Migrating a J2EE project from IBM Rational Rose to IBM Rational XDE Developer v2003

by Steven Franklin

Editor's Note: Each month, we will feature one or two articles from Rational Developer Network, just to give you a sense of the content you can find there. If you have a current IBM Rational Support contract, you should join Rational Developer Network now!

If you've been using IBM® Rational Rose® in a J2EETM (JavaTM 2 Platform, Enterprise Edition) project and have been wondering whether and how to switch to IBM® Rational® XDETM Developer (known until v2003 as simply Rational XDE), this five-part series of articles should help you decide. I'll use the same approach as in an earlier article series I wrote, and use a fictional project (the ASDI project) as the basis for showing how to make the transition from Rational Rose to Rational XDE Developer. I'll assume you have some previous knowledge of Rational XDE, and I'll update you on some of the new features that have been introduced in Rational XDE Developer v2003 -- Java Platform Edition. Although familiarity with my earlier articles isn't required, you may find them helpful (for example, for background on Rational XDE); see "Earlier Articles" for more information.

Of all the feedback I received on the ASDI project articles and my later articles on Rational XDE, the most popular questions were "Should I switch from Rational Rose to Rational XDE?" and "How do I transition my team from Rational Rose to Rational XDE?" This series will address those questions.

Concern over moving to a new tool

Earlier Articles

I've written a number of articles about Rational software tools, with a focus on Rational Rose and Rational XDE (now Rational XDE Developer). The goal in each case has been to show techniques for effective use of these tools, pointing out their strengths and weaknesses for particular applications.
The most popular series to date has been the ten-part "Applying Rational Tools to a Simple J2EE-Based Project," which shows how a fictional J2EE project could start at a basic level of process and evolve into a more efficient and mature team with the help of the IBM® Rational Unified Process® (RUP®) and other Rational software tools. Although the sample project (for a company called Audiophile Speaker Design, Inc., or ASDI) is fictional, it's based on my personal experiences and observations. My aim in writing these articles was to:

- show how the RUP and other Rational software tools could be used by a talented team on a fast-changing project
- share some of the lessons learned by myself and others in integrating the tools into actual projects

Since writing that series, I've written two more, both on the subject of Rational XDE ("First Encounters with Rational XDE Professional" and "Data Modeling in Rational XDE Release 2"), in which I discuss some of its interesting features and its strengths and weaknesses as an integrated environment.

is understandable. Companies and projects invest time and money in their tools, so there's a natural resistance to changing products if the current tool is working properly. A team also gathers intellectual property in the form of a tailored process, customized scripts, and training, some or all of which can be lost in the move to a new technology. I particularly understand concerns regarding moving from Rational Rose to Rational XDE Developer. Rational Rose -- the flagship modeling tool of Rational Software Corporation (now part of IBM) -- is an analysis and design tool that supports round-trip engineering (RTE) and other advanced features. If Rational XDE Developer is also an analysis and design and RTE tool, how does it differ from Rational Rose? I'll discuss the differences between the two and offer some suggestions on their optimal application.

This set of articles will continue with the project where it stands at the end of the earlier ASDI article series: the team has released their product and is now maintaining the software. They're facing a potentially major revision of the project, and they also want to sell their product to other customers. The tools and process they used during implementation were acceptable, but now they feel there are some improvements that can be made. They turn to Rational XDE Developer as a way to improve the integration in their development environment. I'll show how they make that transition with a legacy model and system in place. I'll discuss their concerns, mistakes, and benefits realized as a result of using Rational XDE Developer, and I'll look at the effort required, the features lost or gained, and the impact of the changes on the team.

After reading these articles, you should be able to:

- understand Rational XDE Developer's role in different types of projects
- see how Rational XDE Developer and Rational Rose differ and how they can coexist
be comfortable with the process of migrating from Rational Rose to Rational XDE Developer

understand how Rational XDE Developer can integrate with other tools already in use on your project

Below is a roadmap to this series. Each article will include this list, linking to the other articles as they become available.

