらす圧力は，今日の組織と人々により多くのストレスを蓄積させる様になっています。今日では，普通のビジネス人でさえ，携帯電話，スマートディバイス*[1]，そして，ノートパソコンを携帯しています。５０年前，電話は壁に掛けるものでした。一握りの最も裕福な人達だけが，彼らの車やボートの上に無線電話を据え付けることができました。今日，裕福な人々と同様に，より多くの人々や働く人達が携帯電話を持つことができます。毎日の様に，地球上のほとんどの場所からの交信を可能にする，より多くの装置が発表されています。

（*1：＜訳者修正＞原文では PDA と書かれていましたが、スマートディバイスと変えています。）

同様に，ソフトウェアの世界でも変化が加速しています。各種プラットホームや言語の寿命は加速度を増して短くなっています。２０年前*[2]には，Java について考えている人は誰もいませんでした。

１５年前*3 には，.NET について考えている人は誰もいませんでした。そして，10 年前*[4]には，ほとんどの人達がリナックスについて考えていませんでした。

（*2：＜訳者注＞Java が最初に公開されたのは 1995 年 5 月 23 日の SunWorld カンファレンス）

（*3：＜訳者注＞2001 年に Microsoft は.Net を公開）

（*4：＜訳者注＞Linus Benedict Torvalds の Linux に大手メーカーが興味を示しだしたのは 2000 年から）

ハードウェアの世界では，変化は更に加速しています。５年前[*5]には，誰も iPhone のことを知りませんでした。そして，今日では誰でも iPhone[*6]を所有しています。今日，HDD やメモリ，そして CPU はとても小さく，そして，安価になり，どんな装置にも入れることが可能になりました。新しい装置を各世代毎に見ると，統合化が進んで小型化し，費用は一桁安くなりました。

（*5：＜訳者注＞iPhone が発表されたのは 2010 年）

（*6：＜訳者修正＞原文では iPod と書かれていましたが、現在では iPhone、iPad が主流なので原文を修正）

しかし，人はビジネスや技術の早い変化についていくことができません。結果として，組織を元のままに継続させることは困難となります。４０年前[*7]，IT のマネージャは，意志決定を下してから，その良い結果が現れるまでに，最低５年から１０年間を想定していました。今日では，最も有望な新技術やソフトウェアツール，又は，プラットホームでさえ，その寿命は極端に短くなっています。

（*7：＜訳者修正＞原文では 30 年前でしたが、現時点からは 40 年前になります。）

メガトレンド２：未来はより一層予測困難となっています

予測は難しいものです，特に未来は....。 – ニールズ・ボーア

21 世紀の始まった直後から，巨大な変化を目にしました。例えば，IT バブルの終焉，9 月 11 日の世界貿易センターやペンタゴンへの攻撃，インドと中国経済の爆発的な成長，極東地域への仕事や製造のアウトソーシングの増加，アフガニスタンとイラクでの戦争，数時間の間に２５万人もの人命を奪い去ったインド洋での地震と津波。

最も明敏なアナリストや歴史家だけは，過去５年間に起こった出来事を予測できたかもしれません。そして，最も無鉄砲な人達だけが，今後５年間に起こること，まして今後２０年間に起こることを予測したいと思うかもしれません。

歴史的にも，未来学者や予測者達は自らの予測のリスクや間違う確率を説明してきました。今日では、未来の予測を専門とする人達は，今日の世界の変化があまりにも速く，これまでのやり方では未来を

予測できなくなったので，いかなる期間のどの様なレベルの確実性に対しても、説明を求められることに嫌気がさしています。

メガトレンド３：未来は現在の延長線上にはありません

また，未来を予測することで生計を立てている多くの人達の使う重要な手法の一つは，現在の延長で未来を予測する方法です。彼らは，多くのトレンドが現状からの直線的な延長線上に存在することを知っています。しかし，未来はもう直線的な延長線上にはありません。ムーアの法則として知られ，歴史的に高い信頼を置かれていた法則でさえ，その効力に陰りが見え始めています。７０年代の半ばから 21 世紀前半に至るまで，パソコンとその背後に隠れている総ての技術は，ムーアの法則という巨大な波に従って技術革新と拡大を続けてきました。しかし，パソコンは最早世界を牽引する技術ではありません。実際，パソコンの世界は現状を維持するのに苦戦している様にさえ見えます。ですから，ムーアの法則には陰りが見え始めているのかもしれません。

世界は不連続の時代を経験しようとしています。２０世紀後半は，パソコンとインターネットがエンジンとして時代を引っ張りました。現在は他の何か，多分スマートディバイス，個人的なエンターテインメント機器，無線通信が世界を牽引しているのでしょう。パソコンの時代に未来を予測しようとした人は，数百万台の装置を対象としているだけでしたが，スマートディバイスの時代に未来を予測しようとする人は，数十億台の装置を対象としているのです。当然のことながら，組織も人も，この様な規模の変化に順応していくのは大変な苦労です。

この総ての変化に対応するために，ソフトウェアの分野で組織が必要とするものは新しい世代のツールです。より正確に言えば，２１世紀に発生する変化に追随していくための，ビジネス・アプリケーションを開発し保守するための新しい方法であり，この方法をサポートしてくれる一群のツールです。この論文の後半では，この様なことを可能とするために，どんな種類の技術が必要となるのか，という点について説明します。

ソフトウェアシステムの未来を守る事に関する問題

「未来を守るために重要な解決策とは何か」を議論する前に，新たに注目しておくべき幾つかの重要な事柄を考えておく必要があります。

- 新たに要求されるアプリケーションや機能の必要性は，それらを提供する側の能力を遙かに超えています。
- 次世代のアプリケーションを構築するには，全く新しい能力が必要となります。
- 技術的問題の解決はあまり気にせず，新たなビジネスの問題やビジネスチャンスへの対応に一層フォーカスするには，安定した環境を作り出す必要があります。

新たなアプリケーションや機能の必要性は，それらを提供する側の能力を遙かに超えています。

これまで，変化について述べてきました。この章では，極めて順応性の高いシステム（*hyper-adaptable systems*）の面から変化について論じなければなりません。これらの極めて順応性の高いシステムは下記に対処できるものでなければなりません。

- ビジネスニーズの変化
- 利用可能な技術の変化
- より複雑さを増す開発環境
- 熟練した人材の不足
- ベビーブーム世代の定年退職による大量の経験豊富な人材の流動化

IT バブルの崩壊以来，特に，新システム開発の分野で，非常に多くの組織が IT 関連費用の削減を行っています。多くの組織は以下の様な傾向を強めています：(1) IT 運用の部門も含めたアプリケーション開発部門のアウトソーシング化，(2) レガシーシステムの一部を置き換える，巨大で統合されたパッケージの購入，(3) メインフレームの基幹アプリケーションにウェブを使ったユーザ機能を開発。これらの傾向はまさしく変化に対応するためです。

