

Solving Object Modeling's Shortfalls

How the Object Modeling Discipline Must Change To Meet the
New Mandate of Enterprise Software Development

Contents

Executive summary	3
The early promise of object modeling	3
Traditional object modeling is no longer sufficient	4
A silo approach falls short	4
Business models are changing more rapidly	5
Object models are losing their relevance	5
The ever-decreasing half-life of technology standards	6
Rework: The inevitable result of technology change	6
IT employee turnover rates jeopardize deadlines	7
The problem of productivity	7
A new model-driven approach for rapid change	8
The transformation pattern and its many benefits	8
What to look for in an MDA modeling tool	10
Borland’s Model Driven Architecture advantage	11

Executive summary

Object modeling methodologies have long held the promise of enabling development teams to deliver higher-quality software faster with better opportunities for code reuse. Sadly, these promises have gone largely unfulfilled. Architects armed with even the best-quality visual modeling tools still find it difficult to deliver high-quality software on time and under budget. This trend can be expected to continue until companies adopt an approach to object modeling that can overcome the challenges and complexities.

Enterprise-level applications have never been easy to develop. A myriad of obstacles stands between the architects and the software they have been charged with delivering. These obstacles come with an added stressor that has never been as great as it is now: the sheer speed at which business changes bombard the IT organization. The breakneck pace of business change forces architects into a reactive rather than a proactive mode.

Now, in the post-dot-com world, business stakeholders hand enterprise architects even more pressure on top of the already burgeoning list: *Do more with less—and do it faster.*

To overcome these challenges, a new approach to object modeling is required—one that delivers frequent, tangible working results in environments where changes come faster than ever before. Such an approach must leverage all available design and development resources, regardless of how scarce they might be.

Agile methodologies, even when executed properly, solve only part of the problem. While they enable teams to react rapidly to business change, agile techniques don't address the issue of individual productivity limitations and shifting technology standards. Tighter project deadlines require the enterprise to produce more technological assets faster, with fewer defects—all while employing fewer human resources.

The challenge is not insurmountable. Despite the oppressive nature of these obstacles and market forces, some enterprise architects and consultants routinely deliver frequent, tangible, working software under budget and in a timely fashion, with fewer skilled developers.

This white paper explores the challenges faced and how some IT organizations overcome them to produce repeatable success in delivering enterprise-level software using a flexible, standards-based approach that aligns people, process and technology.

The early promise of object modeling

Developing software is a real challenge. But as we move forward into the 21st century, custom software will continue to be an absolute necessity – something nearly every global enterprise relies upon. While smaller companies and individual departments can quickly develop applications without a significant pre-development effort, larger enterprises cannot.

Development of a mission-critical application for a large multinational company demands significant up-front planning—the same way a 40-story office tower does. Such an effort requires skilled manpower, effective tools and, most important, sound architectural design. This business necessity gave birth to a discipline designed to handle the complex task: object-oriented analysis and design (OOA&D), often referred to as “object modeling” for the sake of brevity.

As early as the mid-1990s, object modeling became a staple practice at larger corporations. Organizational charts sprouted new branches containing positions such as chief architect, infrastructure engineer and feature designer. Software vendors built and sold visual modeling tools for these highly skilled software professionals. Not only did companies invest heavily in these tools, but they also invested in expensive training courses to teach their architects to design and build software faster and cheaper.

Just as a building architect draws blueprints from which the construction workers can build, a software architect draws analysis and design documents for the development teams to program.¹ Using the Unified Modeling Language™ (UML®), software architects can model nearly any real-world operation their software intends to automate.

From the start, the architects' UML diagrams intended to make the complicated task of enterprise software development every bit as easy and repeatable as building a house. These guiding design artifacts promised to enable teams to produce more source code faster and with fewer defects than had they used no up-front design at all.

For several years, traditional object modeling appeared to deliver on these promises. UML eliminated a great deal of confusion and excess meetings, allowing development teams to coordinate their work more effectively. However, as we move forward in the 21st century, object modeling success stories are becoming the exception rather than the rule, even for smaller, less complicated development projects.

It is becoming painfully clear that the conventional practice of object modeling is no longer equipped to meet the goals that the corporate world now demands.

Traditional object modeling is no longer sufficient

Why is conventional object-oriented analysis and design no longer adequate? How did the effectiveness of the object modeling profession gradually erode to the point where it no longer works for even department-level solutions? What changed?

