Peeling the Knowledge Onion:
Development of a Bond Tool Operation Using VB Automation

Design Creation and Validation
Product Development

Collaborative Product
Development Associates, LLC
222 Grace Church Street
Port Chester, NY 10573
(800) 573-4756
www.cpd-associates.com

August 2004

CPDA: Collaborative Product Development Associates, LLC

CPDA's Product Lifecycle Management (PLM) research programs target the critical decisions in Product Lifecycle Management challenging Design, Engineering, Manufacturing, and Information Technology managers and executives. CPDA's PLM collaborative research programs provide in-depth analysis of strategies, products, issues, processes, technologies, trends, case studies, and surveys for assessing technology, business goals and objectives, and implementation road maps.

The cohesive suite of collaborative programs clarify and evaluate new capabilities, frameworks standards, and development issues; they highlight the most advanced uses of leading technologies, and they link the technical effort to the realization of business value. The four collaborative research programs include:

- ***Design Creation and Validation:*** A bottom-up view of engineering requirements from the desktop across the enterprise.
- ***Product Definition:*** A top-down view providing a conceptual framework for tightly coupled collaboration across different product development perspectives, bridging Customer Needs, Systems Engineering and Tradeoffs, Design Solutions, and Fulfillment and Manufacturing.
- ***Product Value Management:*** Mapping business process modeling to an IT integration infrastructure in a Federated Enterprise Reference Architecture (FERA)™ loosely couples multiple touch points within and between enterprises extending across the supply chain.
- ***PLM Infrastructure:*** Integration and interoperability in complex PLM environments pose substantial hurdles. The rapid transition to cross-enterprise collaboration, at all levels of design and supply, intensifies the pressure on existing, inwardly focused IT architectures to support and enable new modes of doing business.

Collaborative Product Development Associates was formed by the PLM research team of D.H. Brown Associates, Inc. (DHBA). CPDA now supports and fulfills the PLM research services of D.H. Brown Associates.

Peeling the Knowledge Onion:
Development of a Bond Tool Operation Using VB Automation

Gregory A. Premetz, Technical Project Manager,
Product Design – KBE Applications,
Vought Aircraft Industries, Inc.

EXECUTIVE SUMMARY

Gregory A. Premetz, the Technical Project Manager of the KBE Applications Group responsible for the development and implementation of Knowledge Based Engineering for Vought Aircraft Industries, as well as for the investigation into Knowledge Management, shared the Vought team's experience in the development of a bond tool operation using Visual Basic (VB) automation at the Product Lifecycle Management Road Map™ conference. Their growth in understanding best practices and pitfalls encountered during the application development process is a story called "The Tale of the Knowledge Onion."

Knowledge-based engineering (KBE) is the use of advanced software techniques to capture and reuse product and process knowledge in an integrated way. In the context of CAD there are actually two approaches for applying KBE. The first, a process-centric approach, relies on rules-based heuristics to automate or define the product. The second, a CAD-centric method, employs parametrics rather than rules. The parametric alternative is much simpler, and ultimately not as powerful as the process-centric approach.

Rule-based engineering requires three basic components – rules, objects, and process modeling. Rules, or heuristics, represent most of the "active" intelligence within the application. An object is a self-contained entity that consists of both discrete data and procedures to manipulate that data. Process modeling represents the "passive" intelligence in the application, and involves describing the flow of information in the context of business processes. The application uses the process interactions within the model to map out an implementation framework for the rules. Understanding all three components – rules, objects, and process modeling – is essential for generating an application.

KBE tools apply best to a well-defined process. Applying KBE while designing a process will likely fail; rather it should be used for simply automating what is currently known. In an ill-defined process, you cannot automate what you do not know. Also, KBE does not fare well when there is a judgmental process involved, artistic in nature. As of yet, KBE cannot create new processes or make subjective decisions based on unexpected information combined with experience.

A major goal of knowledge-based engineering is the capture and reuse of corporate knowledge through automation. This drives two steep learning curves – how to acquire the knowledge, and how to codify the knowledge. Knowledge acquisition includes knowledge mapping, process modeling, and managing the requirements. After capturing or acquiring knowledge, you need to convert the knowledge from a form understood by humans into a form that can be understood by the computer. Another aspect of codifying that needs to be learned is when to stop. Due to obvious budget, time, and resource constraints, you must balance between "perfection" and what can be considered good enough.

There is a real danger in KBE because the KBE application in itself does not represent a viable goal. Once a KB (knowledge-based) organization is created, it is often assumed the product of that organization is the KBE application. The application becomes the goal, rather than the product itself. Absolutely wrong. The product is the goal. The product and the process of creating the product, the optimal design process, represent the goal.

The automation of PLM and/or the optimal design process allows more innovation, and innovation should be the second fundamental goal of KBE. If the only goal is to automate, to take a process from thirty hours down to ten minutes, then the real goal has been missed. KBE saves real money, but does nothing for the product. The time savings generated from the use of KBE should be used to innovate, to create new and better products. Understanding these two items as the objective provides companies more mileage out of KBE.

Incorporating knowledge requires learning a unique skill set – a "knowledge engineer" skill set. Automate the well known. Political land mines and/or antiquated processes are sometimes best left alone. Ask the right questions. Over time you will learn to create a "logic tree" while interviewing the subject-matter experts. You will follow that logic and see holes in the tapestry, or missing steps, and you will fill them with the right questions. Also, maintain an ongoing communication with experts during the development process to continually capture hidden knowledge.

Knowledge engineers must understand the entire process, and need to identify all requirements. Any missing knowledge will impact the outcome of an application. Removing barriers to innovation requires getting the experts to agree and work collectively towards the best process, hopefully all within the constraints of existing procedures.

Finally, there are elements of knowledge management (KM) in KBE. Knowledge management is the conscious strategy of getting the right knowledge, and the right people, at the right time, and helping people share, and put information or knowledge into action, in ways that will improve organizational performance. A KBE application contains knowledge that will need to be managed. Managing knowledge deals with truth maintenance (testing), knowledge half-life (new knowledge is continually created replacing the knowledge in the application), and additional concepts that will improve the application like knowledge acquisition, knowledge mapping, complexity theory, semantics, and ontology.

TABLE OF CONTENTS

Peeling the Knowledge Onion: Development of a Bond Tool Operation Using VB Automation

INTRODUCTION

My team at Vought Aircraft has gone far with the tools we work with in KBE, and I would like to present that journey this morning. In the presentation, I will provide a brief description of my company to help understand our context; I will show you what a bond tool is, and I will review the metrics to illustrate the value we found in the KB application. You may be familiar with Steve Denning, and his book called *The Springboard*.[1] A former World Bank executive, he professes that you should use a story to transfer knowledge and to get your company to rally around your cause. At Vought, we developed our own KB (knowledge-based) process, and using Steve's suggestion, I will present that development process in a story called "The Tale of the Knowledge Onion."

You may not be familiar with Vought Aircraft Industries, but I'm sure you are familiar with our products. The Vought name extends back to the military aircraft company founded by aviation pioneer Chance Milton Vought. In 1917, with Birdseye B. Lewis, Vought organized the Lewis & Vought Corporation. Among the more than 15,000 aircraft produced by Chance Vought's legacy companies, some notable ones include the VE-7 Bluebird, the OS2U Kingfisher, the F4U Corsair, the F-8 Crusader, and the A-7 Corsair II.

Over the years, the company went through various ownership changes. Some of the heritage companies with ties to Chance Vought's original company include Vought-Sikorsky, Ling-Temco-Vought, and LTV Aerospace. In 1994, Northrop Grumman purchased Vought Aircraft Company. In 2000, Northrop Grumman sold the majority of its aerostructures business to the Carlyle Group, which established Vought Aircraft Industries as an independent company.

Today, Vought Aircraft Industries is a major subcontracting partner on many commercial and military aircraft programs. The company offers a full range of design, testing, manufacturing, and support capabilities. Vought is positioned as a Tier I Integrator – filling the gap between prime contractors and traditional subcontractors by providing large, complex aerostructures on a turnkey basis.

