GeneXus™

Grow thru knowledge

# Futureproofing Your Organization

Ken Orr

Latest Update: 2009

GeneXus international
www.genexus.com

| | | |
|---|---|---|
| MONTEVIDEO - URUGUAY | Av. Italia 6201 - Edif. Los Pinos, P1 | (598) 2601 2082 |
| | Ruta 8 Km. 17.500 - Edif. @3 Of. 205 | (598) 2518 2870 |
| CHICAGO - USA | 1143 W Rundell PL, Suite 200 | (1 312) 836 9152 |
| SÃO PAULO - BRASIL | Rua Samuel Morse 120 Conj. 141 | (55 11) 5502 6722 |
| CIUDAD DE MÉXICO - MÉXICO | Calle Leibnitz N° 20, desp. 801 | (52 55) 5255 4733 |
| TOKYO - JAPAN | Tk Gotanda Bldg. 303-2 | (81 3) 5793 5481 |

*If you want to know your past, look into your present conditions.*

*If you want to know your future, look into your present actions.*

*Buddhist Saying*

# Futureproofing and Change

*Futureproofing, def., "the act of ensuring that current decisions work in the future."*

"Futureproofing" –what a wonderful sounding concept! Wouldn't it be great if it were only possible to make business and technology decisions today that would stand up under the barrage of constant change and still make sense well into the future? What if it were possible to build systems that were so flexible that they could be adapted easily and quickly no matter what business or technology changes came about? What if it were possible to really "futureproof" your business? Maybe it is not as farfetched as it sounds. This whitepaper is about futureproofing your organization using a proven set of software development and maintenance technologies.

Consider the idea of futureproofing. Futureproofing is all about understanding, really understanding change, and how to deal with it. Clearly, in this modern world, change is the most critical factor in any business, whether that business is public or private organizations; large or small. Every business everywhere has to cope with constant change today. And all managers, regardless of their level of responsibility, have to make critical decisions, decisions that have to stand up in the face of accelerating change, and nowhere is that more true than in IT.

Every week, nearly every day, new devices and new software comes onto the market that creates significant new opportunities and at the same time obsoletes huge investments in existing devices and/or software. Nearly every week, there are business developments that change the business landscape as well. The success, indeed the very existence, of many organizations over the next few years will depend heavily on their ability to manage change—futureproof their organizations—and that's what this white paper is all about.

# Megatrends: where is all this change coming from?

But change, even massive change, is not a new event. In the last century alone, we saw the introduction of radios, airplanes, televisions, atomic energy, integrated circuits, space travel, genetic engineering and, certainly not least, computers. At the beginning of the 20th century, most people were traveling on foot or on horseback; by the end of the 20th century, they were traveling in high-speed cars and airplanes. At the beginning of the 20th century, telephones were a rare occurrence, by the end of the 20th century cell phones where as commonplace as wallets or briefcases.

But there are new dimensions to the kind of change that is happening in the 21st-century; there are a number of megatrends that are pushing our world faster and faster and making even the most mundane decisions riskier and riskier. The most important of these megatrends are:

- Megatrend 1: history is accelerating
- Megatrend 2: the future is highly unpredictable
- Megatrend 3: the future is non-linear

## *Megatrend 1: history is accelerating*

Today, the pressures of business and technology are such that organizations and people are under more and more stress to keep up. The average business person today is likely to carry a cell phone, a PDA and often a laptop computer as well. 50 years ago, telephones were tethered to the wall. Only a handful of the wealthiest people could afford radio telephones in their cars or on their boats. Today, more and more people, working people as well as wealthy people can afford cell phones. Each day, more and more devices are being announced that make it possible to communicate from almost any place on the globe.

In the world of software, change is accelerating as well. The lifespan for platforms and languages is decreasing at an accelerating rate. Eight years ago no one was thinking about Java. Five years ago, no one was thinking about .NET. And three years ago hardly anyone was thinking about Linux.

In hardware world, change is happening even faster. Three years ago no one was thinking about iPods and today everybody is. Today, disk drives and memory and CPUs are so small and so inexpensive they can go into almost any device. With each generation of new devices there is increased integration and miniaturization and the cost is reduced by an order of magnitude.

But people don't change nearly as fast as business and technology. As a result, organizations have difficulty keeping up. 30 years ago, IT managers could make decisions and expect them to be good for five or 10 years at least. Today, the lifespan of any, even the most promising new technology or software tool or platform, is extremely limited.

## *Megatrend 2: the future is less and less predictable*

*Prediction is very difficult, especially of the future*. – Niels Bohr

Just since the beginning of the 21st century, we have seen enormous changes: the end of the .COM bubble, the attack on the World Trade Center and Pentagon on 9/11, the explosive growth of the Indian and Chinese economies, the increasing outsourcing of jobs and products to the Far East, the war in Afghanistan and Iraq and the devastating earthquake and tsunami in the Indian Ocean that took the lives of 250,000 people in just a few hours.

