The Big Theory of IDEs

CASPAR BOEKHOUDT, INFORMATION METHODOLOGIES

Remember the halcyon days when development required only a text editor, a compiler, and some sort of debugger (in cases where the odd `printf()` or two alone didn't serve)? During the early days of computing, these were independent tools used iteratively in development's golden circle. Somewhere along the way we realized that a closer integration of these tools could expedite the development process. Thus was born the integrated development environment (IDE), a framework and user environment for software development that's actually a toolkit of instruments essential to software creation. At first, IDEs simply connected the big three (editor, compiler, and debugger), but nowadays most go well beyond those minimum requirements. In fact, in recent years, we have witnessed an explosion in the constituent functionality of IDEs.

Doesn't this make you speculate on where this is all leading? I've wondered whether it's perhaps analogous to the Big Bang. That theory postulates that the universe began with a fiery explosion that hurled matter into space, resulting in the ongoing expansion of the universe we now observe. But what of its future? There are many theories: Some believe that it will continue expanding without end; others believe that the expansion will slow and eventually stop,

Pondering the vastness of the ever-expanding universe of IDEs, you might wonder, "Is a usable IDE too much to ask for?"

Bang

The Big Bang Theory of IDEs

reaching its equilibrium; yet another group believes in an oscillatory behavior in which the universe will begin collapsing again (sometimes called the Big Crunch) after reaching a point of maximum expansion. Important and profound additions to the mix are the considerations of energy, entropy, and chaos—each of which is all too apparent in the developers' world of today.

Since the introduction of what was arguably the first IDE—one for Dartmouth Basic back in 1964—we've seen many IDEs come and go. By almost any measure, IDEs have advanced tremendously over the years, but it is apparent that this is not yet a universe in equilibrium, and many, if not all, such developer environments have serious shortcomings. Today's state of the art in IDEs reveals a constant and continual expansion in which IDEs are getting larger and larger seemingly by the minute. But IDE vendors still have a long way to go to meet a single commonsense goal: to make the lives of developers easier, enabling development teams to be more productive and resulting in higher-quality software products.

THE DEVELOPMENT PROBLEM TODAY

Ten years back, life was a good bit simpler. Most of us were usually writing code in just one particular language to run in one particular operating environment, and that was pretty much your lot. Perhaps there was a slight initial conundrum when trying to master all the APIs of the particular system in question, and maybe you had to come to grips with a few other vendors' layered products that you also needed to be part of your overall solution cocktail, but that was the worst of it.

If you're like me, nowadays you're developing for systems with many more moving parts. With today's multi-tier Web-deployed applications, we have a number of new headaches. First of all, developers probably need to know a minimum of four basic programming languages: C++, Java/J2EE, VB, and C#/.NET. Developers also need a fair understanding of several key operating environments: Unix (Solaris, HP-UX, and AIX), Linux, MS Windows,

and Macintosh (as well as perhaps Pocket PC and Palm). Next come the various Web-deployed data description representations such as HTML, WML, XML, XSLT, XSD, DTD, CSS—and then there's the data access methodologies and backend integration stuff such as SQL, JDBC,

ADO, and so on. There're also the various scripting languages and techniques that you'll probably need to use in the front tiers: JavaScript, CGI, JSPs/struts and servlets (for J2*E) or ASP/ASPXs (for Windows), Perl, Python, and PHP. Finally, let's not forget all those componentization technologies such as EJBs and COM/COM+. And we haven't even started talking about what's going on with Web services.

The worst part of this problem is that when you've finally got your head around all the various parts involved in today's denouement, you must ensure that all these different parties dance nicely together when they meet up at the resulting debutante's ball. Regrettably, it's pretty darn hard to see that happen, unless you first can conduct a series of trial dances to see them all in action before their big public performance.

So exactly what do we have to work with? Let's take a look at the state of today's development universe.

TODAY'S TOOLS: "BE MORE THAN YOU CAN BE, AS AN IDE"

Have you noticed that yesterday's rather useful stand-alone tool probably has been wrapped and packaged as today's IDE? Dazzled by the glittering prize of tool integration, many tool vendors appear to have confused "integration" with "housing it all under one roof." In a race to provide a tool or component to address every imaginable developer need within one all-encompassing IDE, vendors fail to focus on improving the functionality of each particular tool available in their IDEs.