- Part 1: introduction to the team and the challenges they face
- Part 2: how Rational XDE Developer fits into the evolving project, and its impact on process; the transition from Rational Rose to Rational XDE Developer
- Part 3: Rational XDE Developer's collaborative and usability enhancements
- Part 4: Rational XDE Developer's advanced features and how they benefit the ASDI project
- Part 5: a review of lessons learned; how to tailor Rational XDE Developer usage to specific types of projects; a summary of the article series

A final note: Keep in mind that the ASDI project is fictional and that real-world requirements will be different. To determine whether Rational XDE Developer fits your needs, try downloading and using an evaluation copy of it. Only by using the tool can you really get a feel for its value; these articles express only one perspective.

Revisiting and Evaluating the ASDI Project

The earlier series of articles, "Applying Rational Tools to a Simple J2EE-Based Project," lays out the fictional premise that we're a software company, Lookoff Technologies Incorporated, that was hired by Audiophile Speaker Design, Inc. (ASDI) to meet their burgeoning IT requirements. ASDI wanted us to work with them to reengineer their legacy system. Although they were a very talented team, their process was rigid and not conducive to the rapid changes and the reengineering task that they were about to undertake.

The ASDI project was carried out by a small team and was limited by both schedule and budget. Quality had to be top-notch, since this system would go into production as the supply chain management software for this speaker design company.

The system went into production as hoped, and ASDI was very pleased with the finished product. We then solicited a follow-on support and maintenance contract from them. As ASDI's business grew, they required more maintenance and support from us. Eventually, it reached the point where we had two developers on the ASDI site supporting all IT operations for the system and network. Our customer's speaker business was very successful, and online operations increased their revenue, largely through
We responded as quickly as we could when the customer made software change requests. However, the process that had led us through a successful 60-person-month development project was not well suited to our on-site maintenance and enhancement work. With such a small development team, our analysis and design artifacts were getting out of date, and sometimes we were making quick changes without understanding the full impact, consequently introducing bugs and instabilities into the production system.

It was important to ensure that the production system ran smoothly and reliably, since it was a critical portion of the revenue chain for the company; however, the customer was making frequent, substantial change requests. With increasingly outdated documentation, it was becoming more and more difficult to make changes. There was some concern that our mature framework was evolving into a system that would be hard to maintain.

Having a cumbersome software product worked against our long-term strategy to market the product to other small businesses in the high-tech industry. Although tailored to a speaker design company, the system was actually a flexible product that could be of great interest to small- and mid-sized inventory-based companies.

We set out to identify the problems in our current process by looking for areas in which the tools were impeding our productivity. We needed to figure out just where our design and development time was being spent. We knew we had to keep the design model up to date, but there were clearly some problems that were keeping that from happening.

Although we knew our problems couldn't be solved only through tools, we hoped there were some tools that could at least make our development and model maintenance easier. When we learned about Rational XDE Developer, we thought it justified further investigation. We had already significantly tailored our environment with Rose Extensibility Interface (REI) scripts, and we knew the Rational software tools (namely Rational Rose, IBM® Rational® PurifyPlus, and IBM® Rational® ClearCase®) inside out. Although nervous about moving to a new integrated tool, we decided to weigh the tradeoffs.

**Analyzing the ASDI Development Process**

During the initial development contract, the ASDI project had been run according to the Rational Unified Process (depicted in Figure 1). This process worked very well, and led to the holy grail of software delivery: we came in under budget and ahead of schedule. Moreover, the customer was very happy with the end product.
During implementation, the project followed the steps shown in Figure 2. The highlighted steps are those that we expected to focus on as we moved into maintenance and enhancement. In reality, however, our maintenance process ended up focusing heavily on the "Update Code" and "Mini-Builds" tasks, with mini-builds going straight to production!

Our carefully planned process seemed to be weakening as a result of schedule pressures and the fact that although each change was "just a small change," there were a lot of them. With better planning, we could have implemented the changes more efficiently and effectively.

It may seem as if our original delivered product lacked sufficient analysis.
This could be, but I believe the main reason for our weakening process was the rapid growth of our customer's business combined with their desire to add significant new functionality. We had deferred implementing certain features during the original contract due to a lack of budget or priority, and some of these features were now being addressed under the maintenance phase. The customer's "wish list" was much longer than the current budget would allow, and we also had ambitious goals for our product. It was time to tighten up the process and address some of the technological deficiencies that we had identified.