Only the most perceptive analysts and historians could possibly have predicted what has already occurred in just the last five years. And only the most foolhardy would be willing to predict what is likely to happen in the next the five years, much less the next twenty.

Historically, futurists and forecasters have attached risks or degrees of error to their predictions. Today, people who are paid to look ahead are more and more reluctant to be pinned down to any level of certainty, for any period of time —the future is just not predictable in the way it used to be— the world is moving too fast!

### *Megatrend 3: the future is nonlinear*

Many people who earn their living trying to predict the future will also tell you that one of their most important tools is a straight edge. They have found that many trends are linear. But the future is no longer linear. Even a guide as historically reliable as Moore's Law is beginning to falter. Beginning in the mid-70s and running up until the early 21st-century, personal computers and all the technology behind personal computers drove an enormous wave of technological innovation and expansion according to Moore's Law. But the personal computer is no longer driving technology; in fact the PC world is struggling to keep up. And Moore's law may be faltering.

The world is undergoing a period of discontinuity. Where the personal computer and the Internet were the engines that drove the late 20th century, something else, perhaps smart phones, or personal entertainment devices or wireless communication, is driving the world now. And where in the age of the personal computer, forecasters talked in terms of millions of devices, in the age of wireless smart phones, forecasters now talk in terms of billions of devices. Not surprisingly, organizations, like people, have trouble adapting to such changes in scale.

With all this change, what organizations need in the software arena is a new generation of tools, or more precisely, a new way of developing and maintaining business applications and a set of tools that support this new way so that they can keep pace with the changes of the 21st-century. The rest of this white paper addresses just what kind of technology is needed to make this happen.

# Problems associated with Futureproofing our software systems

Before we discuss what the serious futureproofing solutions might look like, though, we need to consider some of the major new issues that will have to be addressed:

- The need for new applications and new capabilities will far exceed our ability to deliver them
- The next generation of applications will need a whole new set of capabilities

- There is a need to create a stable environment that will focus more on dealing with new business problems / opportunities and less on solving technological problems

## *The need for new applications and new capabilities will far exceed our ability to deliver them*

So far we have been talking about change; in this section we have to discuss change in terms of *hyper-adaptable systems*. These hyper-adaptable systems have to deal with the following:

- changing business needs
- changing technology opportunities
- increasingly complex development environments
- a lack of trained people
- a huge turnover of experienced people due to retiring "baby boomers"

Since the collapse of the .COM bubble, a great many organizations have been retrenching their IT expenditures, especially in the area of developing new systems. Organizations have been increasingly moving to: (1) outsource their application development departments along with their IT operations functions, (2) purchasing large, integrated packages to replace portions of their legacy systems, and (3) developing web-based front ends to their mission-critical mainframe applications. These trends are about change.

The business world in 2005 is much more competitive than it was a decade ago or even five years ago. The world has become much more interrelated, integrated and technology savvy. At the same time, the systems that many organizations rely on for their day-to-day survival are getting older and older, and the people who develop those systems, if they are still around, are getting closer and closer to retirement.

In the next decade, most large organizations will have to either replace or redevelop many of their most critical applications —their core applications. And in many cases, the applications that they will need the most will not be available on the commercial market—they will have to be developed, and they will have to be developed quickly.

## *The next generation of applications will need a whole new set of capabilities*

What will this next generation of hyper-adaptable applications need to look like? Well, for one thing it will be increasingly important that these applications be capable of being moved quickly from one platform to another. While there has been lip service given to "open systems," most applications, even today, are developed for only one platform, only one language, and only one database system. Cost and development concerns are likely to change that outlook more rapidly than anyone could have imagined even a few years ago. Today, for example, there are not just two competing development platforms (Java and .NET), there are at least three (Java, .NET, and Linux). How many will there be in the near future? Nobody knows.

And while most organizations have become good at building systems that work on desktops and laptops connected to the Internet, the need to support wireless devices with small screens that connect at low bandwidth represents a new challenge. So is developing applications that need to work with devices that are not connected at all times. Who knows what the next challenge will be? Maybe it will be RFID, maybe it will be GPS enabled devices, maybe it will be all of the above. What we do know for sure is that the number and types of devices are multiplying even while we're thinking about our current choices. And every new device represents a new challenge. Devices created for one purpose will get modified and used for something else. Devices nobody ever thought about will be created and have to be supported.

So the next generation of IT applications need to be able to move easily from one platform to another, from one language to another, and from one set of hardware to another. And they will need to be able to support different database management systems. As programmers will tell you, this is no easy matter. Even though major relational databases are all very similar, the way things are constructed today, there are enough differences to make even relational database to relational database conversion a nontrivial task. And the same is true for languages. Even though Java and C# are very similar, converting from one to the other is again no easy matter.