What do those assemblies look like? Figure 1 illustrates some of our products, which are highlighted in blue and green. Our customers include Airbus, Boeing, Hawker; G5 is Gulfstream Citation.

[1] Denning, Stephen. *The Springboard: How Storytelling Ignites Action in Knowledge-Era Organizations*. Butterworth-Heinemann, Woburn, Mass., 2001.

FIGURE 1
Vought Aircraft
Commercial Products

Courtesy Vought Aircraft Industries, Inc.

Figure 2 displays our military products, C17, C5. Our facilities span the country.

FIGURE 2
Vought Aircraft
Military Products

Courtesy Vought Aircraft Industries, Inc.

What is knowledge-based engineering? Knowledge-based engineering (KBE) is the use of advanced software techniques to capture and reuse product and process knowledge in an integrated way. In the context of CAD there are actually two approaches for applying KBE. The first, a process-centric approach, relies on rules-based heuristics to automate or define the product. The second, a CAD-centric method, employs parametrics rather than rules. The parametric alternative is much simpler, and ultimately not as powerful as the process-centric approach.

KBE BASICS (CAD RELATED)

Figure 3 illustrates one of our rules-based applications, a form block. Rule-based engineering requires three basic components – rules, objects, and process modeling. Rules, or heuristics, represent most of the "active" intelligence within the application. They are primarily conditional and logic statements that define the outcome. An object is a self-contained entity that consists of both discrete data and procedures to manipulate that data. It is difficult to write code or applications for KBE without objects. By defining their functions and characteristics, they are able to carry their own "intelligence" as they move throughout the code, passing in and out of various sub-routines as bits of information rather than just data. Process modeling represents the "passive" intelligence in the application, and involves describing the flow of information in the context of business processes. The application uses the process interactions within the model to map out an implementation framework for the rules. Understanding all three components – rules, objects, and process modeling – is essential for generating an application.

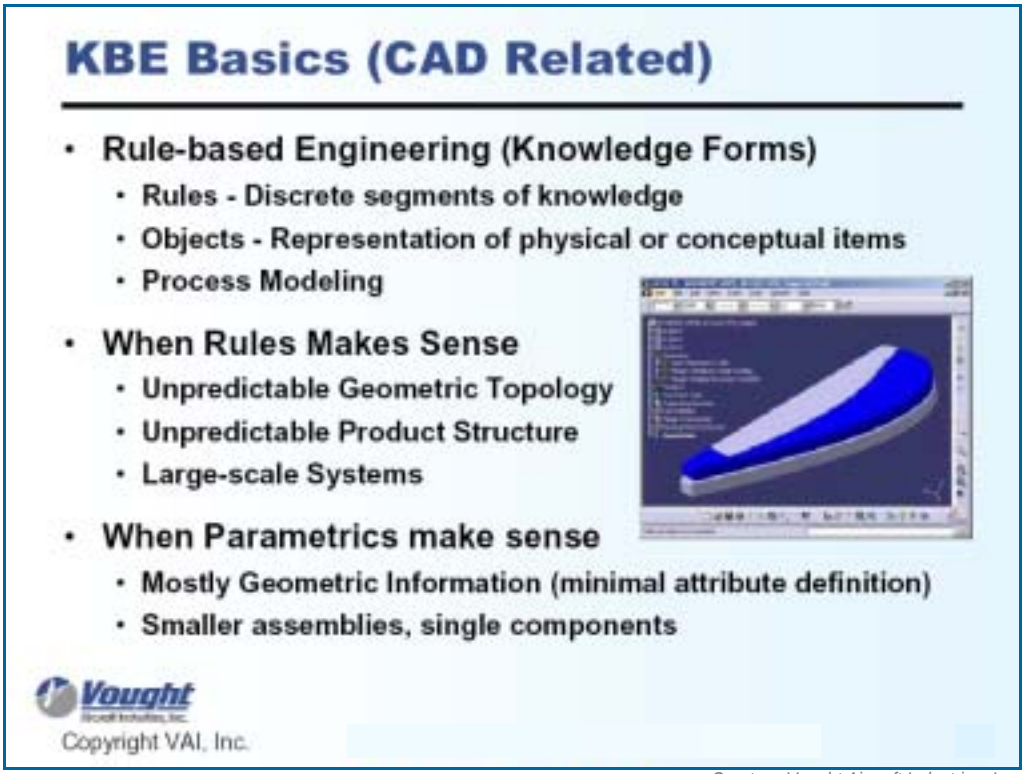


FIGURE 3
KBE Basics
(CAD Related)

Courtesy Vought Aircraft Industries, Inc.

When do rules make sense? When you are faced with unpredictable geometric topology or large-scale systems. Unpredictable topology requires numerous rules to accommodate all of the possible outcomes, and applications that encompass multiple systems increase the level of complexity in the process modeling part of the code. These fall well out of the scope of simple parametrics. Parametrics make sense when you just have geometric information. You can automate geometry quite easily within CAD. It does not make sense to write code when the assemblies are small or uncomplicated. It is more cost effective to let simple parts morph themselves using parametrics or auto-generate themselves with "intelligence" contained in the CAD model.

KBE BASICS (IN GENERAL)

In general, where is KBE good? KBE is good in a well-defined process. I cannot emphasize this point enough. If you do not have a well-defined process, you do not know what you are going to automate. You do not want to be doing KBE while designing a process, rather it should be used for simply automating what you currently know. To get a sophisticated integration between many parts, you need a deep understanding of that integration provided by a well-defined process. Obviously, when the knowledge is well defined, KBE can really shine.

Where is KBE not good? In the ill-defined process; you cannot automate what you do not know. Another place KBE does not fare well is when there is a judgmental process involved, artistic in nature. As of yet, KBE cannot create new processes or make subjective decisions based on unexpected information combined with experience. For example, there are still many manufacturing processes that depend as much on "the feel of the wheel" as they do on the process specifications. On the other hand, many times the shop floor or other functional groups will assert that automation in impossible because their process is highly judgmental. Even though their processes are well defined, they tell us the results will always differ and ultimately depend upon their "expert" manipulation of that process.

Through further investigation (mapping the knowledge, flowcharting the process, and creating a preliminary logic tree), it becomes apparent that the process is a good candidate for automation. As it turns out, they just want to retain the judgmental part of the process for a perceived improvement in job security. With improvements in machinery and our ability to capture repetitive processes, allowing judgment to remain in a process that does not need it will result in instability and variation. Therefore you do not attempt to automate the process that took the individual five years to learn, rather target the job that would be given to the new hire.

You do need to be sensitive to the apprehensions of the workers because ultimately they will be your subject-matter experts. Automation should be used to free people up for other work that is more productive – getting stuck with low productivity does not add to job security. And it does not necessarily result in a

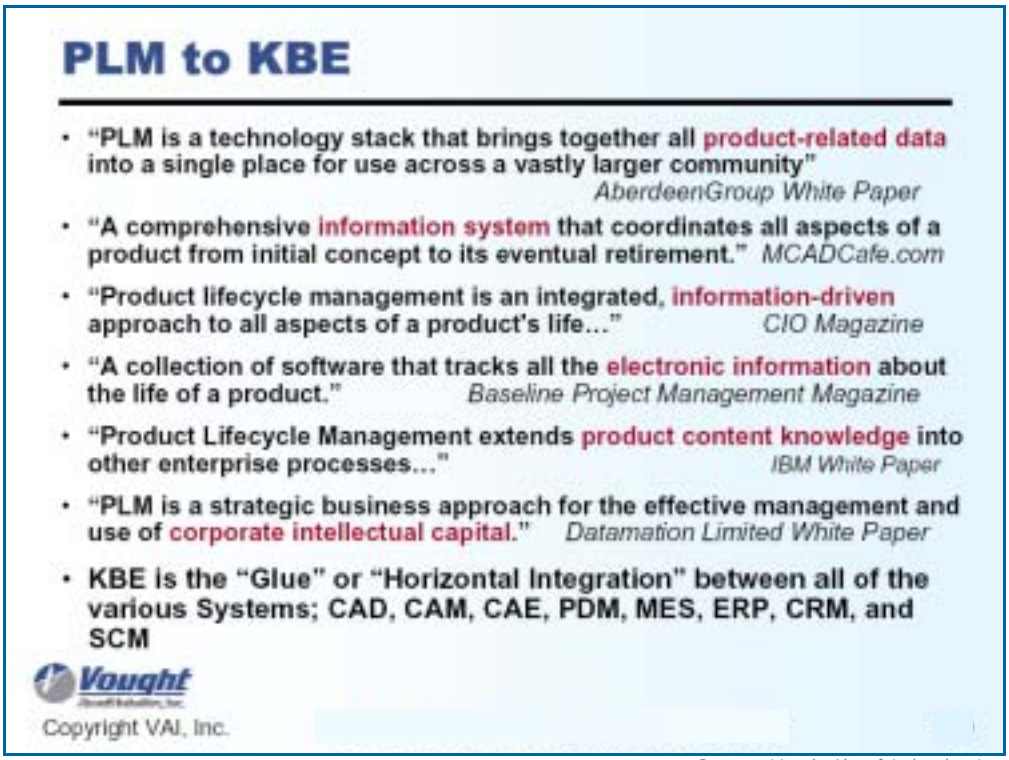
loss of job – we all move around. Bottom line, it is important to automate processes to remain competitive.