２０１０年[*8]のビジネス業界は，１０年前や５年前に比べてより一層競争が激化しています。世界はより一層統合され，相互に影響し合う様になり，技術動向に敏感になっています。同様に，多くの組織が日々の活動で依存しているシステムはますます古くなり，そして，それらのシステムを開発してきた人々（もし，彼らがまだ周囲に居るとしても）の引退時期は，より間近に迫っています。

（*8：＜訳者修正＞原文では 2005 年でしたが、2010 年に修正しました。）

今後１０年間で，ほとんどの巨大組織はその中核を支える基幹アプリケーションシステム（中核アプリケーション）を置き換えるか，再構築しなければならないでしょう。そして，多くの場合，それらの組織が最も必要とするアプリケーションを市場から購入することはできず，自ら開発するしかないでしょう。しかも，できるだけ短期間で開発しなければなりません。

次世代のアプリケーションを構築するには，全く新しい能力が必要となります。

極めて順応性の高い次世代のアプリケーションが必要とする要素とはどんなものでしょうか。さて，これらのアプリケーションを一種類のプラットホームから他のプラットホームに直ぐに移植できる，ということの重要性が増してくるでしょう。「オープンシステム」という甘い言葉がありますが，ほとんどのアプリケーションは，それが今日開発されているものでさえ，たった一種類のプラットホーム，言語，データベースシステム用にしか開発されていません。システム開発に関わる費用と関連事項は、数年前に誰もが想像したものより更に急速に変化しています。例えば，今日では，競合する開発プラットホームは Java と.NET ではなくて，少なくとも Java と.NET と Linux となっています。近い将来に，幾つのプラットホーム[*9]が現れてくるのでしょうか。誰にも分かりません。

（*9：＜訳者注＞現在では Android が急速にシェアを伸ばしています。）

そして，ほとんどの組織ではデスクトップやノートパソコンをインターネットに接続して作業を行う，というシステム開発方法が一般的になりましたが，更に，狭い帯域幅で接続するワイヤレス端末の小さい画面を使った開発は新しい挑戦となるでしょう。常時接続されていない端末を使って作業する必要のあるアプリケーションの開発もその例でしょう。次に挑戦するものが何であるかを，誰が知っているでしょうか。多分，RFID でしょうし，GPS の可能な装置もそうでしょうし，上記の組み合わせの総てかもしれません。現在選択可能な装置の中から選ぼうとしている間にも，そのタイプや数が増加し続けている事は良く分かっています。そして，新たな装置はどれでも新しい挑戦を行っています。一つの目的のために開発された装置が改造され，別の用途でも使われることになります。これまでに誰も考え付かなかった装置が開発されると，誰かにサポートしてもらわなくてはなりません。

ですから，次世代の IT アプリケーションは，或るプラットホームや言語やハードウェアから，別のものにより簡単に移植できなければなりません。そして，違う DBMS（データベース管理システム）をサポートできなければなりません。プログラマの指摘通り、それは簡単な話ではありません。主なリレーショナルデータベースはとても似ており，今日のデータベース構築技法も似た様なものですが，リレーショナルデータベース同士の変換でさえ，無視できない作業量になるほどの違いがあります。そして，言語の変換も同様に多くの違いがあります。Java と C#は非常に似た言語ですが，再度一から書き直すのは簡単ではありません。

最後に，次世代のアプリケーションは以下の様なものでなければなりません。つまり，(1)より簡単なコミュニケーション，(2)より高度なフォールトトレラント，(3)桁違いの安全性。コンピュータが，よ

り一層日常生活の中で主要な位置を占める様になると，システムはコンピュータ・オタク向けではなく，素人のお婆さんやお爺さんが使える様に設計されなければならないでしょう。アップル・コンピューターがハイテクの世界で競争相手として生き残り，復活を果たした意味は決して小さくはありません。これは，アップルが使いやすさを実現するために，常にヒューマンインターフェースの設計に注意を払ってきた結果なのです。

技術的問題の解決はあまり気にせず，新たなビジネスチャンスやビジネス上の課題に対処することに、より多くの経営資源を集中させるには，安定した環境を作り出す必要があります。

もし，組織の未来を守る（futureproofing）ための解決策を見付けようとするならば，ビジネス上の問題を解決するための最良の解決策を見付けるために，大半の時間をユーザとのやり取りに使える様なアプローチ方法に注力しなければならないでしょう。システム開発に際し、今日存在している膨大な量の技術基盤をどの様に利用するか、という問題を解決することに，現在では余りにも多くの努力が費やされています。

将来成功を収めるためには，技術の問題にほとんど注力することなく，代わりに，ビジネスの問題を理解し扱うことに開発の努力の大半を振り向ける様な，解決策を提案しなければなりません。

未来を守るための戦略

もっとも，これまでは組織の未来を守るという話題についてだけ着目してきました。しかし，現在の組織の未来を守る話題だけで，別の大切な話題については見てきませんでした。コンピュータの時代が始まって以来，明らかに人々は「変化の問題」を解決しようと試みてきました。実用的なソフトウェアの開発自体は挑戦的なものですが，本来は意図されていなかった保守や変更への挑戦の方がより大きな問題なのです。この章では，問題に対する長短期の解決策を見ていきましょう。

短期的な解決策

今日，ビジネスの世界では，様々な変化に組織が対応し効果を生み出せる様に支援することを目的とした，一般的な解決策があります。最も良く使われているものは以下の通りです。

- 商用パッケージ
- アウトソーシング
- アジャイル開発
- ソフトウェア部品の再利用

商用パッケージ

「ソフトウェア開発は自分達の仕事ではない」と考える組織の数が，世界中でますます増えています。実務的な言い方をすれば，これはソフトウェアは開発するよりもむしろ購入する，ということを意味しています。自社で非常に高額な人件費を払ってソフトウェアの専門家を雇用し続けるよりも，多くの会社は業者から商用ソフトウェアパッケージを買う選択をしました。こうすることで，ソフトウェアを最新に保つという重荷を業者任せにできます。技術の変化、ビジネスルールの変更，そして，新規の改革は、パケージ費用の一部として提供されます。もちろん，パッケージ自体の保守も含みます。パッケージソフトを購入するということは，組織内に訓練されたアナリストやデザイナー，そして，多くの開発者を抱える必要がない，ということを意味します。そして，組織が心配しなければならないのは，どのパッケージを購入すべきかを判断し，そして，実際に購入することだけです。

問題はパッケージの選択は良い面ばかりではないということです。つまり，巨大な組織が持つ独創的な総てのニーズに，完全に合致するパッケージなど存在しません。ですから，パッケージを購入しようとする会社は，次のようなジレンマに陥ることになります。(1) 組織のユニークなニーズに合致する様にパッケージをカスタマイズする。(2) パッケージは元のままにしておいて，パッケージに合うように組織を変更する。

