

Unifying Enterprise Development Teams with the UML

Grady Booch

Rational Software White paper



There is a fundamental paradox at play in contemporary software development. On the one hand, organizations are faced with the demand for faster time to market; on the other hand, these same organizations are under pressure to deliver systems with higher quality at lower cost. Keeping a balance between these two forces is quite hard: rush a software system to market, and its quality will undoubtedly suffer; focus only on quality, and you may still fail because it took you too long to release your system to its users.

Compounding this paradox is the fact that the very nature of software development has changed. Historically, many information systems were architecturally quite simple: applications would be built upon a middle layer that typically encapsulated business rules and data access, and that middle layer in turn would be built upon a persistent store, usually manifest as a relational database. That relational database (or databases) would essentially be the center of that system, capturing the vocabulary of the problem space and serving as the repository for the system's state. The advent of client/server architectures helped to codify this three-tier separation of concerns, and made it possible for organizations to respond to changes in a controlled fashion. In particular, applications could be rapidly created and modified while preserving the state of the system, new business rules could be introduced without destabilizing the system, and data could be, over time, mined in new and unexpected ways. This proven and stable architecture led many organizations to structure their teams in the corresponding fashion: analysts would work with end domain experts to translate user needs into requirements, data modelers would build domain models that satisfied the functional requirements of those users, and application developers would rapidly construct and deconstruct then construct anew systems that satisfied the system's behavioral requirements.

However, with the presence of the Web, the world of software development has irreversibly changed. In traditional client/server systems, a system would typically have a controlled number of users, often numbering in the hundreds or thousands; on the Web, a system might have millions of users, many of whom are not under the control of the software development organization. In traditional client/server systems, the conceptual distance from the application to the data was quite small; on the Web, most interesting systems consist of thousands of moving parts, some scripted and some compiled, using mechanisms that are quite distant from classic relational stores. In traditional client/server systems, change was inevitable, but could reasonably be managed; on the Web, change is continuous, and happens at every level of a system's architecture and implementation technology. In traditional client/server systems, the number of stakeholders invested in the successful development and deployment of that system were relatively small; on the Web, there are many new stakeholders, from content creators to information architects to network designers, all of whom must work together with the traditional software development team to overcome the e-software paradox.

Organizations that are successful in confronting this software development paradox operate in materially different ways than those that do not. Specifically, hyper-productive organizations treat development as a team sport, where the many different stakeholders who contribute to the development and deployment of a system are unified by a common process, a common language of expression, and tools that support and encourage these best practices associated with this process and language.

The Rational Unified Process™ (RUP)¹ is one such process that has proven useful to many organizations for confronting the software development paradox. The RUP is a process that encourages the incremental and iterative delivery of a system's executable releases. The RUP is risk- and use case-driven, meaning that it focuses upon the early identification and confrontation of the risks to a system's success, and its iterations are directed by use cases from the perspective of the system's different stakeholders. Additionally, the RUP is an architecture-first process, wherein the system's architecture is stabilized early so as to establish and validate strategic design decisions and then is successively refined at each new iteration.

Traditionally, many data-heavy systems were relatively homo-geneous in their implementation, with Cobol appearing as the dominant language. Again, the presence of the Web has changed everything, even for

¹ Kruchten, P., 1999, The Rational Unified Process: An Introduction, Addison-Wesley Publishing Company.

legacy systems that have been migrated to the Web. On the Web, a data-heavy system might use Cobol, C++, or Java on the server, with sprinklings of scripting languages (such as Perl, VBScript, JavaScript), 4GLs (such as Delphi), and classic languages (such as Visual Basic and Java) on the client. Languages such as XML play a role in this space as well: XML in particular has emerged as a common language for expressing structured data on the Web. In addition to being faced with this cacophony of programming languages, the enterprise development team must also cope with a variety of technologies such as Microsoft's WinDNA and Sun's EJB, all of which present different programming models to the developer.

For the successful organization, enabling the members of the enterprise development team to communicate with a common voice is essential: different stakeholders will have different views upon the system's design and implementation, and unless they all speak with a common vocabulary and language of expression, it will be impossible to unify the activities of that team.

This is the role of the Unified Modeling Language (UML)², a standard of the Object Management Group. The UML is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. In effect, the UML is a standard language for writing software blueprints.

For a construction project, you can't visualize, specify, construct, and document a high rise with just a single page of a blueprint. So it is with software: you need several different views of a system's architecture, each from the perspective of different stakeholders on the team, in order to capture all of the strategic design decisions that make up that system. As figure 1 illustrates, there are five views that are particularly important to describing a software-intensive system.

