Define application architectures with Rational Software Architect: Part 2: Iteratively refine the architecture

Jean-Louis Marechaux

October 11, 2011

This series presents techniques for creating models to specify and communicate the architecture of software-intensive systems. It illustrates the elaboration of the software architecture for a fictional company, Yummy Inc. Using an iterative approach, it describes the key architectural activities that are necessary to specify a software-intensive system with IBM® Rational® Software Architect. Part 1 of the series explained how to create a vision for the architecture during early stages of your project. In part 2, we describe how the architecture is iteratively refined using Rational Software Architect. Both articles assume that readers are familiar with methodologies based on iterative development.

Introduction

Architecture envisioning is an agile best practice to outline the technical decisions that guide development and testing of software-intensive systems. A software architect typically uses the following steps to outline the target architecture of a software-intensive system (see the Resources section to access part 1 of this series):

1. Identify significant requirements
2. Define the candidate architecture
3. Define the initial deployment model
4. Define the domain model

But architectural thinking is not a one-time activity. It is a continuous effort that experienced agile teams integrate into their development work. As your team progressively builds the system, you uncover new technical challenges, and you have new architectural decisions to make.

The following architectural activities are typically performed during development iterations:

5. **Refine architectural mechanisms:** Technical concepts to satisfy architecturally significant requirements identified in the first step.
6. **Refine design elements:** The architecturally significant design elements
7. **Refine the deployment architecture**: The deployment units and topologies

**Iteratively refine the architecture**

Understand that typically you do not need to specify the design for all the parts of your system. Just like everything else in a project, you only do it if it brings some value. But some parts of software-intensive systems must be specified in greater details than others. It might be because some components are more difficult to understand and implement, or because they impact critical elements of other subsystems. Some organizations need more detailed specifications before they hand off the development to a business partner or a geographically distributed development team. Some teams have to specify and document their architecture more carefully to fulfill regulatory requirements.

**Refine architectural mechanisms**

When your team needs to specify the design of a component, you can start by defining the architectural mechanisms to realize it.

The objective of this stage is to define the analysis classes necessary to realize the significant requirements of your system. In other words, you want to identify all the elements that are necessary to fulfill a specific business need.

For a better visual representation, you can choose to associate those classes to an analysis stereotype. Use the **Boundary** stereotype to represent a class that acts as an interface to the system. Use the **Control** class to describe a component that exercises control over other classes. Use the **Entity** stereotype to designate a class that carries data.

A graphical representation of these analysis stereotypes is shown in Figure 1.

**Figure 1. Analysis stereotypes**

![Figure 1](https://example.com/figure-1.png)

Figure 2 shows the analysis stereotype as it is applied to a significant requirement of the system. In the Yummy Inc. example, the ordering business need involves several elements (analysis classes) such as a customer, a menu, and the processing of a payment.)
Define application architectures with Rational Software Architect:
Part 2: Iteratively refine the architecture

Figure 2. Analysis elements for the ordering package

After you identify the analysis classes (figure 2), you might also need to define the static and the dynamic aspects of some significant requirement.

Then, for each significant scenario, the purpose of this step is to define both their static and dynamic aspects. Rational Software Architect assists the architect in achieving this task by providing a use-case realization template.

The template (Figure 3) comes by default with a class diagram to record the static aspect (Participants) and a set of sequence diagrams (Basic Flow, Alternative Flow n …) for the dynamic behaviors.

Figure 3. Use-case realization template for the Order Menus functionality

Each diagram addresses a different aspect of the component. The class diagram (Figure 4) depicts all the classes involved in a use-case realization (dynamic) while the sequence diagram (Figure 5) illustrates the interactions between them (dynamic).
At the end of this step, you have developed some significant requirement realizations of the system with UML models. It is an important part of the architecture when you need to depict the static structure and the behavior of some components of your software-intensive system.

Because the Analysis Model is still technology-agnostic, you can reuse it as the starting point for several implementations.

**Refine the Design Model**

After you define how you want to realize significant requirements in the analysis model, you are ready to dive into the concrete design of the components.

While the Analysis Model is abstract, the Design Model must be closer to the real implementation. The software architect typically identifies a set of goals and constraints related to the technology
that is used to implement the solution. The *Design Model* is usually not developed from scratch. It is an evolution of the analysis model that you already defined (figure 2 and figure 3).

In our example, Yummy Inc. Online Catering is developed using the Java EE platform. Data persistence leverages the Java Persistence API (see Related topics), the application logic is implemented with Session Beans, and access to the delivery service is done through web services.

In architecture and design, a widely adopted best practice is to define the different layers of the solution, and how they relate to each other. The Java EE $n$-tier architecture encourages the separation of concern between layers.

In the Architectural Layers perspective of Rational Software Architect, the design template provides a view where you can define the dependencies between the layers (Figure 6).