The short answer: everything.

Enterprise-level software is funded by business, which demands a return on investment, even in the face of drastic changes to core business models. Technology is changing at an absolute breakneck pace, even faster than it did in the late 1990s during the Y2K scare. Finally, software is written by people, who are finding the post-dot-com IT labor market to be exciting once again—with bigger paychecks around almost every corner.

A silo approach falls short

Modern enterprise software delivery must account for different stakeholders, each with distinct and very valid business needs.

- **Business stakeholders** create demand for new applications regarding requirements and business process models.
- **Software development** accepts business requirements and transforms them into technical specifications—and eventually working software.
- **IT operations** deploys the finished application and maintains it in the fast-paced environment of business and technology changes.

We cannot detach the object modeling discipline from the rest of the Application Lifecycle Management (ALM) functions (see Figure 1 on next page). Separating object modeling from the ALM environment in which it operates would render any solution unworkable in the real world.

¹ Gary Cernosek, Eric Naiburg, *The Value of Modeling*, IBM Software Group, 2004

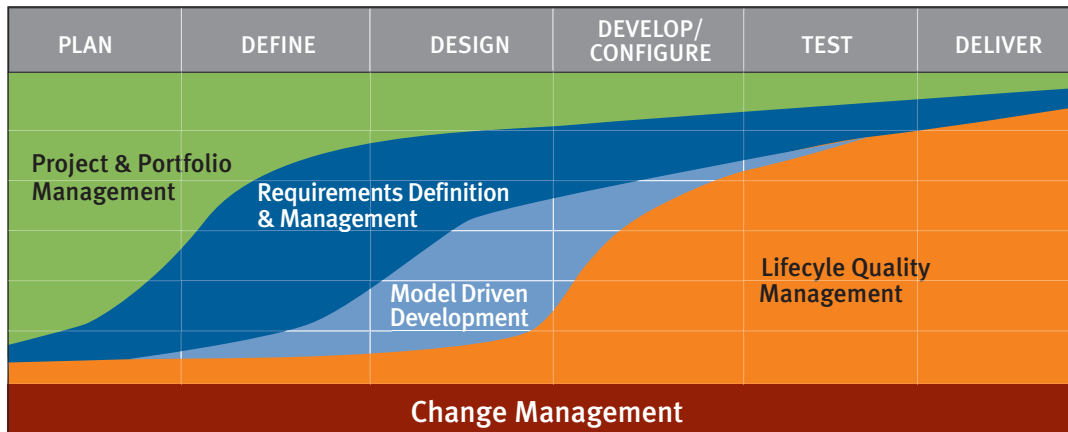


Figure 1: Object modeling must work within the entire enterprise ALM process.

The solution to the shortcomings of traditional object modeling will be valid only if it bridges the historical chasms between Requirements Definition and Management, Development and Deployment, and Testing and Delivery. Failure to effectively integrate with each of these crucial ALM functions will result in a solution that works in a vacuum only—which is not a solution at all.

Business models are changing more rapidly

Mergers, acquisitions, joint ventures, divestments, business model shifts, right-sizing, you name it. It's changing. The hoped-for result is for a company's mission-critical applications to adapt to these business changes as quickly as the changes themselves.

However, depending on the flexibility of the software's underlying architecture (or lack thereof), the costly mission-critical application may not be able to adapt to these changes without a significant amount of rework. If such changes were unforeseen by the original architects, the retrofitting endeavor may take longer to develop than the original application itself.²

In these cases, refactoring becomes kludgy. Hurried developers must force-fit new operations with wrapper code around wrapper code. And the once bullet-proof application turns into a brittle, unstable mess that folds like a house of cards at the first runtime error. It isn't long before scrapping the original application and starting from scratch becomes the only course of action that makes sense.

It would be easy to lay the blame directly at the feet of software architects and object modelers, but that wouldn't be entirely equitable. Considering the way that object modelers are routinely trained, such unfortunate outcomes shouldn't come as a complete surprise.

Object models are losing their relevance

Once software requirements are agreed upon and the baseline is set, architects do as they're trained. They hold multiple Joint Application Development (JAD) sessions, discuss high-level design and analysis concerns, and decide on a development process and deadlines for completion. Then they fire up their favorite visual modeling tool and draw multiple UML diagrams.

They may draw Use Case, Sequence or Class diagrams. They may even make up a few diagrams of their own to enhance the drawing effort. They then click the print button, and pin the new diagrams up on a cubicle wall in the development office.

