The IBM Rational Unified Process solution in action
White paper
June 2007

Rational. software

IBM

The IBM Rational Unified Process solution at the Tivoli Rome Laboratory: a RUP pilot implementation.

Claudio Marinelli
Alex Donatelli
Agostino Colussi
Alessandra Masci
Lucio Bortolotti
Scot MacLellan
Jack Wilber
Francesca Guccione
Sara Severoni

Contents

Introduction

The chances of successfully implementing a change within any organization increase significantly with the amount of motivation for the change and the passion the organization brings to the effort. The IBM Tivoli® Rome Laboratory recently embarked on such a change as it adopted the IBM Rational® Unified Process®, or IBM RUP®, solution.

As the largest software development facility in Italy, the Tivoli Rome Lab has a staff of about 550 employees, including software developers, test engineers, project managers, architects, designers, system engineers and IT specialists.

While the desire and commitment of this diverse team were vital to the successful implementation of RUP, so too was the team's practical approach to assessing needs and properly scoping the effort. By focusing clearly on the organizational changes that were required and by carefully defining the scope of those changes, the Rome lab enabled an efficient adoption of RUP that minimized disruption to the group's existing environment. As a result, the initiative is yielding a significant return on the group's investment of time and effort.

This white paper describes the experiences of the Rome lab in implementing RUP, detailing the motivation for the effort, its major steps, the key decisions made along the way and, ultimately, the results the lab has achieved.

Starting and scoping a RUP pilot

Before beginning the RUP pilot project, the Rome lab took time to assess its operational environment, identify goals and select an appropriate project.

Motivation

Sound business strategy requires a thorough understanding of organizational needs before any major change is instituted. At the Rome lab, this understanding came from a comprehensive assessment of the group's operational environment. After conducting this assessment, the lab identified three areas in which it observed significant opportunities for improvement.

First, different functional teams within the lab were using inconsistent methodologies and modeling languages. This hampered collaboration and communication, and made it difficult for new team members to come up to speed quickly on projects.

Second, many projects at the Rome lab were not using a well-defined methodology. Instead the teams were inventing—or reinventing—the methodology as they went. The time spent working on developing a methodology for each project added overhead, and the resultant processes were not always optimal.

Third, the teams' requirements, analysis and design activities were not automated, and there was no link between the process activities and the development tools used to perform them. As a result, the lab lacked the ability to trace project requirements to analysis, design and implementation artifacts.

Taken together, these operational limitations affected the overall productivity of the team as well as the quality of the team's deliveries. An example of one such limitation was the late discovery of defects. Too often, the lab's development teams identified design and other defects late in the development process when they were more difficult and time consuming to fix.

Goals

The task of finding effective means for improving in these areas fell to the Technical Leadership Community, whose principal aim is to identify, explore and analyze new solutions for improving productivity and quality at the lab. Led by the Rome Central Architectural team, the Technical Leadership Community recommended the adoption of RUP, and set forth several goals for the group's first RUP project.

One of the principal goals of the pilot project was to acquire a deeper understanding of how introducing RUP can impact a real software development project. As part of this effort, the team would measure short-term benefits, long-term benefits and return on investment (ROI), while assessing the risks that the Rome lab might face in its RUP adoption.

At the same time, the pilot project was to provide a foundation for future RUP projects. In pursuit of this objective, a main goal of the pilot project was to train and mentor the development team, and enable its members to simplify and accelerate RUP adoption throughout the lab going forward. The pilot project would also help the Rome lab gain a better understanding of how to customize RUP to better fit the lab's specific needs. Lastly, the community urged the team to publish all findings from the pilot project to an internal Web site, so that the results could be accessed and leveraged by other Tivoli development teams.

Choosing a pilot project

The Rome lab used the Process Engineering Process (PEP) to help guide the implementation of RUP on the pilot project. PEP establishes a set of process engineering practices for adopting RUP in a software development organization, and it includes guidance on tailoring RUP to meet the particular needs of the organization.

According to the PEP, the first RUP project is a critical factor in successfully introducing RUP into an organization; a successful pilot project will lead to more success, while an unsuccessful pilot project can cause an organization to postpone RUP adoption, sometimes indefinitely.

PEP outlines several approaches to implementing RUP and advises organizations to select the approach best suited for their internal environment and situation.