More important is the fact that true integration requires considering the synergy of particularly important collections of tools (say a Java class file navigator, source browser/editor, and debugger for a Java development platform). Instead, it seems that IDE vendors are providing features that, as often as not, hurry first to "cover the earth" by bundling many constituent components (whether other integrated tools, wizards, or add-ons), offering little value to the software development process.

GODZILLA COMES TO DINNER: SYSTEM REQUIREMENTS

IDE vendors mean to make our developer's life less painful by bundling all such features with their products. In reality, IDEs have the ability to make your life a living hell (which I've found is often the case). The mere size of these IDEs alone seems ridiculous. Table 1 reviews recommended system requirements (on the Windows platform) for a few popular IDEs.

Practical experience has taught me that, if they are to be even remotely useful, each IDE requires a lot more resources than advertised. So, while the fantasy of these light "dining requirements" may continue to be flown in the products' marketing materials, developers doing any meaningful development at the minimum levels will quickly find themselves in a slow-motion world, perhaps adding new meaning to the concept of the "halting problem" (see <http://www.cgl.uwaterloo.ca/~csk/halt/>).

"FAT AND STUPID IS NO WAY TO GO THROUGH LIFE, SON" (WHAT WE NEED AND WHAT WE DON'T)
Pick any of today's major IDEs and you'll find that most of them supply a barrage of tools: XML editor, parser, and validator; XSLT, DTD, HTML, CSS, and text editors; FTP, SFTP, Telnet, and SSH; source code and version-control software; database clients; and so on. Some IDEs even bundle in entire application servers and databases. Although none of these individual features is bad, an unfortunate outcome is the rugby team "pile on." Vendors bury you with tools you never asked for. Unfortunately, core tools can't be extracted, resulting in a significant waste of space and other resources.

Further, many constituent tools—even those meant to address an important function—are often just toys. Though many of the tools supplied by a single vendor and integrated into its framework may be useful for the

most basic tasks, often they're not sufficiently powerful to handle your full needs. You end up with at least a half a dozen different development tools and/or environments open at one time, each acting as a separate IDE, complete with integrated tools and associated baggage.

By an order of magnitude, working with a combination of IDEs (including the barrage of mostly useless features contained in each one) is less useful than using more primitive tools independently. These feature-heavy bundles don't just consume precious system resources, their performance burden, combined with the distraction they impart to you, wastes your time—and time is one of the most precious commodities in today's high-pressure rapid turn-around software development life.

For example, although Visual Studio .NET, the NetBeans IDE, Eclipse, and Sun ONE Studio all have integrated XML editors and HTML editors, you will rapidly discover a situation that has you scrambling to download and install xmlspy or Dreamweaver (or both). Now three XML editors are open, along with whatever other integrated tools each IDE has brought along for the ride, but are simply not needed for your current task. These tool integration environments, meant to be the savior of complex development projects, can, ironically, slow development by gobbling up your system resources and forcing it to thrash between the constituent parts of one or more such development tools.

All this at a time when most programming practitioners need to rely ever-more fundamentally on these tools to help manage the scale of the ever-more complex software systems within which they are developing component designs. A tool vendor would do better to focus on the main purpose of its tool: to help the developer construct elegant components of proper granularity and integrate them into the overall software system being developed. Instead, we observe features that at best only add

bloat, and at worst actually obstruct development. Can there be any way out?

TABLE 1

Resource Comparison

	Sun ONE Studio 5*	NetBeans 3.5 IDE	Visual Studio .NET Enterprise Architect	Macromedia ColdFusion MX
CPU	Pentium III 1 GHz	Pentium III 700 MHz	Pentium III 600 MHz	Pentium- N/A
RAM	768 MB	384 MB	160 MB	512 MB
Disk Space	700 MB	125 MB	900 MB on system drive, 4.1-GB installation drive	400 MB

*Standard Edition.

Data compiled from product information pages of company Web sites.

MODULES (LESS IS MORE, MORE IS LESS)

Modularization offers hope for overcoming monolithic monstrosity in today's developer frameworks. Support for adding and removing modules within an IDE or developer framework gives developers the

The Big Bang Theory of IDEs

ability to install only the tools they need without wasting resources on modules that aren't immediately relevant. They download only those modules they will need and load only necessary tools into memory. This saves more than disk space and memory—it also means no wasted time initializing the module at startup, and no wasted time figuring out what this tool is and what it does. In addition, if the current tool just isn't cutting it, you can easily remove it and (after storing a copy) replace it with a better and more suitable module.

