JeeWiz in the Rational Unified Process (RUP)

Jon Hurwiz

Consultant - *JeeWiz!*

Introduction

The Rational Unified Process is the Rational Corporation's version of Unified Process Development. RUP, as it is usually abbreviated, is an approach to the software development process backed by a development framework. It also has a supporting toolset developed by Rational, which is sometimes also referred to as RUP. JeeWiz is a systems generation tool developed by NT/e. This paper discusses how JeeWiz can be used in a RUP environment, predominantly where it fits into the process and only touching on compatibility with the Rational toolset. It presupposes no knowledge of either RUP per se or JeeWiz and gives an overview of both before coming to the discussion of the application of JeeWiz in RUP projects. The paper is written at a non-technical level and is aimed at development and project managers considering the use of JeeWiz in a RUP project.

An Overview of RUP

The RUP approach can be exemplified in a series of principles which include:

- Deliver value
- Address risks early
- Accommodate change
- Focus on delivering code, not specifications
- Use cross-functional teams

It is important to understand these principles before becoming focused on the software engineering cycle and on the bewildering level of detail available through the supporting toolset. I will add another principle which although implied in RUP too often seems to get lost in practise

- Never try to use everything RUP has to offer in a single project

RUP is not a single-track development process which if followed slavishly ends up in the best possible system. Which aspects of RUP you decide to use on a particular project need to be thought through up front and are likely to vary from project to project. An experienced project manager will structure the RUP project plan bearing in mind the principles, deciding what needs to be included and what excluded. This configuration is itself an important part of RUP.

The RUP development framework contains four phases:

- **Inception**: addresses what am I going to build, why, and how; how much will it cost and what are the benefits, what are the risks.
- **Elaboration**: understand the system in enough detail to mitigate technical risk and solidify cost estimates; try out the architecture and development product set.
- **Construction**: iterative development of complete products with increasing functionality.
- **Transition**: get the system used.

Each phase has deliverables and milestones and each phase is iterative. An iteration includes all aspects of a development lifecycle and expects to end with some aspect of the system delivered to the user, although an Inception iteration would clearly involve less coding than a Construction iteration. Part of the plan will be to consider how much iteration you think you want. You can even plan to iterate the entire project cycle. The objective is to try things out and get feedback as early as possible. This falls under both the headings of addressing risk early and accommodating change. Change toward the end of a project costs more than change toward the beginning.

An Overview of JeeWiz

JeeWiz is a product toolset allowing the business application to be separated from the technical architecture of deployment. By automating the repetitive aspects of software development it produces superfast systems generation from a design specification, even when only a minimum level of specification has been achieved. Out of the box architectures, include Java-J2EE and C#-.NET, but JeeWiz is capable of configuration to handle many other platforms, architectures and local standards.

JeeWiz uses a series of pattern templates to convert a design specification to the required system framework. The deployment architecture is also generated from pattern templates specified by the architect. This means that the architecture can be decided on and tested out independently of the business functions of the application, and more importantly the application can be worked on independently of the architecture.

The system generation itself can provide a complete multi-tier model, from database tables to input screens, and business logic can be supported by data-view objects combining to provide the boundary layer. It normally achieves this generation by cascading the various tiers using the default architecture specification. The generation can also be limited to a single tier connection, facilitating otherwise complex integration, or varying multi-tier patterns can be coded by the architect for specialised deployments.

Because it is capable of understanding the technical environment, JeeWiz can also generate systems based on multiple languages and platforms and for deployment on multiple server types.

The approach used by JeeWiz to understanding the specification is also pattern based, and while it supports UML object models exported from several popular toolsets, it will also support other formats such as XML. Some business logic not inherent in a UML model can also be picked up from separately provided XML documents which act as the native specification level. Extra code can be written in the target language such as Java or C# to be inserted into final system. This can be developed in the application developers' favourite IDE or toolset.

Once the architecture and application are specified, generation is rapid and systems become available to deploy with the minimum of effort. For the version of JeeWiz available at the time of writing, NT/e estimate that 70% to 95% of the coding effort is automated, depending on the type of system being generated.

How JeeWiz fits into RUP

JeeWiz fits well in the RUP philosophy, addressing many of its major principles.

Delivering Value

Value delivery in RUP starts with the question of whether a system should be developed or not, and it provides development checkpoints where the business users can decide if the system is worthwhile. Because it can deliver a prototype with the minimum of information, using default values, JeeWiz allows business users to get a flavour of what the system will do early enough to plan a change of direction if required.

While it is a maxim of the RUP to avoid unnecessary scrap and rework, in a standard RUP environment it is expected that some of the prototypes developed will have to be thrown away and recoded. Using JeeWiz this becomes unnecessary as the prototype is part of the application which will enhanced in later phases.

RUP recommends that projects achieve an effective return on investment through tool automation. JeeWiz is just such a tool enabling the fastest of cycle times from change in specification to the desktop, and unlike a “quick fix” JeeWiz automatically fills in all stages, including generating the connectivity code and updating technical documentation.

Addressing the Risks Early

In RUP it is recommended that at least one possible technical architecture be identified during the Inception phase, and that the architecture is proved during Elaboration. In practise, many systems will be constrained to use platforms already available, but it is still necessary to check their suitability for the project in hand. By decoupling the architecture from the application, JeeWiz allows a system architect to try out various constraints independently of the business analyst. He can performance test and tailor the platform if necessary without holding up the business design. These two aspects can be brought together in JeeWiz at any time.