We decided to dedicate some research and development to **refactoring** the system, to ensure that it could easily be tailored and internationalized for different customers and markets. Refactoring is the task of reorganizing, reducing, and removing code to improve cohesion and minimize coupling. Basically, it involves cleaning up the code to adhere to good functional or object-oriented principles. (You can find out more about it at [Martin Fowler's site](http://martinfowler.com).) Refactoring is an important skill in software engineering, particularly as more of us inherit legacy systems that require substantial improvements.

So, in addition to our existing maintenance team (again, with two developers -- one senior and one intermediate), we proposed having an R&D team with additional developers for doing refactoring and more; Figure 3 shows this organization.

![Figure 3: Maintenance team and proposed R&D team](image)

We established the R&D team and gave it these specific goals:

- Assess the existing ASDI code and model.
- Improve the quality of the code.
- Work out an arrangement that allows concurrent access to the source repository both off-site at ASDI and on-site at the R&D lab.
- Factor out all customer-specific business logic and constraints so
that we can easily tailor this system to different customers.

- Internationalize the software to support non-English-speaking customers.

The Bad News

The R&D team first reviewed the project with the original development team members and the maintenance team. A number of deficiencies came to light as the result of these early conversations.

- **Too many tools** -- The team was finding it cumbersome to deal with so many tools: database modeling software, Rational Rose, Rational PurifyPlus, a third-party Java IDE, IBM® WebSphere® Studio's administration and deployment tools, and Rational ClearCase. Although it was possible to integrate some of these tools, the developers still found that they had to switch frequently between tools to do their job.

- **Outdated design** -- The team found that the RTE was not convenient; right or wrong, they were not incorporating every last method signature update into the design. Over time, the implemented system was departing further and further away from the last design snapshot. Without a good design and associated impact analysis tools, it was becoming difficult to judge the complexity and merits of future requirements changes. Furthermore, the R&D team was concerned that their refactoring work would be difficult without an up-to-date system model.

- **Steep learning curve** -- New team members were overwhelmed with having to learn Rational Rose, the Unified Modeling Language (UML), the development environment, the business domain, and the system we had built. We thought we had scheduled enough time with a two-week hand-off, but that proved to be insufficient.

- **No time for R&D** -- The team also wanted to try some new technological and process advances, but there never seemed to be enough time or budget for that. With a long list of small fixes always in the queue, it was difficult to justify time for prototypes and software evaluations.

The Good News

In its discussions with the development and maintenance team members, the R&D team also heard about the positive aspects of the ASDI project -- but even then there were some negative observations.

- **ClearCase Windows integration** -- The team was pleased with the ClearCase Windows integration, and felt that this level of integration needed to be extended to other tools they were using.

- **Rational Rose and REI** -- Rational Rose, particularly its script customization through REI, was a definite plus. The maintenance
The Rational Edge -- August 2003 -- Migrating a J2EE project from Rational Rose to Rational XDE Developer v2003

The team used REI scripts to automate a number of tasks (although they used them less and less as they fell behind in their model updates).

- **Rational SoDA** -- The development team was very pleased with IBM® Rational® SoDA®. The maintenance team, however, didn't know SoDA well enough to be able to quickly define new reports tailored for their maintenance activities, and this further reduced their need to maintain and use the model.

- **RUP** -- The RUP had been very effective during implementation. The maintenance team recognized the value of the RUP during the maintenance phase; however, for reasons that were not yet understood, they were not following RUP principles and practicing good model management habits.

- **Early prototyping** -- The development team felt that their early prototyping efforts had been key to the success of the ASDI project. The maintenance team wanted to continue these efforts to ensure that the system evolved with the company. Lookoff was willing to take the plunge and invest some R&D to enhance the product.