Where is KBE overkill? When other technologies can create the product definition faster and maintain it more easily, there is no sense to do KBE. Today's CAD will record your every function into a macro which can easily be distributed and maintained. With parametrics, you only need a spreadsheet to quickly morph part features or create an entire family of parts. You do not want to spend a lot of money developing code when a simpler method can produce the same results.

PLM TO KBE

Given this is a PLM Road Map Conference, Figure 4 is my attempt to better frame KBE within the context of PLM. These are different definitions of PLM I took from the Internet. Through similar language, they all refer to the concepts of managing and integrating something. That something has been defined as *product-related data*, *information*, *knowledge*, and finally *intellectual capital*. I organized the definitions from data to knowledge as a segue to the next slide and to better illustrate where KBE can influence PLM. What can you make of the fact that to some PLM is nothing more than data, but to others it is actual knowledge?

FIGURE 4
PLM to KBE



Courtesy Vought Aircraft Industries, Inc.

I look at the functional specialties as islands of automation. You have a CAD island; it can be automated. You can automate the CAE and the CAM, as well as non-engineering islands such as ERP, SCM, and CRM. But what pulls them all together? I am not talking about the seamless movement of data or information

from one system to another. I am talking about intelligent products or objects that know which island to contact next and when. They autonomously complete all of the repetitive non-judgmental tasks on that island and then they either contact the owner for input or move on to the next island. Ultimately it will be some Knowledge-based application that creates these links. Whether it is KBE or KBM, the acronym of Knowledge-based engineering or management, is not important; it will ultimately be knowledge based in nature.

TRANSITION FROM DATA TO KNOWLEDGE

In order to speak about KBE, you need to have a common understanding of knowledge. Figure 5 is based on the work of Thomas Davenport and David Prusak,[2] who identified key attributes that distinguish the difference between data, information, and knowledge. I took these attributes, added some of my own, and then tied them to an aerospace example to provide my company with a simple illustration of the migration from data, to information, to knowledge. Since each has its own unique set of attributes, if you can recognize which one you are working with (data, information, or knowledge), then you have a better understanding of how to automate it, and a better understanding of its lifecycle.

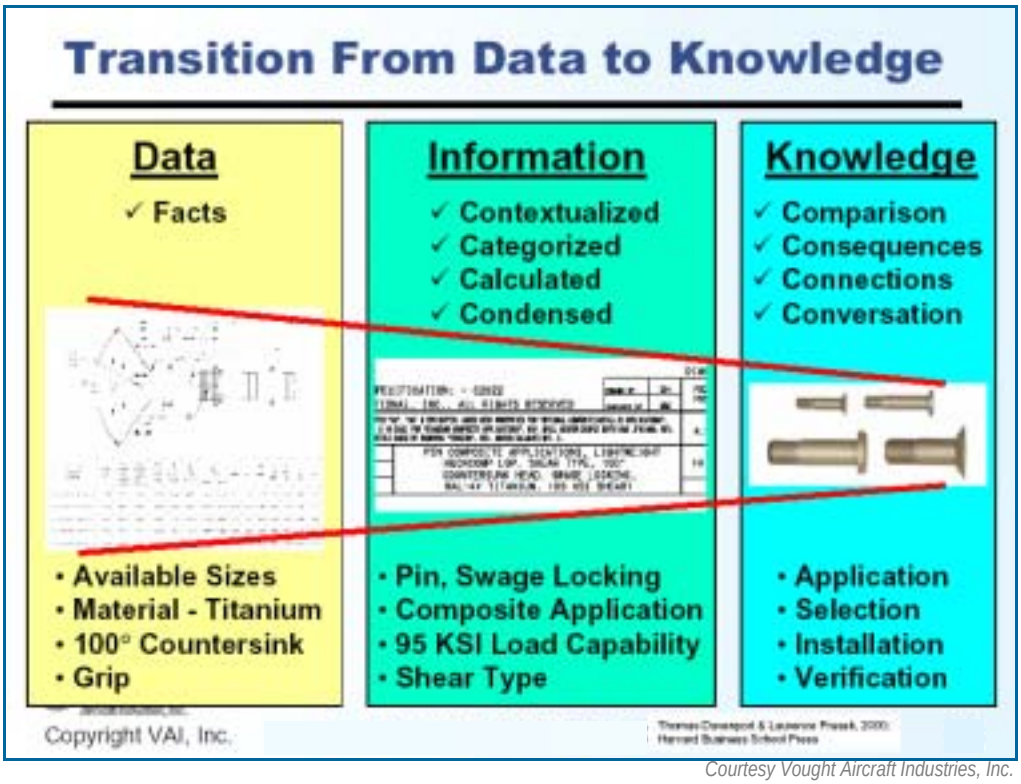


FIGURE 5
Transition from DATA To KNOWLEDGE

Courtesy Vought Aircraft Industries, Inc.

The first column, Data, simply presents *facts*. It is a table of facts, numbers, available sizes on a particular fastener, its material; nothing more than facts. Once

[2] Davenport, Thomas H. and Prusak, Laurence. *Working Knowledge: How Organizations Manage What They Know*. Harvard Business School Publishing, May 2000.

I *calculate* that data and put it in *context* relative to other data, and then *condense* it for my particular purpose, and finally *categorize* it for re-use, I have transitioned to information. I can now describe the data as a 95 KSI load, swage-locking, pin that is used in shear for composites. This is information I would not have known without going through the transition process of converting the data into information. When I talk to another person about that pin, I have in-depth information to describe its function, capability, and application. To someone looking at the graphic, they could conclude that it is a screw, but when I convey information, the one thousand words that are spoken by the picture are condensed into a specific definition.

How do I get to knowledge? To transition from information to knowledge, I do *comparisons* between different sets of information and prior experience; I broaden that experience through *conversations* with others, and I then make *connections* and determine *consequences*. In essence, the process takes information and provides another level of context. This process of identifying relationships and drawing conclusions (typically outside of a vacuum) is the creation of knowledge.

Back to our example, I now know what to do with the pin, as well as what not to do. I know what to expect and how to install that pin, that perhaps I need to use a sealant, a certain force, or a particular tool. This is more than information; it is knowledge. Through this process, you start with lots of data, funnel down to information, and then arrive at knowledge. It is really a funneling process.

BUILDING A FOUNDATION

A major goal of knowledge-based engineering is the capture and reuse of corporate knowledge through automation. This drives two steep learning curves – how to acquire the knowledge, and how to codify the knowledge.

You have to learn how to acquire the knowledge, which is a skill set in itself. I no longer just call my team "engineers," rather I call them "knowledge engineers," because they now have the skills on how to acquire knowledge. Knowledge engineers must know how to quickly gain a subject matter expert's trust and push them to provide knowledge rather than just information. They need to know how to extract only relevant knowledge and how to select the right knowledge when subject-matter experts disagree. Knowledge acquisition includes knowledge mapping, process modeling, and managing the requirements.