Finally, the next generation of applications will have to be: (1) easier to communicate with, (2) more fault-tolerant, and (3) orders of magnitude more secure. As computers move further and further into the mainstream of everyday life, systems will have to be designed not for the techie but for grandmothers and grandfathers. The reason that Apple Computer has been able to survive and reappear as a competitor in the high tech world, is in no small part a result of the attention that Apple has always paid to human interface design—to making things easy to use.

### *The need to create a stable environment will focus more on dealing with new business problems / opportunities and less on solving technological problems*

If we are going to develop a solution for futureproofing our organizations, then we're going to have to concentrate increasingly on approaches that allow us to spend most of our time with the user exploring how best to solve their business problems. Currently, too much systems effort is involved in figuring out how to utilize the enormously rich technology base that exists today.

To be successful, in the future, we will have to come up with solutions that allow us to concentrate the majority of our development effort on understanding and treating the business problems, and smaller and smaller percentages of our time devoted to technology.

# Strategies for Futureproofing

To this point, though, we've only set the stage for futureproofing our organizations. We've made the case for futureproofing, but we haven't look at what the serious alternatives are. Clearly, people have been trying to solve the "problem of change" since the beginning of computing. Developing workable software has always been a challenge, but bigger still has been the challenge of maintaining or modifying software to do things that it was not originally intended to do. In this section, we are going to look at both the short and long-term solutions to the problem.

## *Short Term Solutions*

Today, in the business world, there are a number of popular solutions aimed at helping organizations cope with the effects of change. The most popular are:

- commercial packages
- outsourcing
- agile development
- reusable components

## Commercial Packages

More and more organizations around the world have decided that they were "not in the software development business." In practical terms, this has meant purchasing critical software rather than developing it. Rather than attempt to maintain their own staff of highly expensive software experts, many companies have chosen to buy commercial software packages from vendors. This puts the onus of keeping the software up-to-date on the vendor. Changes in technology, changes in business rules, and new innovations are supplied as part of a cost of the package, and, of course, the maintenance of the package. Buying package software means that an organization doesn't need large numbers of trained analysts, designers and developers, and it means that the organizations only have to worry about which package to acquire —or does it?

The problem is that packages have a dark side. No package, no matter how well conceived, will meet all the needs of a large organization out-of-the-box. So companies purchasing packages are caught on the horns of a dilemma: (1) do you customize the package to meet your organization's unique needs, or (2) do you keep the package intact and change your organization to fit the package?

Either approach is fraught with problems. In the early days of package software, organizations tended to customize the packages extensively. This usually meant that they would have serious problems in moving from one version of the package to the next. After a couple of increasingly difficult version changes, these organizations would usually move to a strategy of minimal customization and increased organizational change to fit the package. But this didn't work either and they would have to go back to more customization. History shows that the pendulum continues to move back and forth.

But packages have an even darker elements than customization. Over time, organizations find themselves wedded to specific vendors. It is extremely difficult to back out of the commitment to a major package in anything under 8 to 10 years. And in this time, the competitive landscape may have change dramatically —Peoplesoft buys J.D. Edwards, Oracle buys Peoplesoft, etc. Software companies go out of business or are acquired all of the time, and suddenly organizations find themselves wedded to software vendors that they never explicitly selected.

## Outsourcing

Another way that organizations have tried to handle their software change problem is to delegate it to some external organization. In the last few years, outsourcing has become a popular method of dealing with software and software changes. Rather than keep your own staff of software experts, the argument goes, why not give that work over to firms that

specialize in software? Trade papers are full of announcements about large organizations outsourcing their software development and/or their computer operations functions.

Normally, like purchasing software packages, outsourcing represents a long-term commitment. Frequently, the in-house staff is transferred to the outsourcing company, and in the short run most of the people, except for the top IT management and professionals, retain the same of the relationship, except that now the discussions between analysts, designers and developers and the business users becomes much more formal and arm's-length—and, of course, expensive. And like packages, outsourcing has definite drawbacks.

Outsourcing often builds barriers between the business users and the software developers. After a while, the original staff that once worked for the customer organization begins to move on or out, and the next generation of outsourcing personnel have less of an attachment and interest in the business they are building software for. In addition, outsourcing companies often have a vested interest in minimizing changes in the underlying software or platforms. Major changes often mean new develop or bringing in packages, activities that threaten the outsourcing arrangement.

Probably the most serious problem that outsourcing encounters is the loss of knowledge on the part of key people in the business; organizations that outsource all or most of their software development often end up lacking individuals who know enough even to manage the outsourcing relationship. Typically, as more time goes by, the outsourcing organization wields more and more power and friction builds between the organizations. Organizations attempting to take back control of previously outsourced relationships often have to rebuild the organization structure that they seceded to the outsourcer, a process that normally takes many years, sometimes decades.