どちらのアプローチも悲惨な問題を引き起こします。パッケージソフトの初期の段階では、組織は自社のビジネスに合致する様にカスタマイズする傾向がありました。しかし，この様にしてしまうと，パッケージのバージョンアップ時には深刻な問題が発生することになります。困難さを増すバージョンアップによる変更を２～３回経験すると，これらの組織はできるだけカスタマイズを最小限に止め，パッケージに合わせて組織を変更する戦略に移っていくでしょう。しかし，この取り組みも旨く行かず，元のカスタマイズに戻っていかねばならないでしょう。歴史は，振り子が前後に動き続ける様に、方針が揺れ動くことを示しています。

しかし，パッケージにはカスタマイズの問題よりも更に悪い一面があります。時間が経つにつれ，組織が特定の業者と結婚させられていることに気付くでしょう。どんなものであれ，基幹業務のパッケージに委ねてしまうと，８～１０年以内にそのパッケージから手を引くことは非常に困難となります。そして現実に，Peoplesoft が J.D.Edwards を買収し，Oracle が Peoplesoft を買収するといった劇的な変

化が，競合するパッケージ業者の間で発生しました。絶えず，ソフトウェア会社は倒産したり，吸収されたりしています。そして，突然気が付くと，以前は明らかに選択肢にも挙がっていなかったソフトウェア業者と結婚させられているかもしれません。

アウトソーシング

ある組織がこれらソフトウェアの変更に関する問題を取り扱おうとする場合，別の解決策は外部の組織にソフトウェアに関する事柄を委託してしまうことです。最近数年間の内に，アウトソーシングはソフトウェアとソフトウェアの変化に対処するための一般的な方法になりました。自社にソフトウェアの専門要員を抱えるよりも，なぜソフトウェアを専門とする業者に業務を委託しないのか，という議論が広がっています。業界紙の紙面は，ソフトウェア開発やコンピュータの運用機能をアウトソーシングする巨大な組織の発表記事で一杯です。

通常，ソフトウェアパッケージの購入と同様に，アウトソーシングを選択する場合も長期間の委託をしなければなりません。多くの場合，ＩＴの最高責任者や専門家を除いて，短時間の内に社内のスタッフはアウトソーシング会社に転籍となります。これまでの関係を保っていくという前提はありますが，現在のアナリスト，デザイナーや開発者と業務を担当するユーザとの間のやり取りは，より正式なビジネスライクなものとなり，しかも，よそよそしくなり，当然、高価なものとなります。そして，パッケージを購入する場合と同様に，アウトソーシングには明らかな欠点があります。

アウトソーシングをすると，しばしば業務担当者とソフトウェア開発担当者の間に壁ができてしまいます。しばらく経つと，かつて顧客の組織で働いていた担当者は他に移動したり，退職していきます。そして，アウトソーシングされた次の世代のソフトウェア構築担当者になると，顧客のビジネスに従属している意識が少なくなり，関心も薄れていきます。さらに，アウトソーシングを引き受ける会社には，しばしば基本ソフトウェアやプラットホームの変更を最小にするという特権があります。従って，大きな変更要求が発生すると，しばしばユーザは別途独自で新規開発をしたり、別のパケージの利用を余儀なくされることもあり，アウトソーシングの構造自体を脅かすことになります。

アウトソーシングを選択した場合に遭遇する最も深刻な問題は，ビジネスに携わる一部の中核となる人達から，業務知識が失われていくことです。つまり，総て，又は，ほとんどのソフトウェア開発をアウトソースした組織からは，結果として，アウトソーシングを引き受けた会社との関係を管理する知識を持つ人さえ失われていくのです。もっと長い時間が過ぎると，通常，アウトソーシングを引き受けた会社の力がますます強くなり，依頼した組織との関係に摩擦を生じていきます。アウトソーシングを引き受けた会社と以前の関係を取り戻し，管理しようとする組織は，しばしばアウトソーシン

グを止めるために，組織の構造を再度作り直さなければなりません。この処理には通常何年も，時としては何十年もかかります。

再利用：オブジェクト，ソフトウェア部品，そしてサービス

パッケージ，又は，アウトソーシングの導入は，経費の引き下げや，変化に晒される度合いを下げようと試みる組織が，近年採用している主な戦略です。これらの戦略では，基本的に何処か他の会社にソフトウェアの開発と保守を任せることになります。しかし，多くの場合，多量のソフトウェアは内部で開発しなければなりません。しかし，幾つかのプロジェクトは，外部に委ねるには戦略的過ぎたり，時間が無さ過ぎたり，基幹に関わる箇所だったりします。

まだソフトウェア開発を行っている組織（非常に貴重な存在ですが）にとって，ここ１０年から１５年間は，ソフトウェアの再利用が主な話題でした。ソフトウェアは，とても高価で，常に多くの誤りを含んでいる傾向が強いので，ソフトウェアについて研究している多くの人達は，他の製品を形成しているものと違って，未だに，ソフトウェアは「再利用可能な部品」というエンジニアリング上の学問のレベルには達していない，と信じています。再利用の概念に基づき，「オブジェクト指向革命」が幅広く販売されていました。オブジェクトは，多くの異なったプログラムで簡単に再利用できる様な方法を実現することを意図して設計されました。

ここ１０～１５年間，最初はオブジェクト，次に部品，そしてサービスと，全体的な範囲で再利用可能な解決策が現れてきました。現在，「部品に基づく開発」と呼ばれているものの背景にある考え方です。もし，正しい方法で部品が設計されていたら，部品の市場が形成されていたかもしれません。他の分野のエンジニアの様に，開発者は最初からプログラムを開発するよりも，むしろ，カタログやインターネットから標準の部品を選んで抽出し，新しいプログラムを作成する際には，単純にそれらを「組み立てる」だけでいい，という考え方です。

この再利用の話題に関する幾つかの箇所は，確かに機能していました。オブジェクトと部品は，今日様々な分野のプログラミング・コミュニティで広く利用されています。例えば，グラフィカル・ユーザー・インターフェース(GUI)の設計で，オブジェクトは広く利用されています。GUI の開発は従来から複雑で，作成は簡単ではありませんでした。そして，オブジェクトの利用は複雑なインタフェースを造る自然なやり方となっています。別の領域としては，数をグラフや図形に変換する必要のある，プレゼンテーションのアプリケーションでもオブジェクトを利用しています。しかし，GUI やプレゼンテーションのロジックと同様な方法で，ビジネスに関する領域で，「ビジネスオブジェクト」や部品が著しく進歩し，成功したという話はほとんどありません。