But as soon as the last pin is pushed and the first line of code is written, the design starts to lose its value. The connection between design and development fades with each passing coding day. The architect's ingenious design is supposed to guide the developers' every keystroke. But by the middle of the development cycle, it becomes little more than a suggestion.³

² Peter Kutschera, Steffan Shafer, *Applying Agile Methods in Rapidly Changing Environments*, <http://www.jeckstein.com/papers>, 2002.

³ Anneke Kleppe, Jos Warner, Wim Bast, *MDA Explained*, 2003, Addison-Wesley, p.2.

The team may even hold code reviews in an attempt to ensure that developers are adhering to the architectural guidelines. But these code reviews quickly become the first casualty of the development effort as deadlines draw near.

Then when the business model changes (and change, it will), one of two very bad things invariably happens:

- The design must be amended to account for the new business model, even as developers are still coding to the original. This causes substantial rework of plumbing code by hand, taking large chunks of precious development time the team can ill-afford to give up.
- In the interest of time to market, the design documents are left as is (i.e., ignored), forcing the developers to code the changes as they see fit, regardless of the architect's guiding UML directives. It is at this point that the role of UML is demoted from being the source of critical direction to a no-longer-relevant suggestion.

Sadly, disasters like this happen every day, sending major repercussions throughout the enterprise. Customer needs go unmet, causing lost business—and even lost careers.

Business models will change. Architects must design software that supports any business model change with working software that incorporates the change. However, the conventional modeling techniques commonly used in the enterprise today simply aren't responsive enough to meet this demand.

The ever-decreasing half-life of technology standards

The effective lifespan of a technology standard is becoming shorter and shorter as we move forward in the 21st century. Soon after a major technology vendor comes out with a standard, real-world developers prove it to be lacking, and therefore unsuitable.

ASP gave way to ASP.NET. The once-ubiquitous J2EE™ Struts framework has been replaced by the newer Spring and Velocity frameworks. And Sun's tour de force, Enterprise JavaBeans™ (EJB™), has been kicked to the curb by nearly all J2EE shops, now that newer, more lightweight data access alternatives are available.

Like building an office tower, building software requires the architect to design a technical infrastructure—a foundation that underlies all of the features the business requires. This foundation supplies all of the basic “plumbing” services every feature will need: data access, security, concurrency, logging, et al.

Changing this infrastructure once the application has already been built and deployed is akin to changing and repouring a skyscraper's foundation after the tenants have moved in. It's very difficult, super-expensive, and will cause a lot of service disruption.

Yet major corporations—even the ones that hire the brightest design architects—travel this path often, regarding it as a normal course of software development. The conventional object modeling techniques they employ nearly guarantee they will continue to find themselves in this unfortunate situation.

Rework: The inevitable result of technology change

Using traditional object modeling techniques, an enterprise architect usually begins by drawing UML from a pure business model, without regard to the underlying technology. However, as the design process moves forward, the architect is usually forced to blend the two before turning over the design to the development teams.⁴

The reason: Architects are paid to design applications that yield the optimal performance after deployment. This demand for the fastest runtime speed usually means architects must intermingle their business and technology models, thereby making the two inseparable. The result is code that runs very fast, but can't be upgraded to a newer architecture without risking the entire application's viability.

⁴ Frank Baerveldt, *Agile MDA: Setting Up Projects for Long-Term Success*, Compuware Corporation, 2006.

This presents a sobering problem once a new technology framework becomes accepted as a new standard development paradigm. Since the rate of technology change continues to increase, an enterprise needs to be able to adopt the new standard and incorporate it into its mission-critical applications rapidly. However, for most IT departments, any new standard that appears after the application's initial design phase will most likely have to wait until the next major release.

Could this be considered a design flaw on the part of enterprise architects? Not at all. The gambit of runtime speed versus flexibility to change has always been a very real tradeoff—and traditional object modeling simply does not have a good answer for it.

IT employee turnover rates jeopardize deadlines

If knowledge is power, then the most powerful people in the enterprise are the ones designing and developing mission-critical software applications. They're the only ones who can make the software do what it takes for the business to survive. And if those employees leave, so does all that knowledge of how their projects are to be developed and deployed. And leave, they do. A recent human resource survey shows that a full 85 percent of American IT workers expect to be employed by another company within the next 12 months. Replacing them is no simple matter. Software development professionals take an average of 37 percent longer to replace than nontechnical talent.