### Reusability: Objects, Components and Services

Installing packages and/or outsourcing are major strategies organizations have been employing in recent years in an attempt to lower their costs and to a high degree lower their exposure to change. Both of these strategies basically involve offloading software development and maintenance to someone else. But in many cases, large amounts of software still have to be developed in-house. Some projects are too strategic, or too time sensitive or too critical to farm out.

For those organizations that are still doing software development, the Holy Grail, for the last 10 or 15 years, has been reusability. Software is so expensive and so error-prone, many software thinkers believe, primarily because software, unlike other forms of products, has

not yet evolved into an engineering discipline based on "reusable parts." To a high degree the "object oriented revolution" has been sold on the concept of reusability. Objects were intended to be designed in such a way that they could be easily used in a large number of different programs.

Over the last 10 to 15 years a whole range of reusable solutions has emerged, first objects and components, and now services. The idea behind what is now called "component-based development" is that, if people design components the right way then there will become a marketplace in which rather than developing programs from scratch, developers like engineers, will be able to pick and choose standard components from a catalog or from the Internet and simply "assemble" them to create new programs.

Some parts of this reusability agenda have worked. Objects and components are widely used in various parts of the programming community today. For example, objects are widely exploited in graphical user interface (GUI) design. GUIs are traditionally complex and difficult to create, and objects are a natural way of building complex interfaces. Other areas include presentation applications, such as translating numbers into graphs or charts. But, the concept of "business objects" has not been nearly as successful —objects/components have not made serious headway into the business domain in the same way they have into GUI or presentation logic.

Part of the problem behind reusable components is their static nature. From the beginning, it has been assumed that objects in software are similar to parts in an automobile or a refrigerator—that has not turned out to be the case. For one thing, software parts turn out to be much more complex than most physical parts. For another thing, object/component designers have forgotten to include a major element in product design in their equation—the product structure.

In the real world, product engineers spent a great deal of their time designing the overall structure of the product. The most important design diagrams show the overall structure and where each component piece or part fit. Software designers have unfortunately left the product structure out (or have underestimated its importance). The result is that most off-the-shelf components are static components. One of the results of this oversight is that components end up being much more complex than they ought to be, since the designers have to think about all the possible uses that a static object might be put to.

The consequences of these assumptions about reusability have been that developing object-oriented software has turned out to be both more complex and more expensive than had

been anticipated and the software produced has turned out to be more difficult (and costly) to maintain, an especially vexing problem since more than half of the cost of any software application is ultimately spent in maintaining and updating the application.

So there is a great deal to be learned from looking at product engineering in a broader sense. Over the last 30 or 40 years all forms of engineering have taken great strides in using computers and computer software to help them design, engineer, prototype and manufacture their products. Computer Aided Design and Drafting (CAD), Computer Aided Engineering (CAE), Computer Aided Prototyping (CAP), and Computer Aided Manufacturing (CAM) are uniformly used throughout the engineering world.

Today, it is possible for engineers to move from conceptual drawings to prototype parts in a matter of hours. CAD drawings can be analyzed and simulated using CAE software, problems can be identified, changes can be made, and new parts can be fabricated using CAP or CAM software, all from digitized, virtual products stored as meta-data in high-speed, graphic computers.

The digital products and parts stored in computers in aerospace, electronics, and architecture, unlike those in software, can be instantly reconfigured and final products produced. Unfortunately, software development in most advanced software development shops still requires enormous manual intervention. Software developers argue that software is too complex to be generated by computers, but the rest of the engineering world has gotten past that idea. *If organizations are going to truly futureproof themselves they must turn to proven technologies based on the other more advanced form of engineering — dynamic requirements-design-generation-prototyping tools.*

### Agile Development

Finally, there's one more short-term solution to the software change problem—something called “agile development.” Throughout most of the history of software, the predominant approach has been a phased, “waterfall strategy” wherein projects are broken into major phases (planning, requirements, design, construction, and installation). Each phase is completed before the next phase is begun. This approach was taken over from strategies for building large-scale hardware products during and just after World War II. The great strength of this approach is that it allows for specialization, and it is easy to understand.

But the waterfall strategy, while appealing, tends to have a large number of problems itself. Because software is not a mature discipline, there are no agreed-upon standards for specifying, designing, and testing programs, databases and especially business rules. On

very large projects, each of the major phases may take months, even years, and as a result by the time large projects are implemented they are already out of date. The larger the project the more likely it is to fail.

In the mid-90s, a variety of software thinkers began to attack this bedrock of software development. They began to ask if it would not be better to design, build and test software in small increments, working directly with end-users. Then, after the user evaluated the current increment, refined their ideas, and defined the next set of requirements, the developers could redesign the product adding any new pieces and correct any errors or changes to the previous set of requirements. Then they could create a new increment to the user to review, again in a very short period. This whole process became known as “agile development.” Organizations that have employed this approach have reported amazing gains. But like everything else, there's a dark side to agile development.