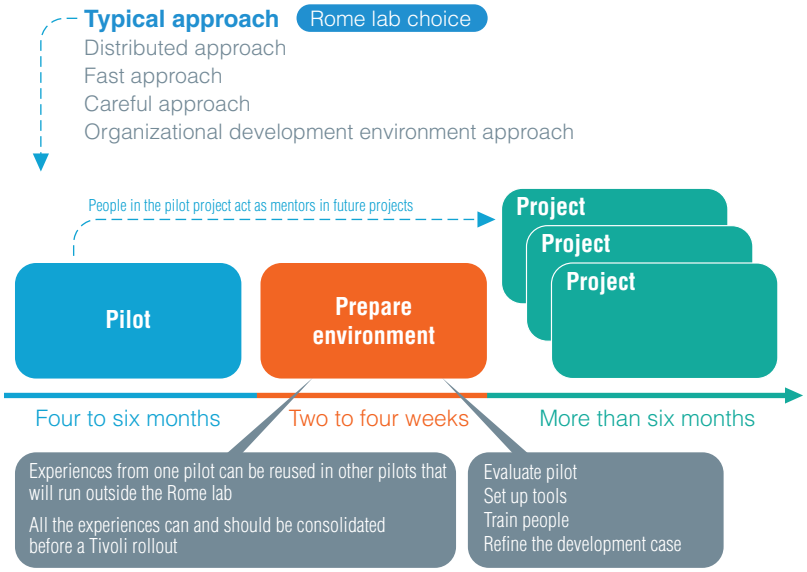

Figure 1: The Rome lab selected the "typical" approach for its RUP pilot.

The Rome lab selected the "typical" approach, which is the most commonly used. As PEP notes, this approach is especially applicable when the organization is skeptical about process change.

Once the approach was selected, the lab identified an upcoming project that would serve well as a RUP pilot. The group selected IBM Tivoli Configuration Manager, Version 5.1 software, based on two main criteria. First, this was an entirely new development effort, with no legacy code base—and the team felt that introducing RUP on a project starting from scratch would be easier. Second, the project relied on a Java™ Platform, Enterprise Edition (Java EE) architecture, and RUP incorporates documented best practices for Java EE development.

Complementing the Tivoli Process Development Model

Prior to its RUP pilot project, the lab had been following a development model used throughout the Tivoli organization, the Tivoli Process Development Model (TPDM). TPDM effectively defines roles, artifacts and milestones for a software development effort, but gives the development team a high degree of freedom in determining how to carry out design and analysis activities.

One of the primary aims of the pilot project team was to use RUP in a way that complements TPDM instead of conflicting with it. In practice, the two development models fit together well: TPDM defines artifacts and activities in use at the Rome lab, and RUP provides the details on how to create the artifacts and perform the activities.

In addition there was some overlap in the best practices outlined in TPDM and those in RUP. For example, *developing iteratively* in RUP closely matches the staged approach the Rome lab used with TPDM. Similarly, the *manage requirements* best practice in RUP is completely consistent with the interaction design activities already being applied by teams using TPDM.

In short, the Rome lab uses TPDM to define *who* does *what* and *when* (that is, which roles produce which artifacts for which milestones, checkpoints or stages), while relying on RUP to define *how* to use best practices, guidelines and tool mentors. Along the way, the team mapped and consolidated the roles, artifacts and activities in RUP with the corresponding elements already in use in TPDM.

Customizing RUP

While tailoring RUP to complement TPDM, the Rome lab sought to focus its RUP implementation on the specific disciplines identified by the operational assessment as underperforming: requirements, analysis and design, implementation, and environment.

The lab followed the guidelines of the environment discipline, modifying them slightly to create baseline and iterative process customizations. The lab also customized the requirements, analysis and design, and implementation disciplines to focus only on the artifacts that were most relevant to the lab's targeted improvement areas. This set of artifacts formed the core of the lab's pilot customization. Figure 2 shows the customization scope in the context of RUP and the Enterprise Unified Process (EUP).