再利用可能な部品の背後には，変わらないプログラムの本質部分があります。最初は，ソフトウェアのオブジェクトは自動車や冷蔵庫の部品と同じ様なものだと想定していましたが，実際にはそうではないことが分かってきました。一つには，ソフトウェアの部品は，ほとんどの物理的な部品よりも遙かに複雑であることが判明しています。更には，オブジェクトや部品を設計する人は，重要な要素（製品の設計段階で，製品の構造という方程式）を含めるのを忘れていました。

現実の世界では，製品を作る技術者は，製品の全体的な構造を設計するのに膨大な時間を費やします。最も重要な設計図では，製品の全体的な構造と，どの箇所で各要素の部品が使われ，それが組み合わされるかが示されています。残念ながら、ソフトウェアの設計者は製品の構造を省いてしまいました(または，重要性を過小評価してしまいました)。結果として，ほとんどの直ぐに入手できる部品は静的な部品となっています。この見落としの結果の一つとして，設計者は静的なオブジェクトが処理できる使い方を総て考えておかねばならないので，最終的には，その部品は本来の姿よりはるかに複雑となります。

ソフトウェア部品の再利用に関して，これまでの仮説の結果として，オブジェクト指向のソフトウェアを開発するのは，以前想定されたよりもはるかに複雑であり，高価につくことがわかりました。更に，この技術で開発されたソフトウェアは，保守が困難（そして，高価）なのです。全く腹立たしい問題ですが，最終的にアプリケーションの保守と更新にソフトウェア費用の半分以上が費やされているのです。

広い意味で，製造工学を見ることから，学べることは沢山あります。ここ４０年か５０年[*10]の間，コンピュータとコンピュータのソフトウェアを，製図や設計，プロトタイプ，そして，製品の製造に利用することで，エンジニアリングのすべての分野が大きく成長しました。コンピュータの支援による設計とドラフト作成(CAD)，コンピュータの支援によるエンジニアリング(CAE)，コンピュータの支援によるプロトタイピング(CAP)，そして、コンピュータの支援による製造(CAM)はエンジニアリングの分野で広く使われています。

（*10：＜訳者修正＞原文では「30 年か 40 年の間」ですが、これを「40 年か 50 年の間」に修正しました。）

今日では，技術者は概念的な図面から部品のプロトタイプまで、数時間の内に移行することが可能です。CAE ソフトウェアを使用すれば，CAD 図面を分析してシミュレートし，問題を特定し，変更することができます。そして、CAP か CAM ソフトウェアを使用すれば，新しい部品を作ることができます。更に，高速なグラフィックコンピュータに，デジタル化された総てのものを仮想製品のメタデータとして格納できます。

航空宇宙，電化製品，建築物のディジタル化された製品や部品はコンピュータに格納され，他のソフトウェアとは違って，即座に，再構成可能で，最終完成品を作り出すことができます。残念ながら，ほとんどの最も先進的なソフトウェア開発業者でも，ソフトウェアの開発は，未だに莫大な人手の介入を必要としています。ソフトウェアの開発者は，ソフトウェアというものはコンピュータで自動生成できないくらいに複雑である，と主張しますが，エンジニアリングの世界でその様な主張は通用しません。*もし，組織が自らの未来を守ろうとするなら，他の -柔軟な要求-設計-生成-プロトタイピング- といったより高度なエンジニアリングの形態に基づいた，立証された技術に頼る様に変わらなければなりません。*

アジャイル開発

最後にもう一つ，より短期間でソフトウェアの変更の問題に対応しようとする『アジャイル開発』と呼ばれるソリューションがあります。ソフトウェアの歴史の大部分を占める支配的な開発方法は，開発プロジェクトの工程を（計画，要件分析，設計，製造，そして，導入）といった段階に区分する『ウォーターフォール型』の開発方法です。各工程は，次の工程が始まる前に必ず終了します。このやり方は，第二次世界大戦とその直後に大規模なハードウェア製品を製造するための戦略から引き継がれたものです。このやり方の非常に強力な点は，各工程毎に特化することで，担当者が専門化し，理解しやすいことです。

しかし，このウォーターフォールの戦略を実際に適用しようとすると，それ自体に多くの問題を持つ傾向があります。ソフトウェアは成熟した分野ではないので，仕様作成，設計，プログラムテスト，データベースや特にビジネスルールなどに，合意された統一規格は全くありません。非常に大きなプロジェクトでは，各主要フェーズに数ヶ月や数年も要するものがあり，結果として，出来上がった時には，既に時代遅れになってしまっています。プロジェクトが大きくなればなる程，その様な失敗は起こり易くなるでしょう。

ユーザが現在の小さな単位の開発分（増分）を評価すると，彼らのアイデアはより洗練化し，次の部分の要件定義に反映できます。開発担当者は新しい機能を追加することで製品の再設計をすることができ，そして，以前の要件の塊に対する誤りを修正し，変更を加えます。それから，開発担当者は，短期間の内に再度新たに付け加えた部分をユーザに提示して評価してもらいます。この開発工程全体は『アジャイル開発』として知られるようになりました。この開発方法を採用した組織は驚くべき利点を報告しています。しかし，他のものと同様に，アジャイル開発も良い面だけではありません。

アジャイル開発が開発担当者とエンドユーザの間で一般的になってきた理由の一つは，直ちに結果を評価してもらえるということです。要求と応答の間に時間差が無く，無駄な努力は必要ありません。要件は直接コードに反映されます。しかしながら，ソフトウェア開発をしたことの有る人なら誰でも，複雑なソフトウェアを開発するには，しばしば詳細な分析や設計が必要となることを知っています。

アジャイル開発の良くない面は，設計書の不足と，時間が経過するにつれて保守費用が嵩むことです。アジャイル開発の黄金律の一つは，開発プロジェクト・チームはどこでも可能な限り紙の設計書の作成を避けるべきである，ということです。多くのエクストリーム・アジャイル開発者は，「コードだけが本当に必要な唯一の設計書である」とさえ信じています。さて，この様な発言は，ソフトウェア開発の歴史において，幾度もハッカー達によって行われましたが，現実には決して受け入れられるものではありませんでした。保守を行うためには，特に，オリジナルのソフトウェア開発に携わらなかった人達が保守を担当する場合には，常に良く書かれた設計書が要求されます。貧弱な設計書というのは，保守の難しい（つまり，費用のかかる）ソフトウェアと同じことです。

しかし，アジャイル開発の考え方には同意できるものも沢山あります。例えば，インクリメンタル（漸増）開発のアイデアは素晴らしいものです。歴史的に見て，ソフトウェア開発での主な問題点の一つは，現在の要件を反映しようと試みた結果が，将来の要件とは大きくかけ離れてしまうことがあるということです。実際，完全なオブジェクトを設計したい，という願望を強く持っているオブジェクト指向設計者が直面した中で，最も重要な問題の１つになっています。ですから，インクリメンタル（漸増）開発は良いアイデアでしょう。