One of the reasons that agile development is so popular is that both developers and end-users are rewarded immediately. There's no lag between request and response, no wasted effort. Requirements are turned directly into code. However, as anyone who has done software development software knows, complex software often requires serious design and analysis.

The dark side of agile development, then, has to do with the lack of design documentation and with the cost of maintenance over time. One of the Golden Rules of agile development is that project teams should avoid paper documentation wherever possible. Many extreme agile developers even believe that “the code is the only documentation you really need.” Now, this is a statement that has been made many times in the history of software development, mostly by hackers, but it never turned out to be the case. Maintenance, especially maintenance by people who didn’t work on the original software, has always required good documentation. Poor documentation = software that is difficult (i.e., expensive) to maintain.

But, agile development has a number of things to recommend it. For example, incremental development is a great idea. Historically, one of the major problems with software development has resulted from attempting to project current requirements too far into the future. Indeed, this is one of the most significant problems faced by object-oriented designers: the desire to design the perfect object. So, incremental development is a good idea.

And working closely with the users to develop prototypes is also an excellent idea. Experience has shown the faster one can translate requirements into running programs the better. Most users are better with tangible things like prototypes than they are with abstract drawings of those screens.

And redesigning programs, really redesigning them for each increment, is also an excellent idea. In the agile development world, this has come to be known as "refactoring." While it sounds super sophisticated, refactoring simply means reviewing all of the database (object class) design and code, to make sure new requirements and changes are "built-in," rather than "added-on." The only problem with refactoring, especially manual refactoring, is that the bigger the system or application, the longer it takes. So that as a program or application gets bigger and better, the more effort it takes for each new refactoring. This means that either the development iterations get slower and slower, or refactoring gets thrown out with the bath water.

## Learning from Short Term Solutions

In the mid-1980s, there was a great push to "automate software development" to produce Computer-Aided Software Engineering (CASE) tools. Unfortunately, the technology was not ready. There were simply too many things that had to be invented and integrated; too many things the software industry simply hadn't figured out. But the goal to automate software development was right, in other words, to define software much as product engineers or architects define products or buildings, and then have the computer generate the end solution in seconds or minutes rather than weeks or months. Indeed, when you think about it, it would appear that software would be the ideal domain for this to happen, since software products are essentially digital in nature to begin with.

In order to truly futureproof our organizations, we will have to solve this problem of automating the software development process. We need solutions today that will allow us to define our business processes, activities, screens/reports, and business rules, and then let the computer automatically generate the databases and programs that run against those databases to produce running applications. And this cannot be just a one cycle process. Not only must the computer be able to generate the first iteration from our business processes, etc., it must be able to take any changes to those elements and redesign, regenerate, and redeploy those new programs as well again in seconds or minutes. Only then will it be possible for organizations to adapt to new business circumstances and new technologies (futureproof) without fear and without enormous cost.

## *The Long-term Solutions for Futureproofing*

As we've seen, there are a great many potential short-term solutions for futureproofing our organization, but all have significant drawbacks. The fundamental problem that they all have is they accept the current software paradigm —that software is fundamentally a manual process. In order to have a truly long-term strategy for futureproofing, organizations are going to have to look to what other engineering disciplines have employed to dramatically improve the development and maintenance of new products. To make the future reality, organizations have to change the way they look at software development. That means two things:

- adopting a real engineering model to software development
- utilizing automatic design and automatic code generation tools

### Adopting a Modern Engineering Model to software development

Engineering has been around in one form or another for hundreds of years, and in some cases, e.g., civil engineering and architecture, for thousands of years. As a result, these disciplines have created strong methods and tools for documenting and implementing their designs. For hundreds of years now there have been standard methods of documenting designs so that others could implement them—in the 20th century, they were called "standard blueprints." These standards, developed over the decades to ensure that what was designed was built exactly as it was designed, greatly benefited from computer power.

Over time, computer software engineers were able to take what draftsman did in creating blueprints and translate drafting in graphic CAD programs. Today, a large portion of modern architecture and engineering begins with CAD tools. The CAD tools have mostly replaced a large number of those that were employed by engineering firms, replacing them with a smaller number of CAD-savvy technicians. Those that are left have to be able to do traditional drafting but more importantly be able to work easily with CAD tools.

But modern engineering and architecture did not stop with just automating the drafting process; that would be equivalent to "drawing programs" like VISIO or PowerPoint. No, modern engineering technology has produced computer aided engineering (CAE) and computer-aided manufacturing (CAM) tools, tools that start with the CAD drawings and then go on to apply to computer-aided intelligence that simulates the object designed and then actually produce it.