This was not always the case. The economy that followed the dot-com “bomb” did not treat software development professionals well. But now that the labor market for IT workers in the United States is back to its pre-9/11 fervor, IT workers are taking full advantage—by jumping ship for better pay and benefits.

Forbes magazine columnist Bernadette Kenney writes: “All too many employees have come to regard corporations as promise breakers and, as such, hypocritical and untrustworthy. In particular, employees assign these less than complimentary attributes to their managers. Workers no longer see the point of company loyalty, while many feel betrayed and angry. Indeed, many of those who feel violated by employers actually begin a job search as an act of revenge.”⁵

Even those software professionals who aren't looking for another IT job contribute to the problem by leaving the information technology field altogether. Long hours and the threat of offshore outsourcing contribute to employee burnout and insecurity, prompting even seasoned IT pros to seek a career with less stress, even if it's accompanied by less money.

The enterprise must accept the fact that its highly talented IT workforce will turn over. And it will be costly, both in real dollars as well as lost knowledge of the inner workings of the firm's mission-critical software.

The problem of productivity

Even the most sound technical architecture doesn't address the issue of individual developer productivity. Eventually, a team will have to stop drawing pictures and start writing code. Regardless of how accurately an architect's UML diagrams communicate the chosen design, it will still have to be implemented by a team of live humans. And all the UML diagrams in the world cannot help here.⁶

The most effective visual modeling tools on the market allow for simultaneous round-trip engineering. Draw UML, the tool makes code out of it. Write code, the tool makes UML out of it. The tool essentially creates a one-to-one ratio of abstraction to code. This ratio may have been sufficient in years past (although some developers from that time period may argue this). But today's “do more with less, and faster” mandate demands a higher level of productivity from individual developers.

A human programmer can write only so much code in a day's time. With each missed deadline, it becomes clearer that this ratio is no longer sufficient to meet the more aggressive deadlines of the 21st-century global enterprise. An effective UML tool is the obvious choice to compensate for this human limitation. But to do more with less, the tool has to improve upon the current one-to-one ratio of individual developer productivity.

⁵ Bernadette Kenney, “The Coming Crisis in Employee Turnover,” [<http://www.forbes.com>], 2007.
⁶ Anneke Kleppe, Jos Warmer, Wim Bast, *MDA Explained*, 2003, Addison-Wesley, p. 3.

A new model-driven approach for rapid change

Today's enterprise needs a new approach to object modeling—one that overcomes these obstacles while shortening the development cycle and minimizing project risk. In fact, such an approach exists. It is an emerging advancement in object modeling from the Object Management Group (OMG) called Model Driven Architecture® (MDA®).

Forrester defines Model Driven Architecture as “an iterative approach to software development in which models are the source of program execution, with or without code generation.”⁷

While Model Driven Development™ (MDD™) is the industry-standard term for all modeling activities, MDA is a subset of these activities; it deals with the use of models for generative approaches to software development and delivery.

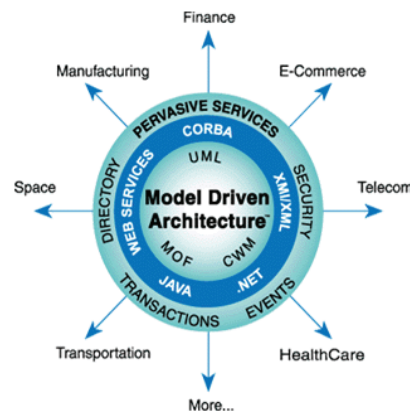


Figure 2: Model Driven Architecture

The Model Driven Architecture starts with the notion that the architect can craft a model of the core objects in the enterprise's business domain (see Figure 2). In this model, these are business objects only—with no forethought of the technology that will be used to build them. Once the architects are satisfied that their electronic model accurately reflects the real-world business model, they invoke the transformation pattern of their MDA-enabled modeling tool.

Once invoked, the tool iterates through every artifact it finds in the business model, and takes certain actions based on those objects using conditional logic encapsulated in the pattern. The output of this transformation is executable source code, ready to compile, deploy and run. With the technical architecture in place and functioning properly, the company's developers are free to program the business features that the end users require, as only skilled developers can.