そして，ユーザに密着し，プロトタイプを通して開発を進めるアイデアは素晴らしいものです。経験では，ユーザの要求を実際のプログラムに素早く出来ればできるほど，良い結果となることは分かっています。大半のユーザにとっては，抽象的に書かれた画面の図面よりも，プロトタイプの様に実際に触れることの出来るものの方が良いのです。

そして，プログラムの再設計，つまり各増分毎に実際の再設計を行うアイデアも素晴らしいものです。アジャイル開発の世界では，これは『リファクタリング』として知られています。これは最高に洗練された考え方に見えますが，単純なリファクタリングは，新しい要件を確認するために，データベース（オブジェクトクラス）やコードの総てを見直すことを意味し，変更点は「追加」というよりも，むしろ「組込」されることになります。リファクタリングの持つ唯一の問題は，特に，手作業でリファクタリングを行った場合，システムやアプリケーションが大きくなればなるほど，対応に時間が掛かるということです。ですから，プログラムやアプリケーションが大きくなればなるほど，新しい各

リファクタリングに要する工数が増えていくことです。これは，開発の繰り返しにより多くの時間が掛かるようになることを意味し，リファクタリング自体の持つ意味が失われてしまいます。

短期的な解決策から学ぶこと

1980 年代の中頃には， Computer-Aided Software Engineering (CASE)ツールを開発して，「ソフトウェア開発を自動化」しようとする大きな流れが有りました。残念なことに，当時の技術水準はこれを実現できるほどには進んでいませんでした。当時のソフト産業が全く理解していなかったために，発明し，統合しなければならない多くのものが単に残されただけでした。しかし，ソフトウェア開発を自動化しようとする目標自体は正しかったのです。言い換えれば，製造エンジニアや建築エンジニアが製品やビルを定義するのと同様に，ソフトウェアの定義を行うのに，数週間や数ヶ月ではなく，数分や数秒間で結果を得られる様にしようとするものでした。実際，この実現可能性を考えてみると，ソフトウェア製品は当初から本質的にデジタルなものなので，これを実現するには理想的な領域であることがわかります。

本当に我々の組織の未来を守ろうとするには，ソフトウェア開発工程を自動化するという問題を解決しなければなりません。今日，必要とされている解決策は，先ず，ビジネスのプロセス，活動，画面や帳票，そして，ビジネスルールを定義すると，コンピュータが自動的にデータベースと，そのデータベースに基づいて実行されるプログラムを自動的に生成してくれる，というものです。そして，これは一度きりの処理ではありません。ビジネス処理等からコンピュータが最初のシステムを生成する，というばかりでなく，数分や数秒以内に，それらの各要素への変更を取り込んで，再設計し，再生成し，新しいプログラムを再配置できなければなりません。それが実現した暁には，組織は膨大な費用や失敗の恐れ無しに，新しいビジネス環境や技術に順応（Futureproof：未来を守る）することが可能となるでしょう。

未来を守るための長期的な解決策

これまで見てきた様に，我々の組織の未来を守るための潜在的な短期的解決策は非常に沢山有りますが，それらの総てには重大な欠点があります。それらの解決策総てには，現在のソフトウェアのパラダイム（つまり，ソフトウェアは基本的に手作業で作成）を引きずっている，という根本的な問題があります。本当の意味で長期的な戦略として未来を守るためには，組織は新しい製品の開発と保守を劇的に改善するために，他のエンジニアリング分野で使われたことのないものを探し求めなければならないでしょう。未来を現実のものとするために，組織はソフトウェア開発に対する見方を変えなければなりません。つまり，次の２点です。

- 実際のエンジニアリングモデルをソフトウェア開発に取り入れる
- 自動設計と自動コード生成ツールを利用する

実際のエンジニアリングモデルをソフトウェア開発に取り入れる

エンジニアリングは，様々な分野で数百年，土木や建築では数千年に亘って進化を続けてきました。結果として，これらのエンジニアリング分野には確固とした方法論と設計書や設計を実際に行うためのツールが整備されてきました。数百年間にわたり，２１世紀[*11]の現在でも別の技術者が実装できる様な，設計書を作成するための「標準的な青写真」と呼ばれる標準的な方法があります，これらの標準的な方法は，コンピュータの処理能力の向上の恩恵を受け，近年数十年の間に設計されたものは，ますます設計書通りに正確に構築されるようになってきました。

（*11：＜訳者修正＞原文では２０世紀となっていましたが、２１世紀に修正しました。）

時間が経つにつれて，コンピュータのソフトウェア技術者は製図担当者が青写真を作成する際に，グラフィック CAD プログラムを使って，下書きを作成できる様にしました。今日では，現代建築とエンジニアリングの大半は CAD ツールを使って作業を始めます。CAD ツールの出現により，エンジニアリング会社で雇用されていた多くの社員は，より少ない人数の CAD を使える技術者に置き換えられることになりました。去っていった人達にとっては，伝統的な製図の下書きが出来ることが必須でしたが，残された人達にとっては，CAD のツールを使ってより簡単に作業できることが，もっと重要なのです。

しかし、現代のエンジニアリングと建築学は，下書き作成処理の自動化に止まりませんでした。このレベルの処理では，VISIO や PowerPoint のお絵かきプログラムと大差は無いでしょう。いいえ，同じではないのです。現代のエンジニアリング技術は Computer Aided Engineering(CAE)と Computer-Aided Manufacturing(CAM)のツールを作り出しました。CAD の製図から始まって，次に，コンピュータの支援する知性にまで適用し続けるツールは，設計されたオブジェクトをシミュレートし，そして，実際にそれを生産してくれます。

今日，同じ様なプロセスはソフトウェアの分野でも見られます。しかし、この「現代のエンジニアリングモデル」は，それ程広くは採用されていません。この現代のエンジニアリングモデルを採用すると，いくつかの基本原理に従わなければなりません。

- 主要な作業の分離
- 増え続ける知的な機能のツールへの組込
- ユーザと直接やり取りする能力の向上

- コンピュータでの詳細な作業履歴管理

現代のエンジニアリング分野では，*分離された工程*（要件，エンジニアリング，設計，工事）に*主要な作業*を切り離します：各々のこれらの工程には，明確な輪郭とインタフェースがあります。それらのインタフェースには，標準的な書式が用意されています。作業工程の分離と明確なインタフェースが用意されているため，エンジニアリング分野のコンピュータ化には著しい進歩が見られました。