The same process is working today in software but this "modern engineering model" has not been as widely adopted as it should be. This modern engineering model involves committing to a few basic principles:

- separating major concerns
- building increased intelligence into tools
- increasing the ability to relate directly with the user
- letting the computer keep track of the details

Modern engineering disciplines *separate major concerns*: requirements, engineering, design, and construction as separate phases. Each of these phases has a clear delineation and clear interfaces. Those interfaces have standard documentation. It is because of the separation of concerns and the clear interfaces that so much progress has been made in computerizing engineering.

Almost from the beginning, engineers saw the potential that the computer brings to their work. They were able to imagine utilizing the computer to manipulate "virtual products" inside the computer itself. They were also able to imagine taking those virtual products and creating real ones. Of course, they had a strong profit motive in reducing the cost of drafting, but the real gains have come from their ability to work more and more closely with their users. By using the computer across the entire spectrum from conceptualization to manufacturing, engineers have been able to dramatically reduce only the cost of developing new products, but they have also been able to dramatically reduce the time involved in going from concept to product, something that has revolutionized markets from automobiles to airplanes, from computers to cellphones.

*Building increased intelligence into tools* means that fewer mistakes get made and things get done at computer speed. Moreover, when changes have to be made, the computer can make them in all the right places because of sophisticated, integrated meta-data.

*Increasing the ability to relate directly with the user* is one of the fundamental characteristics of "smart tools." Very few people, even very smart, very experienced business users in any field can actually visualize what they're going to get. This is the reason that over decades engineers and architects have invested huge amounts of time and money in building models. The more realistic you can make the model of a new building or a new car or a new television set, the easier it is for people to say whether they like it or not when you actually build it. Computerized design tools are universally used today to help the end-user "see" what they're going to get before it is produced.

*Letting the computer keep track of the details* is yet another major principle of modern engineering and architectural design tools. For hundreds of years, for example, architects and engineers have been able to produce realistic, perspective drawings of their products from regular engineering drawings—but developing perspective drawings took a long time. As a result, only one or two of these drawings were routinely produced in an engineering cycle. Today, with computers, it is possible to build realistic three-dimensional objects instantly within the computer and to let users see these "virtual buildings or products" from any position. Indeed, for things like buildings, it is possible to produce "virtual walk-throughs" of those buildings just as if one were viewing a motion picture of a tour through a completed new building.

Software has to be able to do the same thing as engineering and architecture. Software developers must be able to define new software applications on a computer and then instantly let the user see for themselves what software looks like, and how it behaves. And just like other engineering and architectural software, software engineers have to be able to change any aspect of the software requirements and design, and then have the computer instantly redesign and redisplay the modified application. The only way to do this is to rely on automatic design and automatic code generation.

### Utilizing automatic design and automatic code generation

Utilizing automatic design and automatic code generation is the natural extension of computer-aided design software to building software applications. Software technology has become so complex, that programmers, even super programmers, have trouble keeping all the appropriate options in their heads. If organizations are going to futureproof themselves, they need systems that are easy to develop, error free, and easy to maintain. This becomes even more important in the world of Sarbanes-Oxley. Today more than ever before, organizations are going to have to prove that their systems work and that their controls are failsafe. In order to do this they're going to rely more and more on automatic design and automatic code generation.

Now, code generators have been around for decades but they have been limited in most cases because of their limited metadata. *In order for code generators to be truly effective, they must be based on a very rich set of metadata that ties together both data and business rules consistently*. State-of-the-art code generation uses knowledge bases that contain very rich metadata; rich enough to support all of the relations that go into a complex application.

For decades, programmers have argued that the solution to building complex software involves ever more sophisticated and complex languages. Experience has shown that no

language, no operating system, or no database management system will truly solve the complexity problem. Software complexity has to do with knowledge of all the relevant pieces and their relationships to one another. Code generators, the right kind of code generators, work hand-in-hand with automatic design tools that in turn are driven off of rich metadata databases.

Probably the most significant aspect of a true futureproofing solution is the availability of systems-wide and enterprise-wide knowledge bases. Too many tools in software work only at the program level. But, sophisticated programming always has to do with systems and often multiple systems. Designing a single table for a single program is no problem, but designing a database that will work for hundreds (or thousands) of programs is a different matter. The same data element, for example, may exist on many screens in many programs within multiple applications. The same formula, for example, may be executed hundreds of times across thousands of programs. A rich metadata knowledgebase keeps track of all these uses and when changes are made in one usage, it is reflected throughout the system. The same is true with database changes. A rich metadata knowledgebase keeps track of all the relationships as well as the individual attributes, tables and business rules.

A rich metadata knowledgebase is itself a highly sophisticated database that is constantly being updated to reflect the latest changes. Most importantly, when one fact or relationship changes, the knowledgebase must be consistently updated to insure that all the important changes are made. Automatic design and automatic code generation are simply applications, very sophisticated applications that run against a single, consistent knowledgebase.