So if modules are so great, why are IDEs so large, and why aren't the included modules more useful?

Most IDEs support plug-ins or add-on modules, but this capability comes with its own set of issues. IDEs can include too many modules by default, and many of the

modules that are included with IDEs are simply not powerful enough or are hobbled in some way that makes them useless for all but the simplest purposes. Sun ONE Studio and the NetBeans IDE, for example, each includes a CVS (Concurrent Versions System) client module, but this

client does not support SSH (Secure Shell) and supports only pserver connections. Integrated XML editors are another example. Sun ONE Studio, the NetBeans IDE, Eclipse, Visual Studio .NET, Macromedia Dreamweaver, and many others bundle one of them. Most include an XML parser to validate whether or not your XML is well formed, but might be missing XSLT tools, an XML schema or DTD generator/editor, or XPath checking (or all three)—important functionality needed to work effectively with XML from a developer's point of view.

Module capability adds the burden of choosing which module(s) to use. Beyond worrying if your development system has enough memory to load the giant IDE, you must determine the best available modules. Are your teammates using different modules for the same task,

A Day in the Life of a Developer with IDEs

How many of you have the privilege of specializing in only one development task, or even one development language? I find that I must have about 10 development-tool applications open at a time: one for XML, one for HTML and JavaScript, one for C# or Java, one for SQL, and so on. It is not unusual during a Web application development project for me to have the following applications open: SecureCRT, TOAD, Sun ONE Suite or Visual Studio .NET, FTP, MS Word, Outlook, TextPad, xmlspy, Dreamweaver, Photoshop, Visio, and more (as shown in figure 1).

Managing all these applications at once is a major hurdle. I expanded my taskbar so that it can display two rows of applications, but, at the rate this is going, I will soon have half of my display covered by the taskbar, with little or no

room left for the running applications themselves. This has, however, made me a master of Alt-Tab (under Windows). If you haven't yet discovered this, it's the art of holding the Alt key while operating the Tab key, like those kids with Game Boys (in the fashion of a semi-automatic weapon), to cycle through the different applications you have open.

THE PROLIFERATION OF IDEs

Not only is each given IDE expanding in its own right, I live in a universe populated by ever-more independent IDEs, all of which I must master if I'm to cover a suitable range of languages and system environments required to complete my development projects. For every technology I learn, it seems there's a corresponding new IDE for me to master. Not only do I have to learn and master the new technology (and the new IDE it comes wrapped up in), I also have to master the gratuitously different user interface to 30 other

FIG 1

which might create incompatible code (or, God forbid, might be better than the ones you're using)?

Another big problem is that most modules have to be crafted specifically to a particular IDE. If you were to develop a new tool, you would find you have to develop a new module for every major IDE. Tool vendors look at the current state and decide: "Hey, what a hassle. Why write a module that can be used with only one specific IDE? I might as well write a separate tool. While I'm at it, I might as well throw all these additional simple tools into it. Dude, I've got my own IDE!" And so we find ourselves in a horrible spiral of ever-more IDEs, with each one getting ever larger.

WIZARDS (OR "THE TROUBLE WITH TRIBBLES")

Nowadays, what IDE can consider itself respectable if it doesn't also write your code for you automatically? Welcome to wizards. OK, so maybe wizards can be excellent if used properly. They can improve productivity dramatically by generating the necessary base code used as a starting point for completing complex components. On the other hand, wizards can be extremely hazardous to a software engineer's health. In addition to consuming space and adding to load time, poorly designed wizards can sometimes be feeble, generating simple stubs with no real functionality behind them. This is all but useless. In

other cases wizards can generate far too much machinery, which frankly can be far worse than the other thing.

Assuming that the wizard generates a reasonable amount of code, the quality of that code can vary widely (and wildly). Sometimes wizard-generated code can be so complicated that average developers cannot understand it and are therefore justifiably hesitant to modify it. In many instances wizards obscure some of the outputted code's functionality, which you really need to understand (wizards are perhaps appropriately named). This can easily insulate you from understanding critical aspects of the system at the level on which the generated code is operating. Many less-experienced developers come in knowing how to drive these tools but don't understand the surrounding mechanisms. This can lead quickly to using the tool in lieu of proper thinking.