You must also learn to codify. After capturing or acquiring knowledge, the engineer will need to convert the knowledge from a form understood by humans into a form that can be understood by the computer. We typically do this through the creation of rules, which can look radically different as you move from one program language to another. Learning which tool to use, whether it is Visual Basic (VB), or C++, or LISP, or a macro, and how to use it is included in this learning curve. We started KBE by using Visual Basic, which is easier than most languages. This allowed us to spend time learning how to acquire knowledge at

the same time. We have plans to migrate to more complex object-oriented languages as our needs require more sophisticated applications.

The question comes up as to when the computer science staff should codify the knowledge, versus when engineers should be trained in programming to codify knowledge? There is no escaping the programming aspect of this job, so it becomes a difficult choice. I have seen different companies go either way. We use our engineers, because they can speak the language of the subject-matter experts. We feel it is easier for them to capture and codify the knowledge from other engineers because they share the same experience. It can be difficult to train them how to program and we may suffer from less efficiency, but the programming does not matter as much in engineering as capturing the right knowledge and having it applied correctly.

There is another aspect of codifying that needs to learned, and that is when to stop. Due to obvious budget, time, and resource constraints, you must balance between "perfection" and what can be considered good enough. This has always been an issue, even in design. Engineers want to go the extra mile and sometimes that is neither practical nor affordable. In a perfect world, we would follow each step for application development, make sure all relevant knowledge is captured, have selected the best language or tool, and have optimized the code. You cannot always fulfill all of these requirements, particularly when you talk about incorporating relevant old and new knowledge. That is a difficult target to hit the first time out.

FOCUSING ON PRODUCT AND PROCESS GOALS

There is a real danger in KBE – a danger in using CAD, for that matter – because the KBE application in itself does not represent a viable goal. People lose sight of that. Once they create a KB organization, they assume the product of that organization is the application. The application becomes the goal, rather than the product itself. Absolutely wrong, just like the CAD-generated model is not the goal either.

It was not long after our first CAD workstations were installed that our upper management started to look at the CAD-generated models as the product of engineering. They would create metrics based on how many models were being produced per week. The 3-D digital representation is not the goal. Nor is the best code in the world the goal. The product is the goal. The product and the process of creating the product, the optimal design process, represent the goal.

So for KBE, the automation of the PLM process becomes the goal, because at its core ideology, PLM professes to be a product-centric environment. KBE applications only improve that process by automating the redundant activities so that the important human tasks can be done better and faster. But the minute you do KBE you run the risk of losing sight of the real goal and focusing on the application or the technology, rather than on the product.

The automation of PLM and/or the optimal design process allows me to innovate more. This should be the second fundamental goal of KBE. Again, if our only goal is to automate, to take a process from thirty hours down to ten minutes, then we have missed the real goal. I have saved the company some real money, but have done nothing for the product. I should take that time savings and use it to innovate, to create new and better products. If you understand these two items as the objective, you can get a lot more mileage out of KBE than if you simply aim for the application.

KBE APPLICATION DEVELOPMENT PROCESS

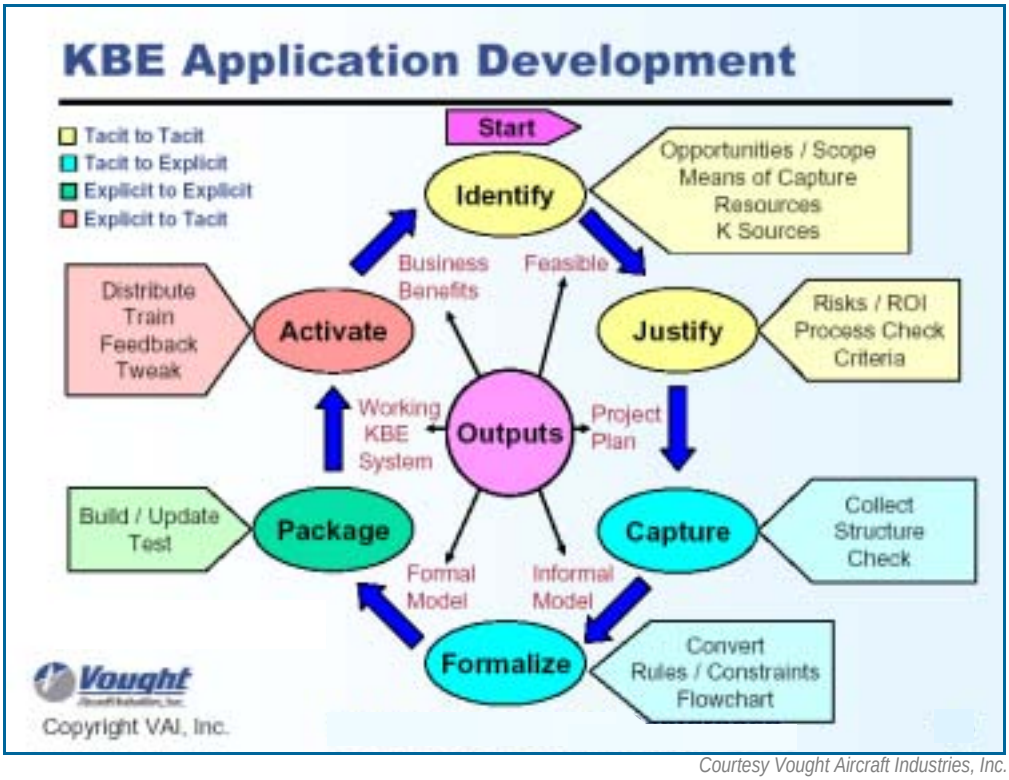
It is widely accepted in the Knowledge Management community that there are two kinds of knowledge, *tacit* and *explicit*. Tacit is typically described as the knowledge in a person's head, the *know-how*, while explicit knowledge is anything that can be documented, archived, and codified, or the *know-that*. As we create knowledge, we cycle through four separate transition states where we either convert tacit knowledge to tacit-or-explicit, or convert explicit knowledge to tacit-or-explicit.

Ikujiro Nonaka and Hirotaka Takeuchi put forth this idea in their book, *The Knowledge-Creating Company*.[3] The authors called converting tacit to explicit knowledge "externalization," and defined it as *doing it and then describing it.* The next stage is explicit-to-explicit, and is referred to as "combination." It is defined as *finding it and combining it*. The next stage is explicit-to-tacit, referred to as "internalization," and is defined as *hearing it and believing it.* The last stage is tacit-to-tacit, called "socialization," and is defined as *watching it and then doing it*. Then the cycle starts all over again. I provide this background information because the definitions of each stage come very close to describing the activities in the KBE application development process.

The process illustrated in Figure 6 (on the following page) is one we had developed internally, but we have found that most everyone doing KBE falls upon the same process. I color-coded the figure so that you can link a knowledge-creating activity to a corresponding step in the KBE development process. As you go around the outside perimeter, you will see the specific KBE activities associated with each step, but deepen the meaning of these activities by applying the knowledge creation descriptions at the same time. On the inside perimeter, I show the outputs between each step.

[3] Nonaka, Ikujiro and Takeuchi, Hirotaka. *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press, May 1995.

FIGURE 6
KBE Application Development



Courtesy Vought Aircraft Industries, Inc.

Briefly going through the process, I would strongly recommend that you start by doing the ROI, to determine if it is worth writing the application. At this stage you would determine whether or not there is a well-defined process, how you would do it (whether in parametrics or with code), and the magnitude of risk. Then you need to collect all the data by capturing all the information and knowledge from your subject-matter experts. The next step is to formalize the information you have gathered into a state the computer can use, and then to package it by writing the actual code, and then activate the system by deploying to the users. Then the cycle starts all over again.

I will not pretend this is how the process takes place in real life. Often, if we identify a hot project, we will jump right to the *Capture* phase, or to a formulized *Package*. Sometimes, the engineer will get really excited and go right from *Identify* to *Package*. Then we have to come back and *Capture* and *Formalize* later. But the figure does represent the most optimized sequence of events.

KBE APPLICATION SELECTION

How do you select a process for automation? Boeing taught us to select the most redundant activity, the one you would give the first year college graduate. This is typically the activity that the experienced engineer hates to do because it is repetitious, procedural, and mundane. Although this sounds like we are maligning the graduate, we all have to start somewhere. And this becomes a great place to start KBE because it is usually easy to automate. Consider this the low hanging fruit.