Slowly, the software industry is becoming aware of this fact. The recent interest in Model Driven Architecture (MDA) is recognition that there must be a tie between the business definitions and the code. To date, MDA is in an early stage where many, if not most, of the major transitions are still done manually. To be a complete solution, MDA must evolve into a true CAD/CAE/CAP/CAM-like software solution.

MDA is a long way off, years, perhaps even decades in the future. However, there are futureproofing solutions that already exist, that have been used to build thousands of very large-scale, multi-platform, multi-database applications. The solution that has had the longest history of success is called GeneXus.

# Combining Business and Technology Knowledge

During the 70s and 80s, the majority of major applications were developed by organizations attempting to utilize technology for their own benefit. These systems were highly customized, often written for mainframe computers and involved implementation of core business processes for these organizations. During the 90s, the trend in applications development turned to purchasing package software rather than building (developing) core applications in house. This trend has created some interesting choices for organizations trying to stay ahead of their competition. If you look closely at figure 1, you will note some of the problems.

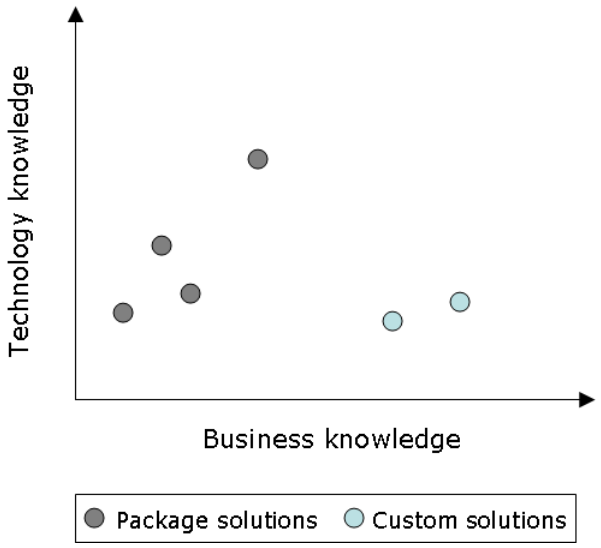


**Figure 1 – Business vs. technology knowledge**

In general, custom applications (the clear dots in figure 1) normally tend to contain superior business knowledge, while package solutions (the grey dots) normally tend to have better, more current technology. In the past, custom developed applications, which had in-depth knowledge of the unique business rules and business processes of an organization, have had problems keeping up with changes in the technology, e.g., relational databases,

Windows, client/server, the Internet, Web services, etc. Package solutions[1], especially newer ones, have just the opposite problem. While they often use the latest technology, they have problems building specific business knowledge into their tools.

As a result of this situation, organizations are caught on the horns of a dilemma: whether to try to build their own custom applications, to make major modifications to their purchased packages to allow them a maximum flexibility as a business, or to acquire purchased software and tailor their business to fit the package. Clearly, in order to effectively futureproof your organization, some solution needs to be found that satisfies both the need to adapt quickly to business changes as well as the need to adapt quickly to technology changes. Because they must support thousands of businesses in hundreds of different markets, package solutions are limited in terms of how much they can adapt to a single organization, or even to a single market.

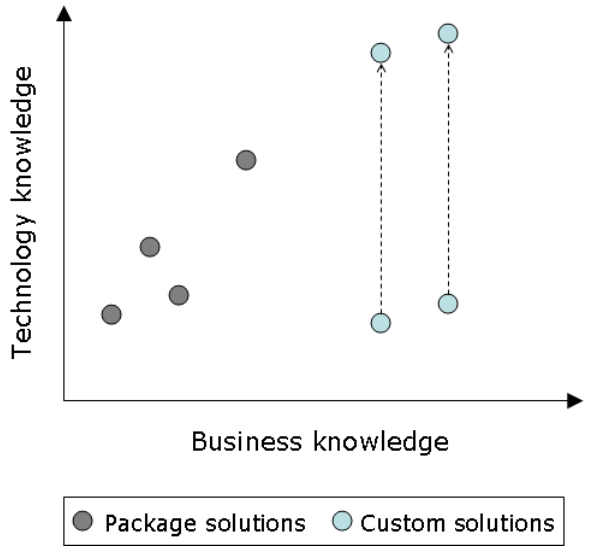


**Figure 2 – Business knowledge vs. technology knowledge (the optimal solution)**

[1] Over the long run, package applications have the same problems with technology, in general, as custom applications—they become wedded to their technology. Successful package companies struggle with new technologies as much, if not more, as their customers. Package companies have invested heavily in new development techniques that promise to minimize the impact of changes in technology, but they have only been somewhat more successful because it is such a major marketing factor. Software package companies cannot afford to appear to have out of date technology.