The decoupling can also shorten the critical path as design no longer need wait from either the final architecture specification or development standards. The insistence that an unstable architecture needs to force another iteration of the Elaboration phase, possibly leading to delays in the project, is mitigated, and the project manager can make a judgement to continue knowing that an application development using JeeWiz can move architectures without recoding. Although some care needs to be taken if there is the possibility of changing the target language used to encode inserted business logic.

Accommodating Change

In RUP change is handled, even embraced, using iteration. It is expected that all aspects of the software development process will be undertaken many times during a RUP project. This can be highly time consuming and increases the levels of regression testing dramatically. The goal of JeeWiz is to automate as much of the development process as possible and as well as speeding the development cycle JeeWiz provides hooks for automated system and integration testing.

Because it is systems generation tool, Jeewiz automatically propagates changes throughout the system framework.

A second more defensive recommendation in RUP is that systems are built with components which encapsulates data and the functionality acting on the data. This minimises the impact on the application of changes to the data or the way it can be manipulated. In reality encapsulation is often handled poorly and functional decomposition causes many links into the data pool. JeeWiz ensures that the basic system components are generated in the desired architecture, isolating data access through a clear entity layer. Even language-coded business methods access the data through this layer ensuring system resilience.

Focusing on Code Delivery

A RUP project manager is warned that producing a specification in itself delivers no value to the business. The progress of a system's development should be measured only by generating and delivering code. It is only when the user sees it running and okays it that the specification is validated. This rather harsh aspect of RUP is largely fulfilled within a JeeWiz project. No matter what the architecture used for prototyping a specification can be immediately turned into code, so real executable code begins from the very first iteration. This means that the specification production does not cloud the project manager's mind giving a false sense of security. Paralysis by analysis is almost eliminated as specifications can be quickly viewed as part of the usable system.

One reason given for this RUP principle is to stop analysts producing everything that RUP tells you how to produce. Projects structured around JeeWiz focus on code production automatically. The code becomes a view of the specification.

Using JeeWiz in the RUP

JeeWiz should be used right from Inception. Because a RUP project can produce simple prototypes even during the Inception phase, it is worthwhile setting up JeeWiz as a prototyping tool. Only the most critical use cases will be identified at this stage, but even minimal information shown working on a screen can firm up understanding of some of the system scope. This is particularly true on greenfield systems, where computer support for the functionality is completely new to the user. At this point feedback can also be gained on the look and feel of front-end screens.

In Elaboration the system architect creates or validates the platform patterns, perhaps generating volume scenarios to ensure that the structure will be robust, while the business analyst might prototype business cases. JeeWiz screen templates and styles can be tailored to help "sell" the system in. All critical use-cases and most of the major ones are identified and for all but the smallest of systems placed in a modelling tool. Most classes are identified and a class diagram produced. Interfaces to other systems will need to be built and tested.

The initial build can be on the target platform if the system architect is ready or on a separate development environment if required. If architecture patterns are already available for a development environment, JeeWiz can profitably generate prototypes with full connectivity, even if that environment does not mirror the final target.

With around 80% of use cases identified by the end of this phase, and around 90% of the classes, the class diagram as a minimum should be exported into JeeWiz to generate the system framework. Models in Visio, Magic Draw, System Architect and Rational Rose all successfully link into JeeWiz, but JeeWiz supports some modelling tools more fully than others depending on the ability of the tool to accept plug-ins. Rose, Rational's own tool for UML modelling, is supported by the use of plug-ins: one to allow the inclusion of certain extra tags in the class and component diagrams, and one to facilitate the export of the model into JeeWiz.

It is in Construction that JeeWiz really shines. Most of the use cases will not have been implemented in the elaboration phase and on average around 60-65% of the projects effort would normally be spent in this phase. With JeeWiz the analysts concentrate on the business logic, the various flows and constraints and feed this in to the underlying structure already built by JeeWiz. Builds can be done daily as recommended in RUP, or even more frequently if required.

Because much less work is required in Construction, typically the phase with the highest staffing requirements, fewer coders are needed and teams are smaller and easier to control. RUP recommends small multi-functional teams, but recognises that multiple teams will have to be used on some projects all held together by the architecture group. JeeWiz tends to limit the systems that need to be built this way to the very largest.

As more functionality is designed and implemented, the focus will move from functionality to feature polishing – choosing the best balance between "only just usable" and "Rolls-Royce". JeeWiz provides a consistent look-and-feel through different style sheets and templates which can be tailored for the whole system.

There is little that JeeWiz can achieve in the Transition stage following deployment, where the system is rolled out, the users trained and the business processes changed to incorporate the systems use. However, if a second system cycle has been planned following a project review, JeeWiz holds all the current system information and even major functionality can be integrated into the system with little effort.

Subsequent maintenance cycles using JeeWiz also allow changes in the system almost independently of system size. Even with good object and component design it is not always possible to isolate the effect of small changes to big systems. JeeWiz propagates changes during systems generation even if they have wide ramifications throughout the code. A maintenance cycle involving migration involves far less work when using JeeWiz. In simpler cases complete migration may just require updating the architecture patterns for the new platform and regenerating the system, especially if the system test tool is compatible.

Conclusion

The RUP is independent of any of the toolset used to support it, but with multiple iterations and frequent builds it becomes important to be able to automate the development and test cycle for all but the smallest project. JeeWiz fulfils this requirement while offering many benefits over and above a standard code generator.