ほとんど初期の段階から，技術者はコンピュータが彼らの仕事にもたらす可能性に気付いていました。コンピュータを使えば，コンピュータ自体の内部で「仮想の製品」を扱える，ということをイメージできました。そして，それらの仮想の製品から本物の製品を作り出すこともイメージできました。もちろん，技術者は製図の下書きの費用を削減することにも強い利点を感じていましたが，本当の利点は，ユーザと共に密接に作業できるという能力から得られたのです。全体的な概念から製造までの範囲で，横断的にコンピュータを利用することによって技術者が削減出来たのは，新製品開発時の費用だけでした。しかし，概念の段階から製品まで到達するのにかかる時間も劇的に短縮することができたのです。自動車から飛行機，コンピュータから携帯電話まで，何かが市場を変えてしまったのです。

*増え続けるインテリジェンスをツールに組み込む*ということは，誤りの数がより減少し，コンピュータの速さで物事が進んでいくことを意味しています。更に，洗練され，統合されたメタデータが存在するので，変更が発生した時は，コンピュータは総ての正しい場所に変更をあてることができます。

直接ユーザと一緒に作業を進められる能力の増加は，「賢いツール」であるための基本的な特性の一つです。現実にユーザが手に入れるものを想像できるのは，どんな分野であっても，非常に賢い，経験の有る少数のビジネスのユーザしかいません。ですから，何十年にもわたって，エンジニアや建築家はモデル作りに膨大な時間と費用を投資しているのです。新しい建物や車，又は，テレビのより現実的なモデルを作ることができれば，実際に建てられたものを見て，好きか嫌いかを言うような一般的な人達には，より簡単な世界になるでしょう。今日，一般的に使われているコンピュータ化された設計ツールを使えば，実際に製品が出来上がる前に，エンドユーザが手に入れることになる物を想像する助けとなります。

*コンピュータに詳細な履歴を管理*させるのは，現代のエンジニアリングや建築デザインツールの別の重要な本質的な機能です。例えば，建築家や技術者は数百年の間，通常の機械の製図から製品の完成図や現実の製品を作り出すことができました。しかし、完成図を作成するには非常に長い時間がかかったのです。その結果として，エンジニアリングの製造サイクルでは，これらの図面は一つか二つしか作られませんでした。今日，コンピュータを使えば，コンピュータの中に瞬時に現実的な三次元の

オブジェクトを作り出すことができ，ユーザはどの様な角度からも，「仮想の建物や製品」を見ることができます。実際，ビルのようなものでは，あたかも，完成した新しいビルの中を通るツアーの映画を見ているかの様に，これらのビルの中を「仮想に歩き回る」様なことも実現できます。

ソフトウェアに関しても，エンジニアリングや建築と同じ様なことができなければなりません。ソフトウェアの開発者はコンピュータ上に新しいソフトウェア・アプリケーションを定義できなければなりません。そして，ユーザが自分の目でソフトウェアがどんなものか，どの様な動きをするかを，確かめられなければなりません。そして，他のエンジニアリングや建築のソフトウェアのように，ソフトウェアの技術者はどんな局面でも，ソフトウェアの要件や設計を変更でき，それから，コンピュータに即座に再設計させ，変更されたアプリケーションを実行して見せられなければなりません。これを実現できる唯一の方法は，設計とコード生成を自動化することです。

自動設計と自動コード生成ツールを利用する

自動デザインと自動コード生成を利用するのは，コンピュータ支援設計ソフトウェア（CAD）をソフトウェア・アプリケーション構築の世界に拡張する，という自然な流れです。ソフトウェア技術はとても複雑になったので，プログラマーやスーパープログラマーでさえ，彼らの頭の中だけで適切なオプションを覚えておくのは難しくなっています。もし，組織が自らの未来を守ろうとするのならば，簡単に開発でき，エラーが無く，保守の容易なシステムが必要でしょう。この様なシステムは SOX 法（Sarbanes-Oxley）の世界では更に重要になります。今日では，以前よりも増して，組織は彼らのシステムが正しく働いており，彼らのコントロールがフェイルセイフであることを証明できなければならないでしょう。こうするためには，ますます自動デザインと自動コード生成に依存することになるでしょう。

さて，コードジェネレータと呼ばれるものは数十年の間に幾つも出てきましたが，多くの場合，自らの限られたメタデータのために機能が制限されていました。*本当に，コードジェネレータが有効となるには，一貫してデータとビジネスルールの双方を結び付けられる非常に充実したメタデータに基づかなければならないでしょう。*最先端のコード生成は，非常に充実したメタデータを含んだ知識ベースを使用しています。つまり，複雑なアプリケーションにある関係の総てをサポートするくらい充実しています。

プログラマー達は何 10 年もの間，複雑なソフトウェアを構築するためのソリューションは，より精巧で複雑な言語になっていくのではないか，という議論をしてきました。経験から見て，どの様な言語，OS，データベース管理システムを使っても，本当の意味で複雑な問題を解決できるとは思えません。ソフトウェアの複雑性というものは，総ての関連する断片の知識や他との関係を伴っているもの

です。コードジェネレータという正しい範疇に入るツールは，自動設計ツールとの連携が取られており，豊富なメタデータ・データベースから順番に情報を引き出して使います。

ソフトウェア開発作業の中の単なるプログラムレベルで，余りにも多くのツールが使われています。しかし，洗練されたプログラム作成作業というものは，常に他のシステムや，しばしば複数システムと関係しています。一つのプログラムのために，一つのテーブルを設計している間は問題有りません。しかし，異なった案件を処理する数百や数千のプログラムで処理するデータベースを設計する場合はどうでしょうか。例えば，同じデータ項目が，複数アプリケーション内の多くのプログラムによる多くの画面上に存在しているかもしれません。例えば，数千本のプログラムを通して，同じ式（フォーミュラ）が数百回も実行されているかもしれません。充実したメタデータの知識ベースを使えば，これらを使用している総ての履歴を押さえることができ，一箇所で変更が入れば，システム全体を通して変更を反映させられます。同じ事はデータベースの変更についても言えます。充実したメタデータ知識ベースには，個別の属性項目やテーブル，ビジネスルールと同様に総ての関係の履歴が管理されています。

充実したメタデータ知識ベースというのは，それ自体が高度に洗練されたデータベースであり，絶えず最新の変更点を反映して更新されています。最も重要なこととして，一つの値か関係が変更された時，重要な総ての変更が行われることを保証するために，一貫して知識ベースは更新されなければなりません。自動デザインと自動コード生成は単なるアプリケーションであり，一つの一貫した知識ベースに対応した非常に洗練されたアプリケーションです。

緩慢ですが，ソフトウェア産業もこの事実を認識するようになってきました。最近の Model Driven Architecture (MDA)への関心の高さは，ビジネスの定義とコードとの間に何等かの繋がりがあるに違いない，という認識に基づくものです。これまで，MDA は初期段階にあり，大半でなくても，主な多くの変更作業は，まだ手作業で行われています。MDA が完全な解決策になるためには，MDA は CAD，CAE，CAP，CAM のような本来の意味でのソフトウェアのソリューションに発展しなければなりません。