The optimal solution for organizations would be if they could somehow build custom applications in such a way that they could be adapted effortlessly as both new business needs and new technologies arise (see figure 2). In the long run, true agile development would mean that organizations would be able to have software that was in the upper right hand corner of the business knowledge vs. technology knowledge chart. In order for organizations to become truly agile, they will need to move into this fourth quadrant, the quadrant of "high technology" and "high business functionality." In the next section, we will describe how this is possible today with state-of-the-art tools.

# GeneXus: A true 21st century futureproofing solution

GeneXus is a state-of-the-art systems development environment that has been under development for nearly two decades now. It involves bringing together a number of advanced technologies including: Artificial Intelligence, automatic database design, user interface design and business rule design. As a result of its breakthrough integration of key technologies, GeneXus has been able to evolve software development environment that has allowed organizations to go from mainframes to client/server to web-based, service oriented architecture while maintaining and updating their critical business knowledge. This string of success is based on some very important insights.

- The first of these major insights was that it is possible to deduce an efficient, normalized database design structure from a set of data structure descriptions of individual business transactions and to generate automatically the navigation necessary to run against this database structure.
- The second major insight was that it is possible, when changes are made to the specifications of the data structures above, or new data structures are added, to re-design and re-normalize the database, convert the old database structure to the new and then regenerate all the code the operates against that new database.[2]

[2] Developers known for years that the database design was ultimately the most difficult thing in any system to change. The reason is that for most development environments, once programmers began writing code against a given database design, the cost of changing that database design went up dramatically because of all the database navigation code that would have to be rewritten. GeneXus solves this problem by generating all the navigation code itself.

- The third major insight was that this process could be carried out in all major languages and all major commercial relational databases by developing a common generation engine and tailoring each set of generators to a given platform.
- The fourth major insight was that any system that would truly be able to keep up with both business changes and technology changes would have to generate 100% of the code for most applications.

By adopting this radical strategy, GeneXus' researchers and developers were able to create a toolset that operates just as CAD/CAE/CAM software does in engineering and architecture. As a result, organizations using GeneXus have been able to create working applications almost instantly, in minutes or seconds as opposed to days and weeks. And, more importantly, they have been able to revise these applications just as quickly. Long before anyone had given a name to agile development, GeneXus developers have been doing it.

But, the proof of any technology is in its results, not its claims. Today, there are over 30,000 Genexus licenses worldwide generating an estimated more than 2 billion lines of code per year on a wide variety of platforms and databases. The average application has the following characteristics:

- Nearly 100% automatic generation
- between 1 and 5 million lines of generated code
- the ability to handle more than 500 relational tables in a single application
- migration from one platform to another with minimal effort

Software development has turned out to be a much more complicated activity that anyone would have guessed in the early days. While programs and program development have received enormous attention over the years, far less attention has been paid to the development of integrated systems. It has been assumed that systems are simple collections of individual programs. But systems are much more complicated than individual programs and they rest, ultimately, on a common set of data and business rules. It is rare for a program to be more than 20 thousand lines of code long, but it is not at all rare for system to include 10 or even 20 million lines of code today.

Historically, systems developers have attempted to solve the systems development problem through an endless variety of project-management schemes. This meant that an enormous amount of time had to be spent on systems and database design, and on systems interfaces, so that individual programs could be written independently and “systems integrity” maintained.

GeneXus has been enormously successful in its approach to code generation because it treats systems as the basic unit of work. A given knowledge base, for example, includes information about all the data in a given application (system). Because of this, the code generator can be much more intelligent in the way it generates code, and more importantly in the way it treats and updates data.

# The Future of Futureproofing

The business world today is as volatile as it has ever been. Globalization, intense competition, new technologies have all contributed to an urgent need for organizations to become more and more agile. Futureproofing has two major elements: (1) more serious thinking about the future and the impact of current decisions on the future, and (2) acquiring tools and technology that allow the organization the maximum flexibility.

ARTech, the developers of GeneXus, are specialists in software futureproofing. Nearly two decades ago, ARTech saw the need for an entirely different kind of software development toolset, one in which software developers could spend the majority of their time focusing on new business capabilities rather than spending the majority of their time on technology changes. ARTech also saw that over the long haul, systems that allowed incremental (agile) development would be much more productive than those that relied on traditional waterfall approaches.

To its credit, Artech has also been ahead of the curve, all along, in recognizing the importance of integrating each new technology, such as workflow and wireless, as they came along, keeping up with technological change. By thinking about the future, ARTech has made "futureproofing" their principal business. As a result, ARTech has produced a toolset that business around the world can use to leverage technology, outpace their competitors and truly "futureproof their organizations."

*"Probably, in the next decade or two, other Futureproofing solutions will appear.*

*But GeneXus is ready and available today, and it can show us the way today."*

Breogán Gonda, Founder and President, ARTech