

# BriteWorks solves the software crisis

## The software crisis

The Standish Group estimates that businesses in the US spent around \$382 billion on software development in 2003 and wasted \$82 billion due to failures, cancellations, etc.

It would hardly be an understatement to say that software has not kept pace with hardware when you consider that programmer productivity has only improved 3% - 8% over the last 30 years.

Software development today is largely an art form based on craftsmanship. Everything is reinvented, resulting in duplication of effort - it's tedious, time consuming, error-prone and very expensive. And as software becomes more complex, it further aggravates the situation.

It is generally agreed that many software systems built today are similar and should be built out of reusable software components. Capers Jones' research revealed that 75% - 90% of code in business applications could be reused. However, much of the hope that was placed in Object-Oriented (OO) and component based development in breaking down the barriers to reuse has yet to be realized.

Currently, reuse and component based software development remains the most studied topic in software engineering. Virtually all initiatives in addressing the problem of software productivity are some variation of the reuse of software components. This applies to model-driven development, UML, frameworks, .Net, J2EE, Eclipse IDEs and code-generation techniques.

A common thread that runs through all the above approaches is a reliance on the traditional *process-centered paradigm* of producing code. They are all code-centric - focused on producing code, managing the code, and looking at various levels of automation. The tools are focused on essentially automating as much as possible the identification and selection of the components. The developer still has to have sufficient knowledge of the internals of these components to adapt them to his specific needs. They are of greater help to the expert developer than the novice.

The guiding philosophy behind this approach is that one selects a set of components that deliver parts of the application requirements and then puts these components together by connecting inputs to outputs. SOA is an extension of this philosophy, but certainly carries with it an assumption that the developer takes responsibility for integrating all the pieces together.

Another fundamental problem with this approach is that they try to be all things to everybody and try to solve every IT problem, ranging from device drivers, kernels, games to business applications. The problem space is just too large to be addressed efficiently and concisely.

### **Why are 3GLs inappropriate for developing business applications?**

Programming languages are the fundamental technology of programming. To make fundamental improvements in programming, you need fundamentally better programming languages. Programming is so hard that only highly talented, trained and dedicated individuals can do it well. The vast majority of programmers are those building end-user applications and the even greater population of potential programmers is end-users themselves. This kind of programming is hard not because of sophisticated algorithms and data structures, but rather because of the overwhelming multiplicity of simple constructions.

However, general purpose programming languages were developed by computer scientists to address the needs of ALL computers systems, inclusive of developing device drivers, kernels, compilers, operating systems as well as run of the mill business applications. Programs are encapsulated by structures and algorithms; well defined structures result in simpler algorithms. Much of what we know about program language design is about clever ways to encode structure in text, and ontologies to rationalize naming. Much of what makes a language unique is its particular style of name resolution, as in polymorphism and inheritance techniques of OO languages. Text is a dead end because all these sophisticated techniques in one way or another are just encoding structures in text strings.

Structures need to be made explicit and directly manipulable, not implicit and encoded. Mentally parsing and interpreting this implicit structure is an enormous cognitive burden for programmers.

We should manipulate semantic structures directly through a WYSIWYG interface. Direct manipulation is the cornerstone of the GUI interface as exemplified in Windows and the Mac. It is also enshrined in products like word-processing, CAD and perhaps best exemplified in spreadsheets where end-users are able to directly manipulate cells and see the results instantly. Research has shown again and again that end-users are instinctively drawn to direct manipulation interfaces; they have a much shorter learning curve and they are accessible even after periods of disuse.

In recent years there has been a backlash against the complexity of Java, and a movement towards simplicity with 'dynamic' languages like PHP, Python and recently Ruby. These languages still suffer from the impedance mismatch between text and structure. What they have done however, is bridged the gap between design time and run time. The instant gratification does not however make any fundamental improvements in programming.