MDA が実現するには，恐らくこの先，何年も，何十年も時間がかかります。しかしながら，既に未来を守るためのソリューションは存在しています。これまでに，そのソリューションは数千もの非常に大規模で，様々なプラットホーム上に，多くのデータベースを持つアプリケーション・システムの構築に使用されてきました。これまでに非常に長い成功の歴史を持っているそのソリューションは **GeneXus**（ジェネクサス）と呼ばれています。

ビジネスと技術の知識の結合

７０年代から８０年代にかけて，主要なアプリケーションの大半は，自らの利益のために技術を利用しようとする組織によって開発されていました。これらのシステムは，自らのビジネスに適合する様にカスタマイズされ，ほとんどがメインフレーム・コンピュータ上で書かれ，これらの組織の中核となるビジネス処理の実現に関わってきました。９０年代になると，アプリケーション開発の傾向は，自ら中核となるアプリケーションを構築する代わりに，パッケージ・ソフトウェアを購入する様に変わりました。この傾向は，ビジネスの競争で有利な立場を保とうとする組織に，興味深い選択肢と写りました。図１を詳細に見ると、幾つかの問題点に気づくでしょう。

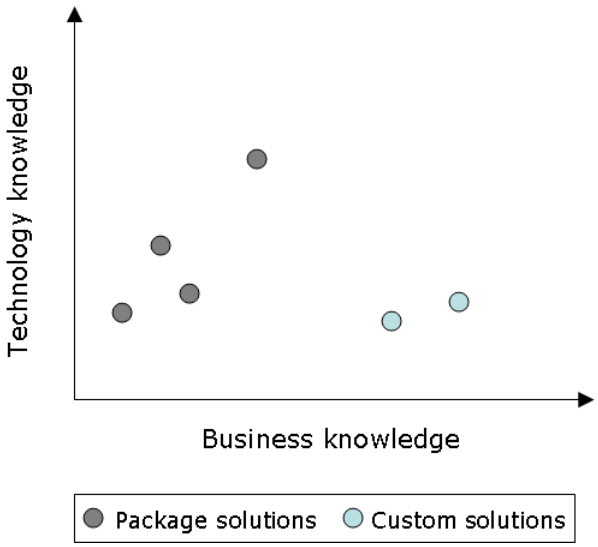

図１– ビジネス　対　技術の知識

一般的に，顧客専用にカスタマイズされたアプリケーション（図１に薄い色で表示）は，優れたビジネス上の知識を含み，一方，パッケージソフト(灰色で表示)は，より洗練された最新の技術に対応している傾向があります。過去においても，顧客専用に開発されたアプリケーションは，独自のビジネスルールやビジネスプロセスを実現した深い知識を持っていましたが，技術の変化（例えば，リレーショナル・データベース，ウィンドウズ，クライアント／サーバ，インターネット，Web サービス，スマートディバイス[*12]等）に追随していく，という点で問題が有りました。パッケージソフトは，特に最新のものは，これとはまさしく逆の問題を持っています。最新の技術を利用しようとすると，それらのソフトに顧客専用のビジネス上の知識を取り入れるのは難しくなる，という問題が出てきます。

（*12：＜訳者補足＞スマートディバイスを追加しました）

この様な状況の結果，組織はジレンマに捕らわれることになります。つまり，ビジネスとして最大の柔軟性を確保するために，購入したパッケージソフトに大きな変更を加え，独自のアプリケーションを構築しようとするか，又は，購入したパッケージソフトが要求する様に自らのビジネスを合わせるか，というジレンマです。明らかに，組織の未来を実際に守るには，ビジネスの変化に迅速に適応し，同様に技術の変化にも迅速に適応できる，何等かのソリューションを見付ける必要があります。パッケージソフトは，異なった数百もの市場の数千ものビジネスをサポートしなければならないので，それがどれほど一つの組織に適応できるか，まして，一つの市場に適応できるか，という点に関しては，パッケージとして提供できるソリューションは限られているでしょう。

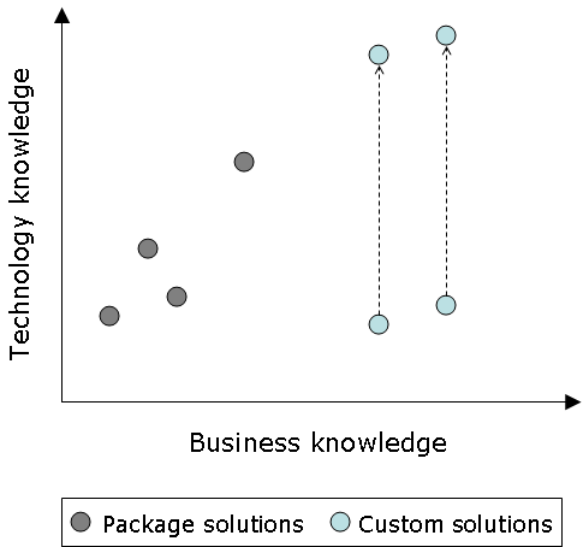

図 2 – ビジネスの知識　対　技術の知識　(最適解)

もし，ビジネスのニーズと新しい技術（図２参照）の双方を，楽に適合させられるやり方で，顧客専用のアプリケーションを構築できれば，それが組織にとっての最適解となるでしょう。長期にわたって，真のアジャイル開発というのは，図２（ビジネスの知識　対　技術の知識）の右上コーナーに位置する様なソフトウェアを組織は持つことができるだろう，という意味を持っていました。組織が真の意味でアジャイルを目指すならば，「高い技術」と「高いビジネスの機能」で表した図の，四番目のコーナーである右上隅に移る必要があるでしょう。次の章で，今日最先端のツールを使って，これがどの様して可能となるかを説明しましょう。

２１世紀の未来を守る本当の解決策

GeneXus(ジェネクサス)というのは，現在まで約２５年間[*13]にわたり開発され続けてきた最先端のシステム開発環境です。このツールには，人口知能やデータベースとユーザ・インターフェースの自動設計，ビジネスルールの設計といった，幾つもの先進技術が組み込まれています。主要技術を統合し，大きく進歩してきた結果として，基幹ビジネスの知識が更新され保守されている間も，組織はメインフレームからクライアント・サーバ，Web ベース，そしてサービス指向アーキテクチャにまで，ソフトウェアの開発環境を発展させることができました。成功のこの流れは，幾つかの非常に重要な見識に基づいています。