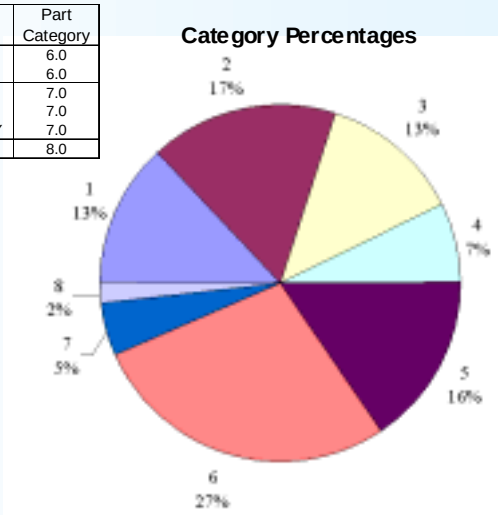
One of the places that we started this process was with a wing. We broke it down and created a taxonomy of it. The taxonomy grouped the parts into categories of similar configuration. Similar configurations were based on how the parts are made. Our resulting taxonomy is illustrated as a pie chart in Figure 7.

FIGURE 7
KBE Application Selection

KBE Application Selection

- **Select Most Redundant & Easy to Automate**

Nomenclature	Part Category	Nomenclature	Part Category
SHEAR TIE	1.0	STIFFENER	6.0
TEE	1.0	RIB POST	6.0
STRAP	2.0	CORE ASSY	7.0
WEB	2.0	CORE KIT	7.0
FACE SHEET	2.0	BONDED WEB ASSY	7.0
PLATE	2.0	SPLICE PLATE	8.0
DOUBLER	2.0		
SKIN	2.0		
SKIN PANEL	2.0		
CHORD	3.0		
STRINGER	3.0		
FILLER	4.0		
SHIM	4.0		
ANGLE	5.0		
CLIP	5.0		
TENSION CLIP	5.0		
ZEE	5.0		
BRACKET	5.0		
GUSSET	5.0		
SILL	5.0		
CHANNEL	5.0		
SUPPORT	5.0		



Vought

Copyright VAI, Inc.

Courtesy Vought Aircraft Industries, Inc.

We looked for the largest number parts in any one category, which happens to be Category 6, for the stiffeners. Stiffeners, rib posts, and stringers all share design similarities that allow us to create any one of them with the same application. In other words, this application will create twenty-seven percent of the wing parts and, given the parts follow a well defined and redundant process, is clearly worthy of automation.

COMPOSITE BOND TOOL

Figure 8 (on the next page) shows a composite bond tool. We selected the bond tool for this presentation because it is made with both rules and parametrics. The top half of it uses Visual Basic to create the geometry, and the base, or bottom half, is created using templates that are supported by parametrics. Our current estimate for creating the top of the tool would be three weeks for one man, in a V4 environment. The time improves with V5, but certainly not to the degree we would like. Note that the design of the tool is well documented and covers ninety percent of the situations encountered by the tool designer. The base is not included in that estimate, because it is simply provided as a rectangular volume, which the shop creates itself by using a design in a shop standard. Through automation we have reduced the process to fifteen minutes from three weeks,

including the base. We realized an additional benefit after our data was transmitted to the shop. They found a substantial timesaving in using our data because every model was standardized from tool to tool. They no longer had to figure out an individual engineer's style as they interpreted the design intent or find data at different locations within the model.

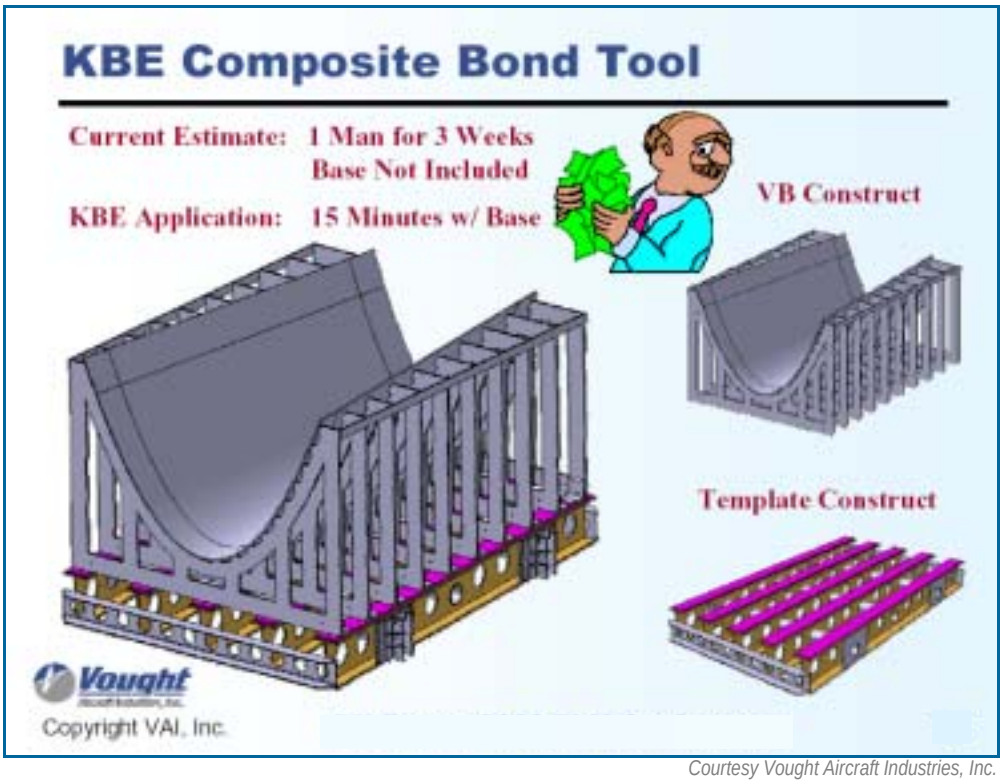


FIGURE 8
KBE Composite Bond Tool

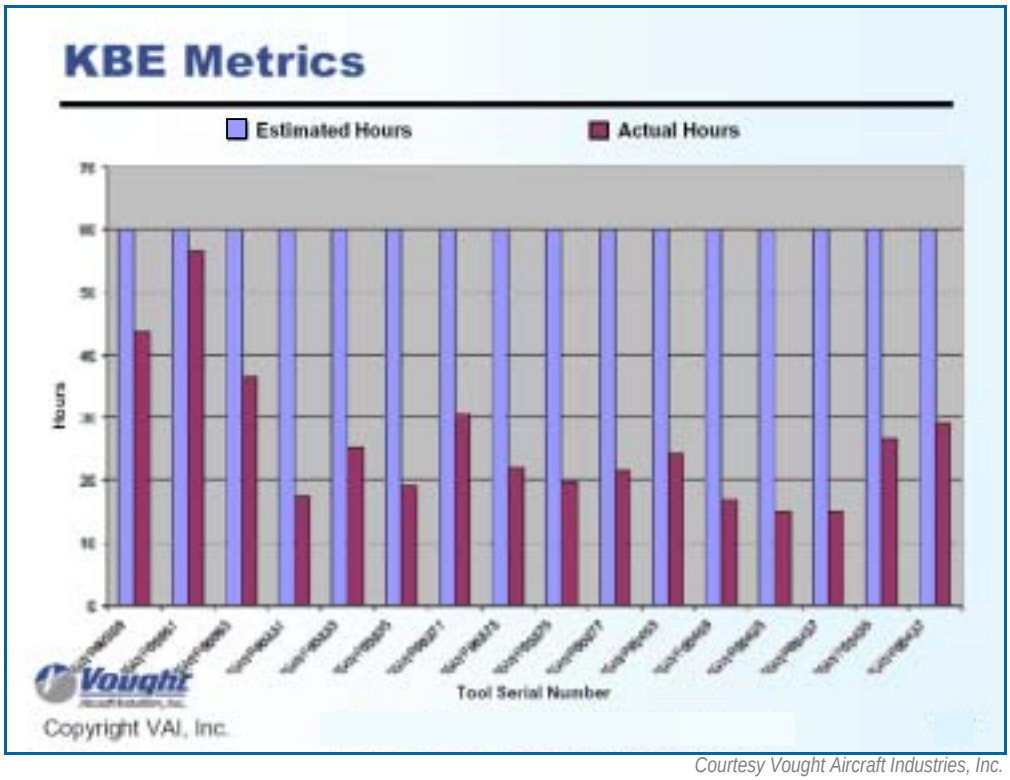
Courtesy Vought Aircraft Industries, Inc.