### Wish List for R&D

In the end, the maintenance team lead came up with the following wish list for the R&D team:

- Simplify synchronization of the model with the system as changes happen.
- Find a strong IDE that ties in with the model and the existing source change management system (ClearCase).
- Reduce the number of required tools.
- Support multiple partitioned models to manage the larger project.
- Improve reuse across projects and developers by sharing code snippets, patterns, and templates.

### Identifying Room for Improvement

The R&D team hoped that Rational XDE Developer could address a number of the issues they faced, and they wanted to explore the tool further before making any recommendations. Since the product was quite new at the time, they found the "First Encounters with Rational XDE Professional" articles to be helpful as an overview. They listed their major requirements and concerns related to the ASDI project, and tried to identify those that could be met by Rational XDE Developer (Table 1).

**Table 1: Matching Rational XDE Developer capabilities to project deficiencies**
<table>
<thead>
<tr>
<th>Concern</th>
<th>Rational XDE Developer capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronizing the design with every little code change took a lot of</td>
<td>Rational XDE Developer automates synchronization, thereby ensuring that code updates are worked into the design and that design updates are applied to the code.</td>
</tr>
<tr>
<td>effort.</td>
<td></td>
</tr>
<tr>
<td>The analysis artifacts were outdated.</td>
<td>Although updates to analysis cannot be automated, the analysis model is integrated into the environment, so it's easier to make changes to it. Rational XDE Developer's support for multiple models (with integration and traceability between models) enables concurrent model management.</td>
</tr>
<tr>
<td>ClearCase integration was adequate within Windows Explorer, but it would</td>
<td>Rational XDE Developer integrates tightly with ClearCase, making ClearCase a transparent part of the IDE while providing change management within the tool.</td>
</tr>
<tr>
<td>be better to have it integrated with the team's third-party IDE.</td>
<td></td>
</tr>
<tr>
<td>The data modeling tool was a separate application. The development team would sometimes modify SQL or the schema directly rather than go to the single PC that ran the database modeling software.</td>
<td>Although Rational Rose Enterprise has data modeling, for cost reasons the project ended up purchasing only Rational Rose Professional J Edition (which also provides some data modeling functionality). Rational XDE Developer includes database modeling functionality within it, avoiding the need for a separate tool.</td>
</tr>
<tr>
<td>Rational Rose capabilities were more complex than the team required.</td>
<td>Rational Rose has significant functionality and comprehensive UML support; Rational XDE Developer provides similar modeling capability, but the model can be set up so that only a subset of the capabilities is visible to the developer.</td>
</tr>
</tbody>
</table>
The system needed a significant amount of refactoring.

Built-in refactoring features in Rational XDE Developer and WebSphere Studio Application Developer make it easy to rename entities, move and extract code to and from methods, encapsulate field access, pull methods up into a parent class, and more.

Useful code snippets and patterns had emerged during implementation but had not transferred properly to the maintenance team.

Rational Rose has some capabilities in this area, but Rational XDE Developer further tightens up the integration of code templates and patterns.

Summary

We knew we had let our process lapse, and we were also not watching for tool advances as closely as we could have. It's always tempting to let things slide like this at the tail end of a development project, but if the product is important to you or your customer, it can't be excused. In our case, the product was important to us and our customer. We foresaw that we could market the system to multiple customers, so it had to be easy to maintain and adaptable to change.

Our existing approach was satisfactory, but our standards were now very high, and we wanted to be sure to maintain our software quality. Maintenance had slowly eaten away at the coupling and cohesion of the design, and the system was no longer the same as the one captured in the latest system model. We wanted to fix that, and we saw that a combination of technology and process could help us.

Rational XDE Developer was one tool that we saw as a potential candidate. We hoped that migrating to it would be safe, since we were moving from one tool to another from the same vendor. We also hoped that by using Rational XDE Developer we would decrease the number of tools and the complexity of mundane tasks.

Part 2 will look at Rational XDE Developer in more detail and show how a project can smoothly make the transition from Rational Rose to Rational XDE Developer after the project is underway.

References

Articles by the author:

- "Applying Rational Tools to a Simple J2EE-Based Project" (a ten-part series)
- "First Encounters with Rational XDE Professional" (a six-part series)
For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!