The transformation pattern and its many benefits

The true intelligence of the MDA tool lies in the transformation pattern. Essentially, the pattern is an iteration-based mapping, instructing the tool how to make source code out of the business model. Most MDA tools come with at least one transformation pattern, usually allowing the architects to create their own code outputs based on the architecture they choose to employ.

Once the architect is satisfied with his transformation pattern, he can use it to generate the entire application infrastructure in a rapid, predictable, repeatable way. Even a very complex technical infrastructure can be generated in a matter of minutes—something an entire team of developers might have taken weeks or even months to program by hand.

⁷ Diego Lo Giudice, The State of Model-Driven Development, Forrester, 2007.

This model-driven paradigm shaves untold man-hours off of the average development cycle. But MDA's true effectiveness lies in its ability to adapt to business changes quickly and gracefully. If the business model changes, the architect need only draw the new model, reinvoke the transformation pattern, and hand the generated code over to the developers to do what can only be done by hand: program granular business features. The benefit to the enterprise is large-scale time savings.

MDA relieves developers from the burden of spending countless hours or days recoding mindless database access routines or underlying services. The MDA tool's transformation pattern performs the task for them. The resulting infrastructure code is ready for the feature developers to program against right away. There's no need for them to understand how it works—only *that* it works.

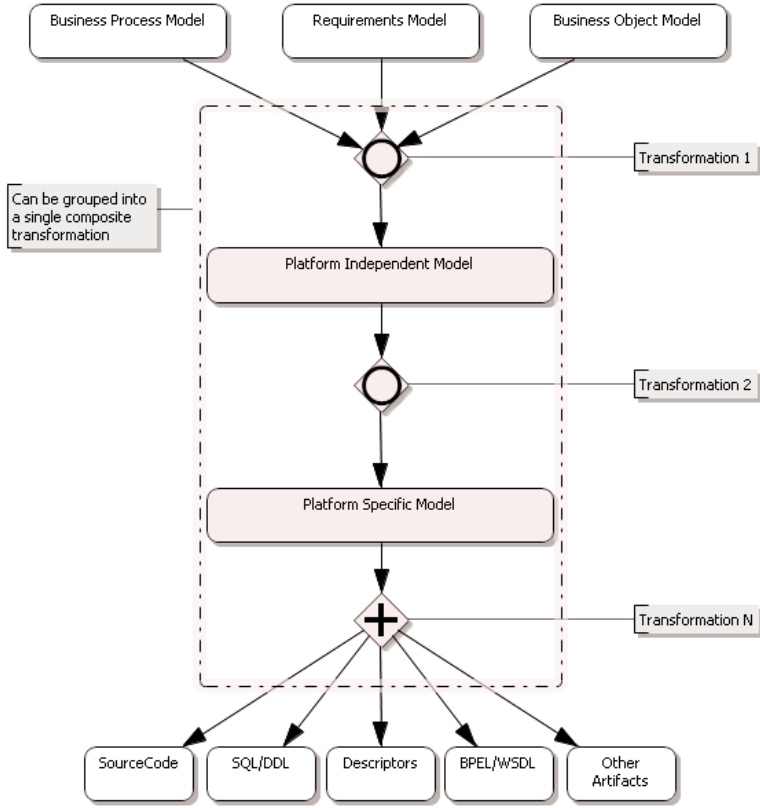


Figure 3: UML models can be transformed into any development artifact.

MDA also allows the business to react to technological change with as much agility as it responded to business change. Higher-quality MDA tools feature pluggable transformation patterns (see Figure 3). If the business started with J2EE but recent changes require the entire application to be rewritten in .NET, the change is an easy one. Just unplug the J2EE pattern, replace it with a .NET pattern, and reinvoke the transformation from the same business model.

Because the technology lives in a transformation pattern and not in an architect's brain, the enterprise is insulated from unforeseen personnel changes. If an architect leaves the firm, reacting to business change becomes as simple as amending an existing business model diagram and invoking a transformation—a task that can be done by anyone capable of drawing in the UML.