Eclipse IDEs have also gained popularity because they go to great lengths to extract the encoded structure back out of the text and help programmers visualize and manipulate it somewhat directly. Another well intentioned, but misguided approach is visual programming with some form of graphical diagrams.

The way forward is clear – we should liberate ourselves from the tyranny of textual representation and manipulate semantic structures directly through a WYSIWYG interface.

## **Develop applications 20 times faster with BriteWorks**

BriteWorks blurs the line between languages, tools, patterns, applications and databases. From one perspective, BriteWorks is a configurable application. As such, it virtually eliminates the construction phase in software development.

It eliminates the need for you to think about architecture or deployment. The system is organized in such a way that it guides you in describing your requirements visually, without having to learn a new language. You don't need to know Java, JavaScript, XML, C# or learn a new programming language as there is no coding! It speaks the language of business so it is easy for the domain expert together with a developer to quickly define their information model, the user interface for supporting their workflow and the business rules. Every aspect and detail about the end-user application is defined declaratively through visual tools and is stored in a relational database as metadata.

This simplifies development dramatically, slashing the development time by orders of magnitude and reduces the development team size from armies of developers to teams of 3-4.

With BriteWorks applications are executable instantly, as design time and runtime are the same; there is no code generation or compiles. Users get to see working prototypes typically within a week or two, if not days. This helps considerably in clarifying requirements early and also serves as an instrument for exploration, with users taking ownership of the application as active participants.

With BriteWorks there is little difference between new development and maintenance - change the description and the behavior changes. Many changes can be made on the fly, even as the system is running. This slashes maintenance costs drastically and has a major impact on the total cost of ownership. BriteWorks applications are intrinsically adaptable, providing organizations a highly flexible and agile platform.

BriteWorks gives organizations a new found freedom and becomes an agent of change.

### **BriteWorks introduces WYSIWYG programming**

The core of the BriteWorks platform is the organization and separation of the user requirements into concrete structures that match directly with the way developers and domain experts think about applications. Concrete is better than abstract, and these structural components are a powerful aid in the analysis and design of the application. These structures are orthogonal, but at the same time related to each other. BriteWorks maintains these relationships automatically, eliminating another major coding effort.

The developer together with the domain expert describes the application incrementally in terms of 'WHAT' they want it to do without specifying 'HOW'. At any time the developer can switch from the design view to runtime and instantly execute the system to clarify or explore the requirements in greater detail.

With BriteWorks you don't have to wait for everything to work before anything works. Users get to see working prototypes very early in the development life-cycle, within a week or two if not days. In many cases prototypes are turned over into production prototypes that satisfy the core requirements. Users have early access to the system and

don't have to wait for the 'big bang.' New capabilities can be added incrementally without causing any disruptions.

With BriteWorks, no assembly is required - it comes out of the box as a deployable system. It includes many non-functional, but essential features for enterprise applications such as security, audit and localization. These features are turned on by configuration and provide a level of flexibility that exceeds capabilities found in many high-end systems.

### **BriteWorks is extensible**

A question that is often asked is 'But my requirements are different. What if I cannot do everything I need to do in BriteWorks?'

Firstly, BriteWorks is extensible. If the need ever arises, you can easily plug-in virtually any kind of custom code written in any language into BriteWorks. The API is easy to use and BriteWorks was designed from the ground-up to co-exist and plug-and-play with legacy systems.

But it is unlikely that you will have to resort to hand coding because the vast majority of programmers are those building end-user applications, and even the greater population of potential programmers is the end-users themselves. Programming end-user applications is hard not because of sophisticated algorithms and data structures, but rather because of the overwhelming multiplicity of simple constructions.

End-user applications seem unique on the surface, but are actually similar at a structural level. They have repeating patterns and are a 'variation on a common theme.' The algorithms themselves follow patterns that are well known and have been in use for decades. It is the sheer volume and the minor variations that make it seem so daunting.