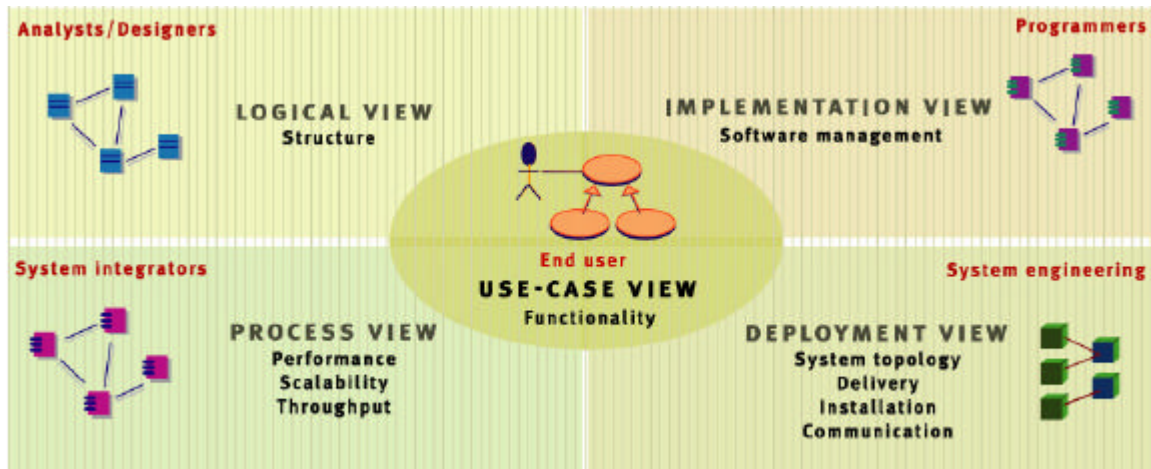


Figure 1: Use-Case View

The *use-case view* of a system is of particular interest to end users, for this view captures the desired functionality of the system. This view is of importance to testers as well, for these use cases form the basis of regression testing for each executable release.

The *logical view* of a system is of greatest interest to analysts and designers, and serves to describe the vocabulary of the problem space, together with the architecturally significant mechanisms that realize the use cases from the first view. Within this view, you'll find application, data and business models that describe the problem space, together with classes, packages, subsystems, and collaborations that realize the system's use cases.

² Booch, G., Rumbaugh, J., and Jacobson, I. 1999, The Unified Modeling Language User Guide, Addison-Wesley Publishing Company.

The *process view* of a system describes the system's decomposition into processes and tasks and the communication and synchronization among these concurrent elements. This view is of greatest importance to system integrators who must address the performance, scalability, and throughput of the system. The implementation view of a system captures the artifacts as seen by a system's programmers, and serves to model the executable components and corresponding source files and content that form those executable parts. This view is at the center of a project's configuration management practice, for it is these components that are assembled into executable releases at each iteration.

The *deployment view* of a system is of interest to a project's system and networking engineers who must craft the system's hardware topology and architect the system for delivery and installation. This view describes the physical network configuration.

All of these views may be expressed using the UML. For example, class diagrams may be used to show the static parts of the logical view, and component diagrams may be applied to the component view. The dynamic elements of each of these views may be captured using any of the UML's behavioral diagrams, such as interaction diagrams and statechart diagrams. Furthermore, with the UML's extensibility mechanisms, it's possible to tune the language to speak to the needs of a particular domain. For example, the Web Application Extensions by Jim Conallen focus the UML to the domain of Web-centric systems³. By working with this common language of blueprints, different stakeholders can contribute to their specific area of expertise, while at the same time communicate with other stakeholders.

The value of having the development team speak with one voice is especially evident in data-heavy systems, wherein database designers must work with analysts and application developers to craft a system. Traditionally, the data-centric parts of a system were modeled using entity-relationship (ER) techniques. ER approaches have served the development community quite well, but again, the world of development has changed sufficiently such that ER approaches make it difficult for database designers to communicate with other stakeholders and to express the semantics of contemporary data-heavy systems.⁴ As Dorsey and Hudicka have observed, "there is a compelling need to replace the current industry standard of ER modeling with the much more flexible, robust, and object-oriented UML."⁵ Indeed, this is the very purpose of the data profile extension to the UML developed by Rational® Software.⁶

The UML is semantically more expressive than traditional ER techniques. Not only can you model the same elements as you can with ER approaches, you can also model other kinds of relationships (such as associations) as well as behavioral characteristics (which ultimately might be manifest as triggers or stored procedures). Although the UML notation is subtly different than traditional ER notation, the shift to the UML is not difficult for seasoned ER modelers,⁷ as figure 2 shows.