What to look for in an MDA modeling tool

When considering a tool to implement a Model Driven Architecture, it is important to keep in mind the following characteristics:

- **Pluggable transformations.** An MDA-enabled tool must have transformation patterns that can easily be interchanged. Some vendors market their products with only one transformation that cannot be removed without paying a substantial fee. Such a tool cannot deliver the flexibility necessary in the face of technological change. Choose a tool that allows one transformation to be removed, and another inserted in its place.
- **Transformations suitable for existing projects.** Legacy systems remain very important if they are able to continue to run and meet business needs. A good MDA tool should allow architects to integrate new features with an existing application without requiring a complete rewrite of the legacy infrastructure. New features should be able to coexist side by side with legacy code without causing integration errors. Some tools are suitable only for new development. Make sure to avoid these.
- **Transformations built on QVT standard.** QVT (Query/View/Transformation) is a standard for open source transformations using Model Driven Architecture. On the other hand, a tool that uses proprietary pattern language means that your transformations cannot be shared among tools of different vendors. Quality MDA tools should allow an architect to take a transformation created in one vendor's tool, give it to an architect running another vendor's MDA tool, and expect it to work exactly the same way. A tool that utilizes transformations according to the QVT standard allows architects this freedom and flexibility.
- **Adherence to Open ALM.** No development tool should create too much coupling between an enterprise and its tool vendors. Additionally, the choice of development tool should not force an enterprise to change its chosen development methodology or techniques. Make sure your MDA tool doesn't imply a process unsuitable for your enterprise.
- **Built-in audits and metrics.** Choose an MDA tool that comes out-of-the-box with OCL-based audits and metrics based on modeling best practices. Running audits and metrics provides architects a measure of the soundness of their model before transforming it into source code. Such a tool should alert the architects to potential design flaws they may not have been aware of at the time they constructed the models.
- **Document generation capability.** Individual developers will be programming features against the generated infrastructure code. And while those developers don't need to know how the code was generated, they certainly need to know the API of the generated code so they can produce their features on top of it. A good MDA tool has a flexible documentation generator that communicates the generated API to the developers in a way that yields maximum productivity per developer hour spent.

Borland's Model Driven Architecture advantage

The Borland® Together® solution has been the tool of choice for enterprise architects and designers for nearly a decade, and is the only tool in the object modeling space to offer all of the features discussed in this paper. Together was the first OOA&D tool that featured simultaneous round-trip engineering with LiveSource®, keeping models and code in perfect synchronization at all times. Now as we push forward in the 21st century, Borland Together can help enterprises do more with less—faster than ever before—by offering architects a myriad of benefits that shorten their development cycle while minimizing project risk.

MDA consultant Robert Lario of Inherit Software uses Together exclusively to help his clients to do more with less, and to do so faster:

“With Borland Together, I’m able to build models in a way that I can generate code from them. Models are more of a first-class citizen instead of just pretty pictures. I can utilize the MDA features of Together to make use of the models themselves to produce executable artifacts that are reusable.”⁸

Lario particularly likes the way Together makes creating transformation patterns easy and intuitive:

“I can run my transformation rules directly from my business model. If the result of the transformation isn’t quite what I want, all I have to do is tweak the transformation rules until they produce precisely what I need. So in the future, the enterprise benefits from reuse and repeatability. Borland Together makes the whole process very simple.”

Every enterprise has its own distinct personality when it comes to writing software. In particular, Lario recalls one of his larger telecom clients that had pieced together some patterns and templates that it routinely uses in its portfolio of applications.

Lario and his consultants showed the client how to use Together to model those business patterns and procedures. Then he wrote a QVT to build the code that would give the telecom what it needed to finish the application quickly and in a repeatable way. The result:

“The client told us they would have taken some 3 months to build this part of the framework alone; it was really difficult to communicate to developers because it was so error-prone. With Together, the process went from 16 weeks down to about 8 weeks.

“But it was more than just the delivery cycle. It was the quality. There was no more running 6 months in production, then finding errors in the code. The consistency and quality of what Together delivered was significantly improved over anything the client could have done by hand.”

Lario’s customer story shows how Borland Together overcomes the shortcomings inherent in traditional object modeling using MDA. In Lario’s opinion, any company that doesn’t take advantage of MDA with Together is really missing out on an opportunity to increase their productivity.

If your enterprise needs to do more with less, and do it faster, you can absolutely meet that mandate. The first step toward doing so is to visit www.borland.com/together. Once there, you can view an online Web demo, request an in-person demonstration, or download a free trial copy of Together. Or call 1-800-555-1212 to speak with a representative right away.

⁸ Robert Lario, Inherit Software Corporation, personal interview, May 2007.

Borland is the leading vendor of Open Application Lifecycle Management (ALM) solutions - open to customers' processes, tools and platforms – providing the flexibility to manage, measure and improve the software delivery process.