（*13：＜訳者修正＞原文では約２０年間と書かれていますが、約２５年間と修正しました。）

- 一つ目は，個々の業務上（ビジネスのトランザクション）のデータ構造を記述した情報から，効率的で，正規化された概念データモデルを推論し，このデータベース構造に基づいて作動するためのナビゲーションを自動的に生成することは可能だ，というものです。
- 二つ目は，上記データ構造の仕様に変更が加えられるか，新しいデータ構造が加えられた時，データベースを再設計し，再正規化して，古いデータベース構造を新しく変換し，次に，新しいデータベース[1]に基づいて作動する総てのコードを再生成することは可能だ，というものです。
- 三つ目は，一般的な生成エンジンを開発し，与えられたプラットホーム向けに作成された様々なジェネレータを使うことによって，あらゆる主要言語や商用リレーショナル・データベースやディバイス上[*14]で実行可能な処理を実現することができる，というものです。
- 四つ目は，真の意味でビジネスの変化と技術の変化の双方に追随できるシステムは，どの様なものでも，ほとんどのアプリケーションのためのコードを 100%生成できなければならない，というものです。

（*14：＜訳者修正＞ディバイスを追加しました）

この過激とも言える戦略を採用することによって、GeneXus の研究者と開発者は建築やエンジニアリングの分野で使われている CAD/CAE/CAM **と同様に動作する一連のツールを作り出すことができ**

[1]長年にわたり，開発者は，どんなシステムでも，変更が入ると，結局データベース設計は最も難しい作業になる，ということを知っていました。理由は，殆どの開発環境に依存するもので，一旦，プログラマが与えられたデータベースの設計に従ってコードを書き始めると，データベースの設計に変更が入った場合，その費用は劇的に増加します。何故なら，総てのデータベースを扱っているコードは，再度書き直さなければならないからです。GeneXus は，データベースを扱う総てのコードを自動生成することで，この問題を解決しています。

ました。結果として，このツールを採用した組織は，何日や何週間という単位ではなく，数分や数秒といった単位の短時間で，実際に稼働するアプリケーションを作成することができました。そして，もっと重要なことは，これらのアプリケーションを素早く変更することができたのです。誰かがアジャイル開発という名前を付けようとしていた時よりもずっと以前に，**GeneXus** の開発者は既にそれを実現していたのです。

しかし，どの様に素晴らしい技術であっても，その素晴らしさを主張するだけでは誰も信じません。人が信じるのは，その技術がもたらす結果だけです。今日，世界中で、１０万ライセンス[*15]以上の **GeneXus** が，様々なデータベースやプラットホーム上で，推定で年間約３０億ライン以上のコードを生成しています。平均的なアプリケーションの特徴は，下記の通りです。

（*15：＜訳者修正＞原文では５万ライセンスですが、現在では１０万ライセンス以上です）

- ほぼ１００％自動生成
- 生成されたコードは平均１００万から５００万ライン
- 一つのアプリケーションで扱うリレーショナル・データベースのテーブル数は５００個以上
- 最少の工数で一つのプラットホームから別のプラットホームに移植

ソフトウェア開発というものは，誰もが初期の段階で予想したしたよりも，遙かに複雑な仕事であることが明確になりました。ここ数年の間，プログラムとプログラム開発は非常に多くの注目を集めていますが，統合システムの開発についてはほとんど注目を集めていません。システムというのは，単に個々のプログラムの集合であると思われていました。しかし，システムというのは個々のプログラムよりも遙かに複雑であり，結果として，一般的なデータとビジネスルールの集合に依存しているのです。一本のプログラムが２万ラインを超える様なケースは希ですが，今日では，一つのシステムが１０００万ラインから２０００万ラインものコードから出来ているケースは決して希ではありません。

歴史的に見て，システムの開発者は，際限のない多様性を持つプロジェクト管理計画を通して，システム開発に関わる問題を解決しようと試みてきました。これは，個々のプログラムが独立して作成でき，しかも『システムの完全な状態』を保つために，膨大な時間をシステムとデータベース，そして，インタフェースの設計に費やさねばならなかったことを意味していました。

システムを基本的な作業の単位として扱っているので，**GeneXus** はコード生成へのアプローチに関して，大変な成功を収めてきました。例えば，与えられた知識ベースは，与えられたアプリケーション(システム)の総てのデータに関する情報を含んでいます。このため，そのコード・ジェネレータは，

通常のコード生成の方法よりも，データを扱い更新するというより重要な方法によって，遙かにインテリジェンスを持つことになります。

未来を守ること（Futureproofing）の将来

今日のビジネスの世界は，未だかって経験したことの無いほど変化の激しい状態となっています。グローバル化や激しい競争などの，組織が求める急激な変化への緊急度の高いニーズの総てに，新しい技術は貢献してきました。未来を守るために二つの大事な事を忘れてはなりません。つまり，(1)未来と現在の決定が未来に及ぼす影響に関してもっと真剣に考えること，そして、(2)組織に最大の柔軟性を許すツールと技術を入手すること，です。

GeneXus の開発元である ARTech 社は，未来を守るソフトウェアを開発している専門家集団です。およそ２５年前[*16]，ARTech 社はこれまでとは完全に異なった種類の，一群のソフトウェア開発ツールの必要性に気付きました。つまり，ソフトウェア技術者は技術の変化に対応することに貴重な時間を費やすのではなく，むしろ，新たなビジネスの可能性にフォーカスすることに時間を費やすべきである，というものです。また，彼らは長い時間をかけて，伝統的なウォーターフォール型の開発方法に頼るよりも，インクリメンタル（アジャイル）開発の方が遙かに生産性が高いことに気付きました。

（*16：＜訳者修正＞原文では２０年前となっていますが、２５年前に修正しました。）

また、称賛に値することですが，技術革新に追随するために，世の中に現れたワークフローやスマート・ディバイス[*17]の様な各新技術を統合する重要性を認識することで，時代を総ての面で先取りしようとしています。未来について真剣に考えることによって，お客様の「未来を守る」ことをビジネスの根幹に据えたのです。その結果，ARTech 社は，世界中のビジネスが「組織の未来を守る」ために賭けることの出来る技術として利用可能な，競争相手を凌駕した一群のツールを作り出したのです。

（*17：＜訳者修正＞原文では無線通信となっていますが、時流に合わせスマート・ディバイスと変更しました）

> “多分，これから１０年か２０年の間には，未来を守るための別のソリューションが現れることでしょう。しかし，**GeneXus**は今すぐに利用可能ですし，未来を守るための方法を示すことができます。”
>
> （Breogán Gonda，元ウルグアイ公立大学教授，ARTech社の創業者＆社長）

著者紹介：Ken Orr，**米国** Ken Orr Institute **の創設者，**1970 **年代よりＩＴ技術の進歩に貢献し，**Warnier/Orr Development Methodology (formerly known as DSSD)**の開発に携わった。近年では，エンタープライズアーキテクチャ，アジャイル開発，ビジネスプロセス分析，知識管理等に関する活動をしている。**(http://www.kenorrinst.com/)

（訳責＆加筆修正：ジェネクサス・ジャパン(株)　大脇文雄）