Remember my caveat that the application is not the goal. After we got the process down to fifteen minutes, we approached upper management and asked for an additional four weeks. We felt we could get the process down to ten minutes. We learned a lot and had greatly improved our object-oriented programming skills, and we wanted to go back and add all of the "bells and whistles." But as far as upper management was concerned, they had just won the lottery and did not care about the other five minutes. They were looking towards the next product and its KBE application where three weeks could be reduced to fifteen minutes. For the rest of us, it was easy to get wrapped up in the application and lose sight of the real goal.

KBE METRICS

Figure 9 (on the next page) shows some of our metrics for the bond tools on one of our programs. It was a quick program, so the tooling department lacked an opportunity to examine each tool and properly estimate the design time to completion. As a result, they estimated a time of sixty hours for each tool across the board. It was not a realistic estimate, because a manually created tool can range anywhere from 40 to 240 hours, but they threw 60 hours out as a target to meet our non-recurring cost requirements.

FIGURE 9
KBE Metrics



Courtesy Vought Aircraft Industries, Inc.

As the figure shows, when we started using the bond tool application for the first time in production, the actual hours were almost as high as the estimates. This was mostly due to some bugs and working out the process of incorporating KBE-generated models into our system. As these issues were worked out, the hours came down. The spike for the seventh tool was caused by its large size. We had to add rules to accommodate splices and special headers. The spikes for the last two tools were caused by additional knowledge that had to be incorporated into the bond tool application. These tools were a particular configuration that we had not seen before.

Some of you might be wondering how we can say that we went from three weeks to fifteen minutes and then show metrics averaging twenty to thirty hours upon completion. This is because the program we were supporting required 2D drawings, as well as models, and the three weeks considered only modeling time. Also there are portions of the model that are very difficult to automate, like tooling balls, so they are put on manually after the KBE application is complete. So a good lesson learned is the eighty-twenty rule, where you can typically automate eighty percent of the completed product at a very reasonable development cost. But attacking the last twenty percent could be costly, and usually can only be justified with high volume or special circumstances. When you consider that realistic estimates for the tools I am showing was between one hundred and fifty to two hundred hours, then obtaining the twenty to thirty hours through automating eighty percent of the process starts to make sense.

THE TALE OF THE KNOWLEDGE ONION

I would like to share a story called "The Tale of the Knowledge Onion" (see Figure 10). It is a story in metaphor for our experience implementing KBE. It starts with the purchase of new tools to do KBE; everybody is happy, because we have brand new technology, which causes great excitement. It is like buying a new shiny and smooth onion from the grocery store. It looks and feels like a baseball, and who doesn't like baseball? We learn Visual Basic, C++, the interfaces, and we pick a project. We select a well-defined process. We talk to all our subject-matter experts, those we call the *gray beards.* We start writing code. This is a good time, a fun time in the KB application process.

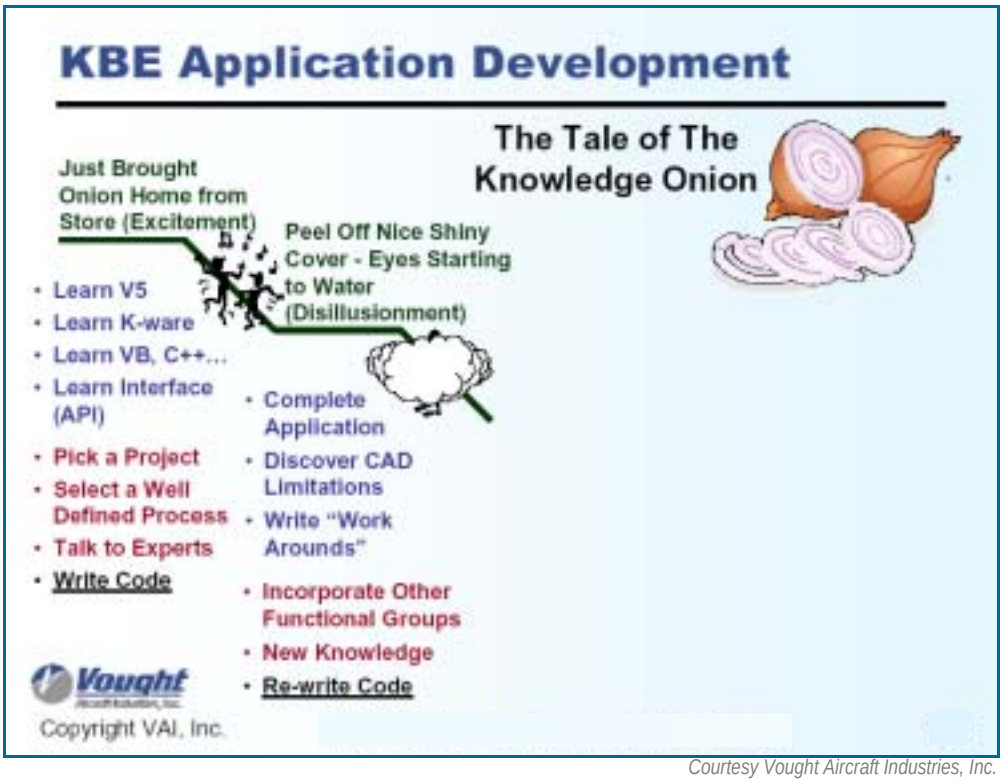


FIGURE 10
KBE Application Development

Courtesy Vought Aircraft Industries, Inc.

It has been a wild ride back from the store, but now we are home with our new onion and it is time to use it. We started peeling off the shiny outer skin of the onion, and our eyes started to water. What is happening? We had actually completed an application, and we discovered the limitations of the CAD system. The API is not as open as we would like, and we had to write some workarounds because we could not access certain CAD operations or the accessible operations did not perform as advertised. New knowledge starts to crop up. We thought we gathered everything the first time out with the experts, but we found something was missing. Now we had to rewrite code, and a feeling of disillusionment fell over us like a thick fog.

What kind of new knowledge comes up? When we created the original bond tool shown in the upper right-hand corner of Figure 11, the experts were finally able

to visualize the parts in 3D and we were ready to send them to downstream functional groups. When we sent the header out, the tool fabrication group wanted to know where the tooling holes were, which confused us since they were not in the design standard. It turns out they always put in the holes in order to line up all the headers, but never got around to requesting an update to the standard. We were able to add those without a problem.