Visual Studio .NET's Simple Object Access Protocol (SOAP) Web Services/Clients wizard, for example, is easy to use and is great at generating simple yet complete Web services and clients. In addition to generating the code behind, Visual Studio .NET generates a Web service proxy to go with each SOAP client. As long as you don't need to modify the generated code, it works well. If the SOAP client needs to be enhanced to do something more elaborate, however, uttering a prayer to the IDE gods is in order. For one thing, like most wizards, the generated

tools that this new IDE also chose to integrate; 25 of these tools I ultimately discover to be perfectly useless to their supposed purpose and must therefore replace with another vendor's tool. You've heard of exponential growth? I soon find myself running 12 different IDEs with a situation that looks like Deathmatch 2003. Does this situation sound familiar to you? All I can say is that Code Warrior is aptly named.

Although development goes much faster now that I've learned how to use these IDEs, picking a tool is still a matter of preference (and budget). One thing remains constant: No matter what IDE you buy, even if it says it provides tools for all your needs, once you exceed the most basic tasks, you'll certainly end up using "just one more" IDE.

THE INTEGRATION IRONY

I keep on observing that vendor-specific tools or components are integrated to form their own IDE monoliths. What I want is the ability to integrate functional components (tools) into different vendors' IDEs and to get easier integration of the code produced by different IDEs themselves (and the operating environments they typically represent). Although

some framework-specific standards do allow tool integration, I don't see the standards that would allow tool components to integrate into more than one framework.

In principle, the NetBeans framework intends to provide a basis for different developers and/or vendors to build components that integrate together in that environment. As an open source environment, it intends to be a vendor-neutral basis for integrating developer tool components. Since NetBeans has increasingly become adopted for Java, however, it appears to have become more narrowly focused on solving just the Java/J2EE development problem. The planets of .NET have little to do with this galaxy.

The audio industry seems to have cracked this: My Sharp TV seems to work well with my Sony PlayStation, my Bose stereo, my Philips DVD player, and my old Atari 2600. Is this too much to ask of my IDEs? Apparently so: Don't ask VS .NET to use the Dreamweaver HTML editor and the xmlspy XML editor instead of the one it bundles. IDE makers must learn that what developers want is only the most powerful function-specific tools for each task wrapped in their IDEs. I want them to stop adding features and start defining framework interfaces.

The Big Bang Theory of IDEs

generated code? What if you just can't understand all that goo? I don't suppose that's ever happened to you! How many junior folks do you know who assume that that wizard-generated code is bulletproof? Even experienced developers can spend hours or days looking for the problem

code includes only the briefest of comments describing the purpose of the function. Missing is text describing the logic behind its functionality (why it selected certain classes, why it does what it does, and how it does it).

Wizards encourage lazy developers (and some of the best of us are lazy) to rely too heavily on the generated code. If it appears to work, why should we bother to understand what it does? Remember my Visual Studio .NET SOAP client proxy example? By default the wizard creates one proxy class for each SOAP client. You might argue that as long as it works, why muck with it? On the other hand, wouldn't it make sense to have one Web proxy that can be used by all the common SOAP clients? With only minor changes, projects could be engineered to have one proxy class to handle a collection of clients, rather than one proxy for each SOAP client. There are obvious benefits when debugging and modifying a single proxy class as opposed to one per service. Too bad the wizards and the IDE can't handle that.

Most wizards also lack real options. They may offer you trivial things like the name of the class you want it to include and the package name to go look in, but they are not very configurable. In fact, why Visual Studio does not give you options regarding the proxy class is beyond me. Why not add a simple option allowing you to choose whether you want it to create a new proxy specific to your client, generate a generic proxy that can be used by a group of similar clients, or refer to an existing proxy? That would be useful.

Of course, wizards have also been known to generate code that doesn't exactly comply with industry, project, or even an individual developer's own coding standards. This might be considered tolerable if the generated code works and never needs to be touched. However, there are many instances where matters (dare I mention performance?) can be improved significantly simply by tweaking the generated code slightly.

