GUEST EDITORIAL

Donald G. Firesmith

Frameworks: The golden path to object Nirvana

THE BIG PROBLEM

There is something fundamentally insufficient with *all* of today's object-oriented development methods, languages, and CASE tools. Please don't misunderstand me or misquote me on this. I love object technology (OT). I have bet my future and the future of my company on objects. I firmly believe that OT, if not the silver bullet of software, is at least a very powerful armory of silver-tipped arrows capable of keeping the software werewolves at bay. I am even a methodologist who has developed one of the aforementioned methods. So what do I mean when I say that the current crop of object-oriented methods, languages, and CASE tools is insufficient? Insufficient for what, and insufficient in what way?

THE BIG SOLUTION

To truly solve the software crisis, we need to increase software productivity and quality by at least an order of magnitude (more probably, two). And that will require a consistent level of reuse of requirements, design, code, and test software of 80% or more, something that is still quite rare using OT today. Objects by themselves can't do it. Classes of objects are better, but still insufficient. Even class libraries based on inheritance will not suffice. They all involve building blocks (i.e., objects and classes) that are just too small to achieve the level of reuse needed. We need bigger building blocks. We need *frameworks.*

But what are frameworks? They are significant collections of collaborating classes that capture both the small-scale patterns and major mechanisms that, in turn, implement the common requirements and design in a specific application domain. I am not just talking about the current frameworks that support a specific development platform (e.g., MacApp) or graphical user interface (GUI) development. I am talking about integrated sets of frameworks including (1) *domain frameworks* that capture the essential classes, patterns, mechanisms, and scenarios in a specific application domain such as investment banking, factory automation, telecommunications, etc., (2) *user interface frameworks* that provide a common look and feel among products, (3) *database frameworks* that provide independence from and interoperability with object bases, relational databases, etc., and, (4) *environmental frameworks* that provide independence from and portability across multiple operating systems and hardware platforms. New applications can be small, relatively inexpensive, and rapidly developed because they will be built on and extend an existing superstructure of frameworks that provides 80 to 90% of the required capabilities. While some frameworks will have widespread use and will be made commercially available, the most important ones will provide organizations with a critical competitive advantage. These frameworks will increase quality and productivity while decreasing cost and schedule. The first organizations to successfully develop, institutionalize, and widely market such domain frameworks will be able to significantly underbid their competition, drastically decrease time to market, and thereby eventually dominate the market. Because of the increasing importance of software to many businesses, frameworks will determine the survivability of many organizations over the next decade.

SECONDARY PROBLEM 1: METHODS

Unfortunately, the major current object-oriented development methods, including my own ASTS development method (ADM), are insufficient. They were designed for developing applications (and objects and classes) largely from scratch. While some do have explicit steps requiring the developer to search reuse repositories and place completed software (and documentation) in these reuse repositories, the main emphasis is on identifying, specifying, designing, and coding new classes. This is certainly reasonable and should be expected because complete, robust repositories containing domain-specific classes (not to mention frameworks) do not exist yet. The current crop of methods is required during this current transition period so that we will eventually be able to pull ourselves up by our bootstraps. They are certainly better than the structured and information engineering methods they have largely made obsolete. They work well when reuse is below 50%, but they all break down long before reuse reaches 80%. We will need a new class of methods designed for (1) small, rapidly developed applications that sit on top of large, stable frameworks and (2) the maintenance of these software treasures, the corporate frameworks. Such methods do not yet exist, although some organizations, such as my own, are beginning to develop them. Although the current class of methods is wonderful and necessary for the current state of our industry, we will not achieve objective nirvana without replacing it with a future class of methods designed for the state of practice yet to come.