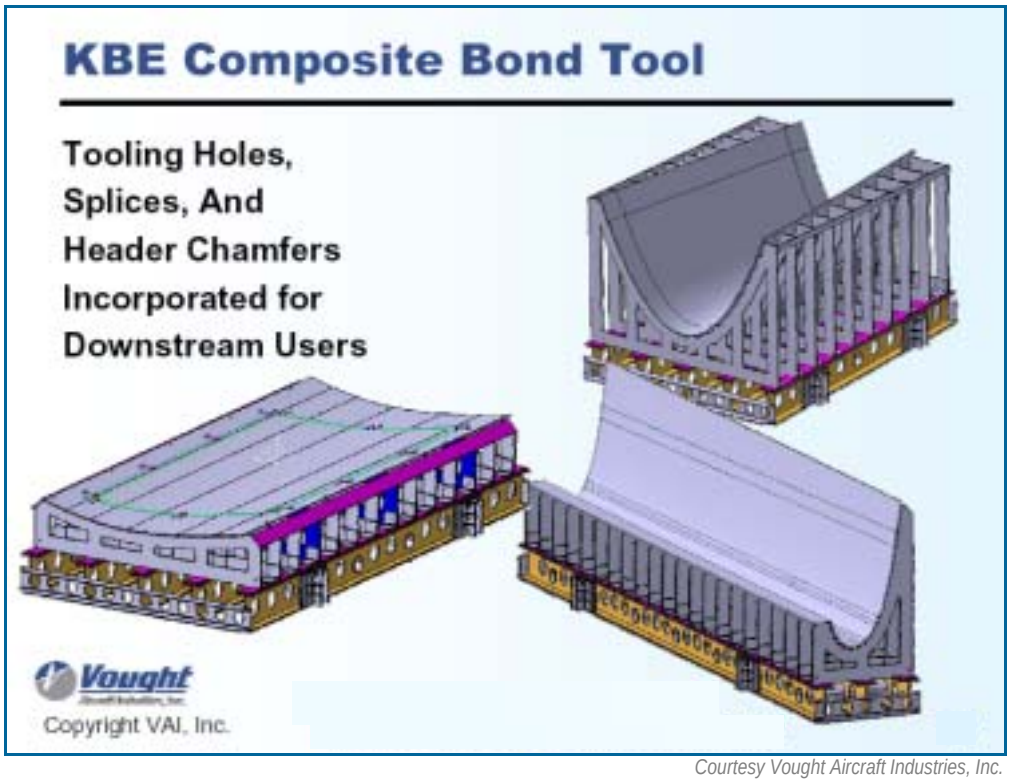


FIGURE 11
KBE Composite Bond Tool

Courtesy Vought Aircraft Industries, Inc.

Then we gave the header to NC Programming for preparation to machine, and they were happy because we put the header in the format they had requested. But right away they went into the CAD and they added a chamfer on each of the headers. Again, the chamfer was not in the spec, but it was something they always added internally. Apparently, they had not told us about it because we had not asked. We put the chamfer on, which saved them time. Then, during the design reviews for two different configurations, we were asked to add a horizontal member in one and splices in the other. To us these tools looked no different than any previous tool, however they had exceeded some unwritten limit in height or length that required horizontal members or splices. Because there was no written definition of where to put the horizontal members or splices, we had to confer with the experts. Pretty soon we were rewriting code, lots of code.

Back to our onion. We have the cover off and we are peeling the layers. By now, we are squinting our eyes and wiping away the tears. But we now encounter another attack on our senses that was not expected, and we are totally taken aback by a strong odor. Where is that smell coming from? Up to now, we have been able to rewrite the application without too much trouble. But even a balloon has

limits to its flexibility, and we have reached that limit in our code. Adding splices is going to require a totally different approach in the code and an entire subroutine will need to be revised. The experts have now had a chance to fully exploit the application and they want more. They offer more "hidden" knowledge and ask that it gets incorporated into the application. Where were they when you asked the first time? "We have not made tall tools in years," they lament, as we try to work horizontal members into the headers. The new knowledge drives another major re-write of the code and reality begins to set in.

Missing knowledge during the acquisition process can certainly hurt, but uncovering undocumented knowledge, what we like to call "tribal knowledge," and quickly using it can be equally painful. Figure 12 shows our bond tool built to the design standard on the left, and the way the shop fabricated the base on the right.

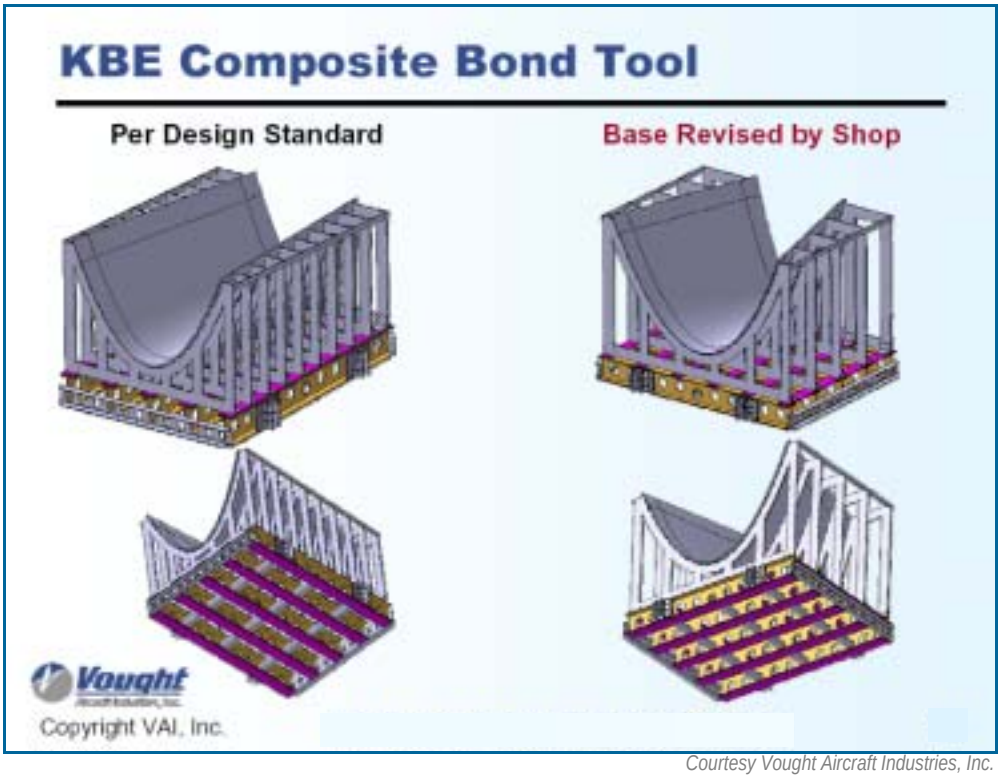


FIGURE 12
KBE Composite Bond Tool
Standard vs. Revised

Courtesy Vought Aircraft Industries, Inc.

Remember that originally they never got the base as a 3D model or a 2D drawing. They were allowed to do whatever they wanted within the constraints of the shop standard. Typically the length of the tool runs parallel with the centerline that aligns headers in a row. This is because the headers act as support ribs and their number is determined by the length of the part. But when the part requires few headers because its width exceeds its length, then the tool is shallow, and the shop would turn the I-beams ninety degrees from what was stipulated in the shop standard, as shown in the right-hand-side view. The I-beams are steel extrusions that in cross section form an "I." The shop did this on their own initiative because it helped them to install the L-brackets. From their perspective it had

little impact on the tool since the headers supported the bond surface of the tool. They had been doing this for five or six years, and because there were so few tools requiring this adaptation, they never fed it back to Tool Design to get the shop standard revised; basically no one outside of Tool Fabrication was aware of it.

When we learned about it, we were pretty excited, thinking we had uncovered some new tribal knowledge. We incorporated it into our application and then presented it to upper management to show them how the application created one base, but then based on shop best practices, it automatically converted to another.

We thought it was wonderful. But it turns out this was not the best design practice. We typically do not have stress engineers analyze our bond tools; it would not be cost effective. But if their analysis could be captured in the application, then we could offer an added bonus to the application and learn how to incorporate stress analysis into KBE at the same time. The stress engineer on our team looked at the bases and started developing the code to analyze the bond tool inputs (size and weight of part), and then output bond tool face sheet thickness, header count, and base direction. Through his analysis, he found out that when you rotate that base, the torsional capabilities fall off dramatically. The shop was accidentally creating unstable tools that were prone to movement without special handling. As it turns out, the users of the tools (the Bond Shop) had developed their own workarounds for these tools. We had to pull all that code back out once we added the stress routine. The experts did not agree, and we had to rewrite the code.

We are at the end of our story (see Figure 13), and we have been pretty intimate with our onion for quite some time. We are starting to gain a new appreciation for our onion, or at the very least a new perspective. It is certainly pungent, affecting more senses than we expected. It is very strong and we have learned that we need to know how to use that strength to our advantage. But this is our first onion, and as much as we would have liked to avoid the adverse affects, the story is pretty much over (except for the crying and a total collapse in discipline). We did not waste any time peeling our onion and now we get to rewrite more code. However, next time our application will be more modular, and we will test our tribal knowledge to make sure all experts agree, and we will keep a better eye out for service packs in the CAD that impact our code, and we will ultimately rewrite less code.