You encounter yet another important problem when debugging your code: What if the problem lies in the

in their own code, never suspecting the generated code as the culprit. If Microsoft or Sun wizards produced the code, it can't possibly be wrong. Think again.

Wizards aren't always the best answer, and in many cases they're more hassle than they're worth: They may, to paraphrase Edsger W. Dijkstra, belong more to the problem set than the solution set. ("How do we tell truths that hurt?" In *Selected Writings of Computing: A Personal Perspective*, Springer-Verlag: NY, 1982.)

WHERE ARE WE HEADED?

Although it's impossible to say for sure what the future will bring, we can speculate on some possible courses of evolution. If you find the Big Bang metaphor an apt one, there are three potential outcomes: IDEs might continue to expand indefinitely; they might ultimately stop expanding; or they might reach a point where they must become smaller again.

ETERNAL EXPANSION

What if the IDE universe continues its expansionary trend, getting ever larger by the accretion of more integrated tools than our present minds can imagine?

In my view, not even Moore's law holds any hope for this path. Visual Studio .Net 2003 Architect comes on five CDs, making me wonder just how many CDs I will have to shuffle into my computer to install the 2006 version. Perhaps it will soon come on its own hard disk, or given the trend toward Web-downloaded software, maybe I should plan a quarterly hiatus of seven or eight days to install it that way. Once I've got the software loaded for all six or seven of the IDEs that I need, I simply plug in four new 64-processor blades and half-a-suitcase of memory, and it's off to the races.

BIG BAMBOO—THE PLATEAU

A second possibility is that at some point in the future, IDEs will come to include a finite collection of integrated tools sufficient for every task. With a utopian view of this

future, this path would lead us to a single “truly Schwarzenegian” development environment. The implications of this outcome are similar in many respects to the eternal expansion model. For one thing, such an environment must include tools to support every possible development task (and most important, support them well). In addition, these tools will have to be so powerful that you do not need to use any other tools.

When an IDE includes all possible tools, and no conceivable tasks might require a new integrated one, it remains the same size. Such an outcome doesn’t seem very likely. Can you imagine a time when we’ll have tools that can handle every possible task that might come up—with tools able to handle every possible future technology, pattern, and task? Human nature alone seems to ensure that we will never achieve it. Even were we to decide on a primary development religion (e.g., Java vs. .NET), would we really be able to agree on using just one IDE?

The irony of this path is that, to the extent that marketing and product differentiation continue to be based on a quantitative features checklist (principally of embedded tools vs. their individual quality, their overall integration, or any measure of real resultant developer productivity), the future of a number of expanding IDE galaxies within an ever-expanding universe of them, along with the woeful productivity drains this implies, is all but inescapable.

One possible conduit for a size plateau, however, is perfect modularity. If IDEs can achieve this, we might be able to have development environments that remain the same size, include all the tools a developer needs, and have the capability for dynamically swapping out modules for a better or newer one able to handle the newer technologies, patterns, and tasks. In other words, IDEs get

Developer Productivity vs. Sustainability?

Wizards, IntelliSense, and debugging capabilities (such as setting breakpoints and walking through code) can improve productivity radically. But I see how IDEs have changed how my codevelopers look at code. It’s easier to debug code into existence than to think and design it from scratch (e.g., desk-checking code for efficiency, accuracy, and other problems). This can cause me a lot of heartache later when bugs are discovered after the system’s deployment. Tracing or logging statements are rarely inserted in the code these days because the IDE is so all-seeing at development time.

Management (and the market) demands that you and I complete projects ever faster, but also expects us not to rely carelessly or sloppily on the features that allow for faster coding. I often find myself asking management, “Do you want it done fast, or right?” It’s true that I can code a program faster functionally using these special IDE features (like breakpoints and code-stepping as I build it up) instead of creating a debugging or profiling framework in the implementation itself. But later, when the system is in production, we have a hell of a time trying to figure out why it’s not working as intended. I find that we are plagued with a dilemma of where the balance lies between coding fast and coding well.

larger than they currently are, but at some point in the future will stop expanding because of an equilibrium in which old modules are discarded and new modules are added.

To be fair, within both the .NET and Java camps some of this is starting to happen. If tool vendors could even begin to agree on documented interfaces between different tool components and modules, there might be some small hope for cross-IDE modules. Which brings us to ...