³ Conallen, J. 1999. Building Web Applications with UML, Addison-Wesley Publishing Company.

⁴ Date, C. J. and Darwen, H. 1998. Foundations for Object/Relational Databases, Addison-Wesley Publishing Company.

⁵ Dorsey, P. and Hudicka, J. 1999. Oracle 8 Design Using UML Object Modeling, Addison-Wesley Publishing Company.

⁶ Booch, G., Rumbaugh, J., and Jacobson, I. 1999, The Unified Modeling Language User Guide, Addison-Wesley Publishing Company.

⁷ Dorsey, p. 71.

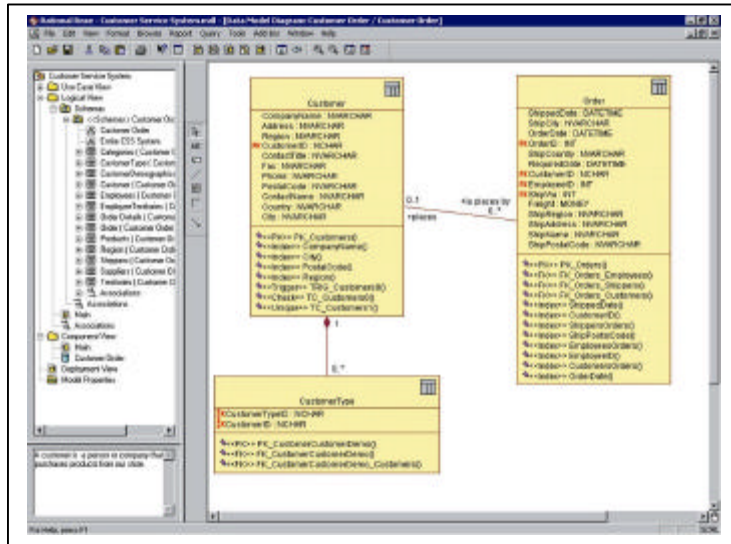


Figure 2: Switch from ER Notation to UML Notation

To specify a data model, you simply apply a UML class diagram. To further capture the logical schema of a database, you use a UML class diagram with classes stereotyped as tables. For each table, you can model its columns (as attributes, including properties such as keys and indexes) and triggers (as operations). To capture the physical elements of a database, you use a UML component diagram with components stereotyped as databases. In either the logical or physical view, you of course have the full expressive power of the UML for modeling relationships (such as associations and inheritance) and behavior (such as via interaction diagrams or statechart diagrams).

In this manner, you can fold your system’s data models and requirements into the complete project, unifying cross-functional team members into a collaborative force. By supporting these models with a tool such as the Rational Rose® Data Modeler, previously isolated members of the data team now have easy access to the data requirements in context of the whole project requirements, and can trace their data models through the application models and system use-case models to the associated requirement text and attributes. Similarly, analysts and application developers can better communicate with the data team, because they now have a common language of expression. Because the UML is so semantically deep, it’s possible to use it to visualize and specify the seams in a system, such as those found at the boundary of object models (typically seen by application developers) and relational models (as managed by the data team). This makes it possible to track the migration of an object model to a relational database model. In the presence of tools that support database round trip engineering, it’s then possible for users to create a data model based on the database structures through forward engineering or to create a database based on the data model through reverse engineering. All the semantics relevant to the data team – tables, columns, constraints, indexes, triggers, and more – can be preserved through such transformations.

It is difficult to build an enterprise software system in a manner that reconciles the forces of the e-software paradox, balancing the demand for rapid development against the need for high quality. Using the UML to visualize, specify, construct, and document the artifacts of that system enable the stakeholders of the development organization to work as one team with one language and one tool.

About Grady Booch

Grady Booch is regarded as one of the leading software development methodologists in the world. Grady is now the Chief Technology Officer of Catapult, a company founded by Rational Software and one of its strategic partners. Previously, he was Chief Scientist at Rational Software Corporation. Along with Rational colleagues Ivar Jacobson and Jim Rumbaugh, Grady developed the Unified Modeling Language (UML).



Corporate Headquarters
18880 Homestead Rd.
Cupertino, CA 95014
Toll-free: 800.728.1212
Tel: 408.863.9900
Fax: 408.863.4120
Email: info@rational.com
Web site: www.rational.com

For International Offices: www.rational.com/worldwide

Rational, the Rational logo, Rational the e-development company are trademarks of Rational Software Corporation. References to other companies and their products use trademarks owned by the respective companies and are for reference purposes only.

? Copyright 2000 by Rational Software Corporation.

TP- 188 6/00 Subject to change without notice. All rights reserved