The opportunity for reuse is not only very high, but highly predictive as well. The conventional approach is to apply reuse only at the implementation level, if at all. This is tactical and yields small improvements, at best. BriteWorks is conceptualized at the *domain* level of business applications incorporating reuse at the architectural and design level. This is a radical departure from ALL the different framework solutions. It is a key factor in the dramatic productivity gains you realize with BriteWorks that can never be matched by other approaches that are tactical.

### **BriteWorks takes you directly from informal requirements to production**

A common artifact in software development is the specification: an informal description of the system behavior, produced by analysts, interviewing users, approved by those users, and consumed by programmers. Programmers in the real world will tell you that specs aren't worth the paper they are written on. They are half-baked informal descriptions. They have to be transformed into a formal specification. The dirty little secret is that writing a complete formal specification can be as difficult and error-prone as implementing it in code.

Conventional development goes through several transformational phases – requirements analysis, documenting user requirements, detailed design, summon an army of developers to translate the design into code, integrate, test, and prepare for deployment.

The multiple steps and transformations distort the initial requirements making it virtually impossible to trace the users' needs to the final product.

BriteWorks trumps this at two levels. Firstly, it eliminates the convoluted process of transforming informal requirements into formal specifications. And then transforming the formal specifications into classes. Immediately the developer has to make decisions about inheritance, polymorphism, etc. at a stage where such decisions often are premature and costly to reengineer later. Secondly, BriteWorks is self-documenting and as such eliminates the need to formally specify the application. Since the complete description is kept in the relational database as metadata, the very tools that that are used to query and report on the application data can also be used to query and report on the application. This provides a level of transparency and traceability that is non-existent in other systems. And you can rest assured that it is always current.

With BriteWorks, the informal requirements with low fidelity examples are usually sufficient to start the design process. BriteWorks presents the user, not with a blank screen to conceive and then write code, but with concrete structures that are the natural building blocks for end-user applications. Design and development are synonymous in BriteWorks.

### **BriteWorks gives ISVs a new edge**

Customers demand a lot more from ISVs now. It is no longer acceptable for software customization to take months and cost hundreds of thousands of dollars. Customers want easily customizable software, with a low total cost of ownership, proven ROI, and immediate impact on productivity. ISVs are forced to heavily discount their software to stay competitive. Ultimately, ISVs spend so much time maintaining each highly customized version that they have difficulty responding to new customer requests. Ironically, the effort to satisfy customer's unique requirements by hand coding customizations typically results in large declines in customer service and responsiveness.

BriteWorks provides ISVs with a powerful, extensible software automation system to rapidly create, effortlessly customize, and cost-effectively maintain any type of custom software. By eliminating the need to write code to handle variations, BriteWorks dramatically improves margins and provides ISVs with a competitive advantage, enabling faster time to market, lower totals cost of ownership, and reduced maintenance costs.

### **BriteWorks and SaaS**

Analysts and industry pundits are predicting that SaaS will eventually become the dominant licensing model. Even large corporations have embraced SaaS and all the major vendors have SaaS offerings in the pipeline.

Current SaaS offerings are undifferentiated products that offer minimal levels of customization. Customers are primarily looking for integration with their legacy applications. As the market matures customers will demand higher levels of differentiation and may not be willing to compromise on tailoring their business process to fit the vendor's offerings. This will pose ISVs with the same problems that they have with their legacy product lines – maintaining highly customized versions, which of course runs counter to the economies of scale offered by a one-size-fits-all model, a' la Salesforce.com.

BriteWorks opens up new and attractive opportunities in the SaaS space. ISVs can respond to their customer's unique needs without being burdened with extensive code modifications and all the inherent problems. On the contrary, this will be welcomed as a lucrative professional services revenue stream and will also help foster a closer relationship with customers. In addition, ISVs could extend the responsibility of customization to their customers IT personnel. In the event of any major problems, you can always rollback to a stable version as all application versions are safely maintained as metadata in the database!

BriteWorks is the ideal platform for SaaS.