THE CRUNCH

It’s at least possible that at some point in the future our software development universe and/or its respective IDEs will begin shrinking in size. For this model to work, either IDEs have to become well modularized, or a more graceful relationship must develop

between different tool integration frameworks, or both. Namely, we might imagine a universe offering independent integratable tools that are reusable within multiple IDE platforms/frameworks. This requires highly efficient modular components that are portable to many different developer frameworks and platforms.

Today, many modules work with only a specific IDE. If they can be built so that modules can be used within competing IDEs, software Darwinism should result in modules sufficiently powerful that one does not need to have a completely separate IDE surrounding each tool. Wouldn’t it be great to load `xmlspy` as a module and use it within Visual Studio .NET, Dreamweaver, or Sun ONE Studio? Imagine a computer with only one XML editor installed—as opposed to the 10 to 20 currently installed on the average developer’s machine. If this were possible, individual tools (or small suites of them) could become more specialized, focusing on their own primary goals.

Imagine a world where you do not have to have 10 applications open, of which seven contain tools that do

The Big Bang Theory of IDEs

development tremendously, but these sharp tools lack safety guards. In uninformed hands, these tools can be dangerous; today's best tools just accelerate our demise when used by bad coders. Good software development, however, still relies on developers who understand

exactly the same thing. We can only dream and hope.

Today's IDEs are designed by competing vendors who are not playing nice. The likelihood of Microsoft developing and including an interface to which a user can add a module developed by Sun (or vice versa) is slim to none at present (although both would argue they already support standards to do that). Instead, IDE vendors seem driven by an apparent competition to add new features—not because developers asked for them, but because they perform better in marketing specsmanship.

CONCLUSION

Although IDEs aren't going away, their future remains unclear. Like enterprise resource planning (ERP) software and other applications whose size and complexity expanded to narrow feature gaps with their competitors, IDEs must find a way to manage complexity and improve developer productivity.

We find ourselves situated within the midst of the IDE Big Bang. IDEs are larger with each new release, and their expansion seems to be accelerating. We software engineers and developers, whose job it is to understand this IDE universe, have been left grasping for knowledge and understanding, as our development environments expand—seemingly exponentially—in all directions.

When used well, development tools can speed up

the business problem as well as the code, and who can apply team and industry best practices, leverage standards, and have an overall knowledge of the technology in question. You use and do only what you know.

Today's tools are like Formula 1 racecars. They contain a lot of experimental parts, and while behind the wheel your life truly is in your hands, especially as you're being asked to drive them pedal to the metal. My advice to racing teams? Make sure you've got trained drivers. ☺

LOVE IT, HATE IT? LET US KNOW:

feedback@acmqueue.com or www.acmqueue.com/forums

CASPAR BOEKHOUDT, a principal with Information Methodologies (imi), a leading enterprise Web integrator for higher education, is a software architect and developer. He is a Sun-certified programmer for the Java 2 platform and IBM-certified developer for XML and related technologies V1. He has been involved in architecture and planning, and has developed various enterprise applications using a combination of XML, XSLT, XPath, HTML, Java, C#, Perl, C++, JavaScript, VB, and SQL—and in so doing has used almost every major IDE currently on the market. He can be reached at boekhoudt@infometh.com.

© 2003 ACM 1542-7730/03/1000 \$5.00.

NetBeans IDE 3.5.1 (for the Java platform): <http://www.netbeans.org>.

Sun ONE Studio 5 (for the Java platform): <http://www.sun.com/software/sundev/ide/>

Eclipse (for the Java platform): <http://www.eclipse.org/>.

Visual Studio .NET 2003 (for the .NET platform, supports C#, VB .NET, J#, and many others): <http://msdn.microsoft.com/vstudio/productinfo/features/default.aspx>.

Macromedia ColdFusion MX 6.1 (for ColdFusion): <http://www.macromedia.com/software/coldfusion>.

Macromedia Dreamweaver MX 2004 (for HTML/DHTML): <http://www.macromedia.com/software/dreamweaver>.

TextPad 4.7.0 (general-purpose editor): <http://www.textpad.com>.

TOAD (SQL editor): <http://www.quest.com/toad/>.

WinCvs (version control): <http://cvsgui.sourceforge.net/>.

RESOURCES