SECONDARY PROBLEM 2: LANGUAGES

With the sole exception of Eiffel (which includes clusters), no major object-oriented language supports a building block

larger than a class. Even ignoring reuse, this is inadequate for large projects that may contain hundreds, even thousands, of classes. Current languages do not support frameworks directly nor do they directly support concepts such as scenarios (or use cases) critical for the integration and testing of frameworks. Developers need the ability to specify, design, and code large architectures and behaviors using the same language for analysis and design that they use for coding and testing. Developers need support for abstraction, localization, modularity, and information hiding when their "modules" are collections of objects and classes (a.k.a., clusters, kits, subassemblies, subsystems) or frameworks. Even Eiffel's clusters are not as powerful as what we will need to maximize reuse. While Eiffel's incorporation of assertions is a major step in the right direction, we also need the ability to document higher-level abstractions and responsibilities so that we can capture business knowledge in these larger building blocks.

During the next decade, the industrialization of frameworks will be the most important advance in software technology.

SECONDARY PROBLEM 3: CASE TOOLS

If the current methods and languages are insufficient, then the current CASE tools that support them must also be insufficient. Yet CASE tools have two additional serious problems.

Project and corporate frameworks must also be accessible by the CASE tools. Because frameworks are analysis, design, code, and test entities, upper and lower CASE tools must be integrated seamlessly using the same repository for requirements, design, code, testing, and documentation. The CASE tools must do more than generate partial code from diagrams and the reverse engineering of some diagrams from code. Changes must propogate easily and automatically between text, diagrams, and code. The same browser should be usable for requirements, design, code, and test information.

CASE tools should provide dynamic views for dynamic behavior design and testing. Simulation engines based on semantic analysis of requirements and design information should animate dynamic diagrams such as interaction (i.e., collaboration) diagrams, state transition diagrams, object-oriented control (and data) flow diagrams, timing diagrams, scenario diagrams, etc. Multiple dynamic diagrams should be animated (e.g., by highlighting or coloring arcs or nodes). Upper and lower CASE tools must allow comparison of expected behavior (e.g., produced by simulation engines that understand the semantics of the design) and actual behavior (e.g., captured by performance analyzers and profilers).

CONCLUSION

Frameworks are the golden path to object Nirvana. While representing a necessary intermediate stage, today's object-oriented development methods, languages, and CASE tools will remain insufficient until they are integrated with and support frameworks. During the next decade, the industrialization of frameworks will be the most important advance in software technology, one that will determine the very survivability of many software organizations. If you do not know about frameworks, study them. If you do not use frameworks yet, begin building them now. Extend your methods to using as well as developing frameworks. Manually provide support for them (e.g., by use of libraries), and lobby language and compiler vendors for language support. Also press CASE tool vendors for adequate support. But do not wait; the window of opportunity that frameworks have opened will not remain open for long. Enter it now, before your competition closes it. Good luck. ■

Donald G. Firesmith is President of Advanced Software Technology Specialists (ASTS), an O-O training and consulting company. He is an author of books on the subjects of O-O methods and testing and the developer of both the ADM O-O system development method and the OOSDL O-O specification and design method. He can be reached at 17124 Lutz Rd., Ossian, IN 46777; 219.639.6305, fax: 219.747.9389.

JOURNAL OF
OBJECT-ORIENTED
Programming

October 1993
Vol. 6, No. 6

The JOURNAL OF OBJECT-ORIENTED PROGRAMMING (ISSN #0896-8438) is published nine times a year, monthly except for Mar/Apr, Jul/Aug, and Nov/Dec. Published by SIGS Publications Inc., 588 Broadway, Suite 604, New York, New York 10012, 212.274.0640. Please direct advertising inquiries to this address. Second class postage paid at New York, New York, and additional mailing offices. POSTMASTER: Send address changes to JOOP, P.O. Box 3000, Dept. OOP, Denville, NJ 07834. Inquiries and new subscription orders should also be sent to that address. Annual subscription rates for the U.S. are $153 for institutions, $59 for individuals. All foreign orders must be prepaid in U.S. funds drawn on a U.S bank with additional postage of $40 per year for air service. Single copies are $9 and orders should be sent to SIGS Publications at the above New York address. For service on current subscriptions, call 800.783.4903.

Manuscripts under review should be typed double spaced (in triplicate) and accompanied by an electronic file in TEXT format. Editorial correspondence and Product News information should be sent to the Editor, Dr. Richard S. Wiener, 135 Rugely Court, Colorado Springs, CO 80906, 719.579.9616 (voice & fax).