**Figure 6. Architectural Layer Dependencies view for the Online Catering system**

With the architectural layers defined, the software architect produces the first iteration of the design. The Rational Software Architect design template provides a pre-defined structure for this activity. The component specification typically contains the interfaces and data used to interact with the component (Figure 7). It is usually the first aspect that a software architect defines. For each component, it is important to specify the data that it manipulates and its interface, including the available operations.
Figure 7. Design packages for the Online Catering system

Note that because the implementation platform has been chosen at this stage, the design model can be enriched with technology-specific information. In our example, persistent objects are marked with a JPA stereotype to identify how the object model maps to the database.

For some component of your software-intensive system, you might decide to specify the design a step further. Remember that you add details to the design only if it helps your team implement and deliver the application. In figure 8, a detailed class diagram has been created with a specific objective in mind. The team wants to generate Java code from the design model using some of the transformations that Rational Software Architect provides (see Related topics for further information about the transformations).
Creating the detailed design for some components is not always the responsibility of a software architect. Depending on the size and skills of your project team, the software designer role can be assigned to one or many different persons, such as senior developers (the boundary between architecture, design, and development is sometimes unclear).

**Refine the deployment architecture**

Architects are responsible for creating systems that fulfill business needs. One key aspect of software-intensive systems is how they are deployed on the physical architecture. The initial deployment model is usually defined when you envision the architecture (see the Related topics section to access part 1 of this series). As the project progresses, architects refine the deployment architecture to incorporate the non-functional requirements and the constraints related to the production environment.

With the Rational Software Architect deployment planning tools, you can create topologies to show the relationships between information technology resources. You can also plan and validate deployment scenarios (see the Related topics section for more information about deployment planning and automation with Rational Software Architect).
Rational Software Architect helps achieve deployment analysis at different levels of abstraction. You can create logical topologies where you capture decisions about the operational architecture of a software-intensive system (figure 9). In a logical topology, you describe the system at a high level of abstraction. You specify how the components of the application are organized and connected, and where they will be located and hosted. Note that the units representing the solution are not always created from scratch but are derived from UML elements for better traceability.

**Figure 9. A logical topology example for the Online Catering system**

Architects can also define a topology to describe how the application is deployed. This type of model describes a specific, complete deployment instance of an application and the specific infrastructure on which it is deployed (Figure 10).
Incrementally build the system

Teams working to deliver a software-intensive system must always keep in mind that working software is the primary measure of progress. Activities required to develop an application is beyond the scope of this article, but it is important to remember that successful teams integrate architecture and design into their development work.

Ongoing architectural thinking is part of the development effort, and teams should not write new code or modify existing one without thinking about how the change will affect the whole application. A popular technique to modify existing code is called refactoring (see Related topics for further information about refactoring). Rational Software Architect provide a rich set of tools to facilitate code development and refactoring. You can change the structure of your code by using refactoring assistance (Figure 11). You can also identify patterns and anti-patterns in your application code by using the architectural discovery capability (Figure 12). Both features helps you write better code and discover problems at early stages of software development.
If you have created some models during your architectural thinking activities, you can also decide to transform them into code. Rational Software Architect ships with a set of transformations to generate Java code, Java EE elements, or SOA artifacts (Figure 13). You can also generate models from existing source code.

As you incrementally build the system, you create the working software that conforms to the design decisions made during architectural thinking activities.
Summary

Agile methodologies promote the adoption of short iterations to build a system piece by piece. If Big Design Up Front (see the Related topics for BDUF) has proven to be a non-efficient approach mostly adopted in waterfall models, it does not mean that architecture and design activities must be banned from agile projects. Design must be both intentional and emergent. Intentional because at the beginning of a project, you anticipate some of the technical needs based on the product backlog (architecture envisioning). Emergent because during development iterations, you adapt and enrich the design based on the new technical challenges that your team discovers.

In part 1 of the series, we focused on typical tasks to outline the architecture and to align the technical vision to development needs. In this second part, we have described how the architecture is refined during development iterations. For the whole project lifecycle, Rational Software Architect provides model templates to specify the architecture of a software-intensive system from different viewpoints (Figure 14)

Figure 14. Models for a variety of architectural views

Architectural thinking is a continuous effort that experienced agile teams integrate into their development work. It does not always result in formal diagrams. Keep in mind that if UML is not the right approach in your context, nothing prevents you from using Rational Software Architect sketches instead, just like you would do on a whiteboard. Rational Software Architect sketches are less formal than UML diagrams but they may be a good alternative for your specific project.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yummy RSA models</td>
<td>Yummy2011.zip</td>
<td>84KB</td>
</tr>
</tbody>
</table>
Related topics

- Learn about modeling with Rational Software Architect in this free self-paced training.
- Learn about the Design Management project, a new solution to help team collaborate on architecture and design activities.
- Learn about OpenUP, the open source process that applies iterative and incremental approaches within a structured lifecycle.
- Watch a short video to learn more about IBM agility@scale.
- Read this article to learn how to create deployment topologies with Rational Software Architect, Deployment topologies with Rational Software Architect.
- Read Architectural Blueprints—The "4+1" View Model of Software Architecture, a paper by Philippe Kruchten.
- Explore the official site for information about UML.
- Download a trial version of IBM Rational Software Architect from developerWorks.
- Evaluate IBM software.

© Copyright IBM Corporation 2011

Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)