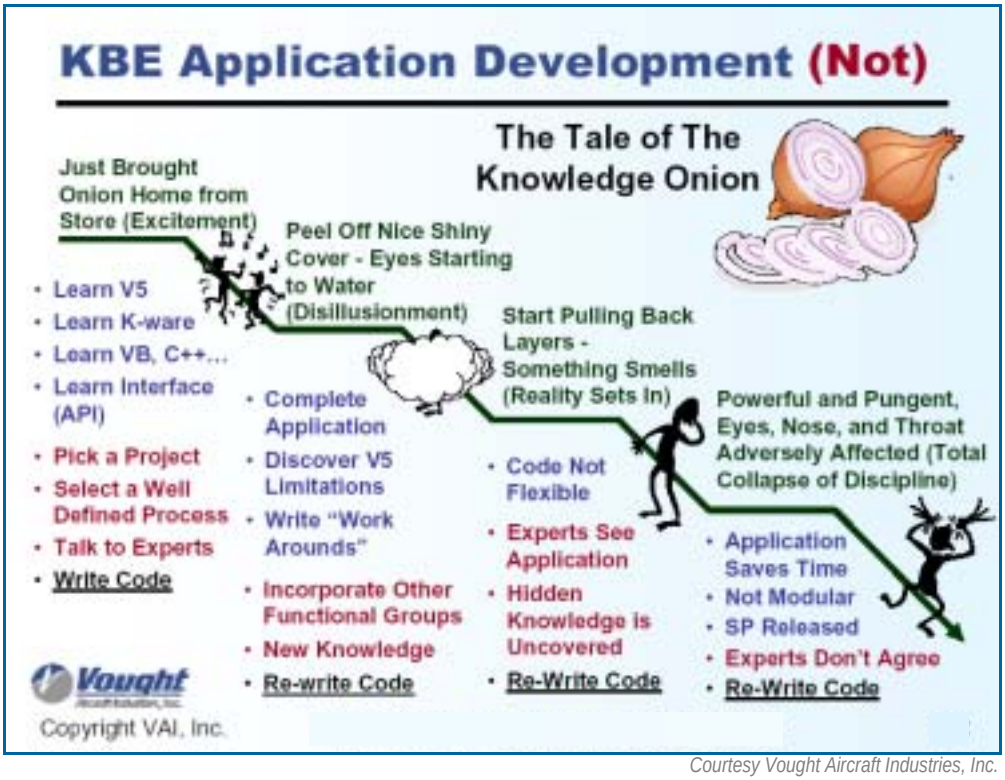


FIGURE 13
KBE Application Development (Not)

Courtesy Vought Aircraft Industries, Inc.

LESSONS LEARNED – ONION WISDOM

I hope that my story will help you remember the lessons learned, what we like to call *onion wisdom*. For emphasis I have summarized some of the more important lessons as follows:

- Incorporating knowledge typically falls somewhere between challenging and frustrating. It is not easy and it requires someone to learn a unique skill set, a "knowledge engineer" skill set.
- You would think that the deeper you dig for knowledge the better. However, oddly enough, sometimes you can dig too deep. There are political land mines and/or antiquated processes that are sometimes best left alone. Your mission is to automate the well known. Redesigning the product or its process is best left to the subject matter expert. Should you uncover an "ugly" under a rock, assess its impact to your ROI and to the company, then either let upper management buy off on the risk, cancel the project, or put the rock back, quickly.
- You need to ask the right questions, as I hope I have illustrated. Easier said than done, but over time you learn to create a "logic tree" in your mind while interviewing the subject matter experts. You follow that logic and see holes in the tapestry or missing steps, and you fill them with the right questions.
- You need to continually ask. Most of us do not realize the number of decisions we make daily as we go about our business. These are the decisions you may want to capture. Often the experts will leave something out simply

because to them it is unimportant or they do not understand your needs. So you must maintain an ongoing communication with your experts during the development process so that you can continually capture hidden knowledge.

- Not every input (be it information or knowledge) is necessarily a good input. Everyone's process is "tweaked" continually by the users. Chasing "best practices" can become as elusive as chasing butterflies. So when in doubt, test your inputs by having them examined by all of the subject matter experts, from the authors to the users.
- Knowledge engineers must understand the entire process, and they need to identify all the requirements. Any missing knowledge will obviously impact the outcome of your application. We are investigating the use of design structure matrix (DSM) and axiomatic design to help us manage all of the requirements we need to codify into our applications.
- You need to remove barriers to innovation. That can be a real issue with the subject matter experts, who are as wary of each other as they are of you. But you will find yourself in a unique situation as you try to automate a process. Procedures will be questioned and plenty of waste will become apparent. People will want to cling to their regular routine and protect their turf. But the computer does not care about any of these issues; it just wants the facts so it can generate an outcome. Your job is to get the experts to agree and work collectively toward the best process, hopefully all within the constraints of your existing procedures.
- There are elements of knowledge management (KM) in KBE. This is an issue because the KM people do not know what KBE is doing, and the KBE people do not follow what knowledge management is doing. Both disciplines are practicing similar skills, but there are very few who see the advantage of maintaining a strong relationship between the two. Knowledge management is the conscious strategy of getting the right knowledge, and the right people, at the right time, and helping people share, and put information or knowledge into action, in ways that will improve organizational performance. This is huge when applied to KBE, and you need to keep it in mind. Like it or not, your application contains knowledge that will need to be managed. Managing knowledge deals with truth maintenance (testing), knowledge half-life (new knowledge is continually created replacing the knowledge in your application), and additional concepts that will improve your application like knowledge acquisition, knowledge mapping, complexity theory, semantics, and ontology.

QUESTIONS AND ANSWERS

Question: What was the impetus within your company to start your group, and to capture the knowledge? Did it come from engineering management, from information systems, from upper management?

Premetz: It came from upper management. After Carlyle purchased us, they wanted to examine our internal capabilities and our ability to accept new work. Upper management identified neglected areas in our manufacturing (Shop Floor), engineering (Design, Manufacturing, Planning), and technologies (Stress, Materials & Processes). They approached middle management to find out which technologies or initiatives were immediately needed that would not just match our competitors' level, but would hopefully leapfrog them by five years. That was the impetus to get us going. KBE was identified as one of those initiatives.

Question: As you mentioned, knowledge acquisition is the most critical part of knowledge-based engineering application. How do you organize your group to support this kind of application – do you have one engineer on the project start knowledge acquisition as well as coding? Or do you organize your group with some dedicated to knowledge acquisition and others to the coding?

Premetz: There are only seven people in my group, so we do the acquisition and the coding concurrently. It is important to me that they learn both skills. We find a person can do a better job of codifying the knowledge if they know how to acquire it, and have an easier time codifying if they are the ones doing the acquisition. Also, due to the way we acquire knowledge by going back to the subject-matter experts at various stages of development, it ends up being more efficient to have the same person do both tasks for their particular application. So one engineer does both.

Question: In regard to the ROI and the justification, if you look at the other lifecycle events, isn't it very difficult to quantify the overall savings? How do you handle that case?

Premetz: You are absolutely right. For instance, in the case of the base, what I did not mention were some savings that were totally unanticipated. Our original goal was to obtain a reduction in hours in tool design through the creation of the 3D digital model. But feedback about a month later from the shop floor identified twenty-five percent savings in the cost of the materials. How did we do that? Before our application, the base was made by using a shop standard. But Procurement could not predict what to order in materials until Tool Fabrication, the people who actually build the base, had sketched the base out for them. So they ordered a lot more material than what was needed rather than risk schedule by not having enough. This created either scrap or over stock. Now they can order exactly what they need, which resulted in the twenty-five percent savings in materials. We then found out from Tool Fabrication that because they did not

actually have to go through the shop standard and put the base together, we saved about sixty percent of their time. It is difficult to identify those savings up front, because you only realize them when the application actually goes out. On the other hand, having never experienced the benefits of KBE, I would have had a difficult time getting the experts to sign up for savings on such a dramatic scale. But now that we have done it once, we know better, and will have an easier time talking to those experts and identifying the savings.