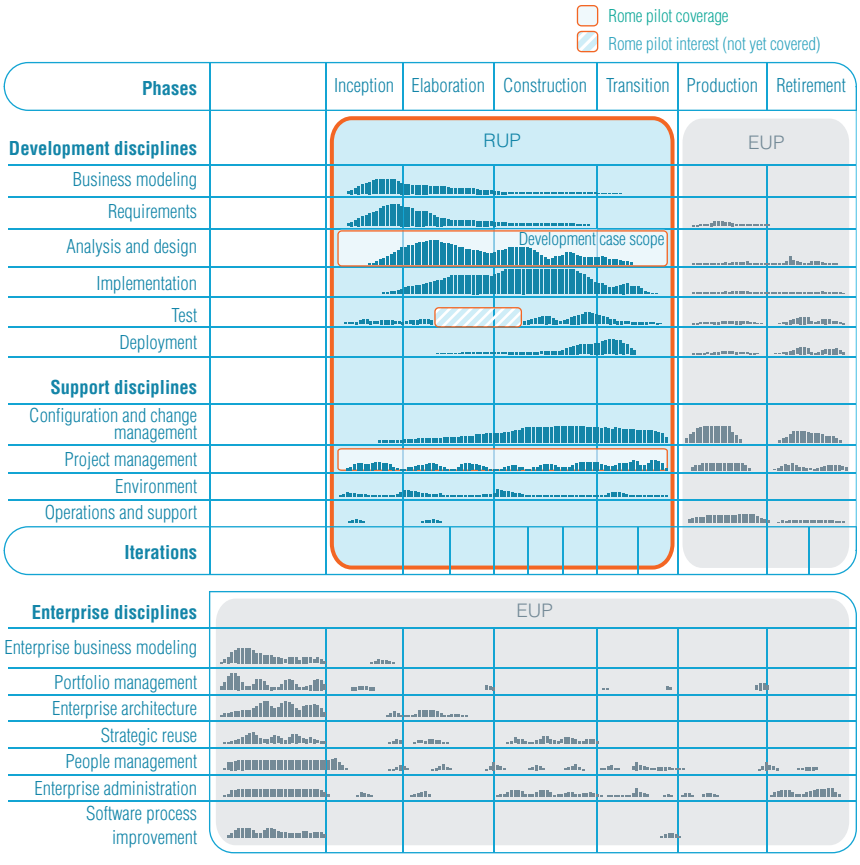

Figure 2: Development case customizations focused on specific disciplines

In determining the scope of the customization, the Rome lab decided to include only those disciplines and phases that affect artifacts directly relevant to the developer. The term "developer" in this case encompasses multiple RUP-defined roles, including architecture reviewer, database designer, designer, designer reviewer, implementer, implementer reviewer, requirements reviewer, requirements specifier, software architect, system analyst, user experience designer and user experience reviewer.

The scope of RUP customizations has been limited to meet the needs of the Rome lab's developers. It focuses on providing guidance on the essential requirements, analysis and design, and implementation activities, as well as artifacts, from the developer perspective. As a result, the RUP elements that the Rome lab considered relevant to developers are described in detail in the process, while those considered less relevant are described only briefly or omitted altogether.

Core artifacts

The Rome lab intentionally focused on a set of core artifacts to serve as steppingstones to implement the system as envisioned. The customization includes other artifacts, but these are either contained within or directly related to artifacts in the core set.

Once the group had selected these core artifacts, they included only the process activities that support the production of the artifacts in the customizations. The result was a set of seven core artifacts, each a model representing a complete description of the system from a particular perspective. The models are complete in that there is no additional information required to understand the system from a particular perspective, and no two models overlap.

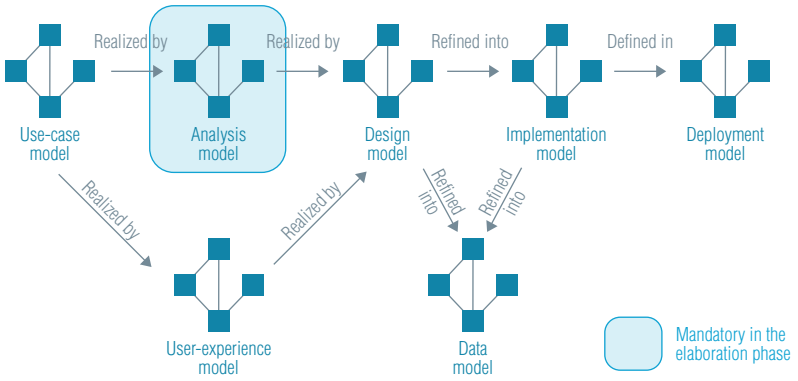

Figure 3: Core artifacts of the Rome lab's RUP customization

Figure 3 provides an overview of these core artifacts—use-case model, user-experience model, analysis model, design model, data model, implementation model and deployment model—and illustrates the relationships that exist between them.

Analysis artifacts are fundamental to communicate decisions and the system architecture to the rest of the team. However, the analysis model is mandatory only during the elaboration phase to help prevent "analysis paralysis"—a condition reached when analysis is overdone and the system is overengineered. The goal of the Rome lab's analysis activities is to produce a first approximation of the system as quickly as possible to demonstrate that the most relevant architectural use cases are addressed and included as part of the solution.

Traceability

The Rome lab formally defined relationships between different artifacts by tracing their dependencies—both between disciplines and in the context of a single discipline. The RUP customization introduced specific activities to define and address traceability during the development process.

The customization focused on traceability as a key objective because the Rome lab's experience has shown that the ability to trace requirement artifacts through the stages of specification, architecture, design, implementation and testing is a significant factor in creating quality software. The ability to track these relationships and analyze the impact of a proposed change is shared by many modern software processes focused on delivering quality results. Figure 4 illustrates the dependencies within and between different disciplines.

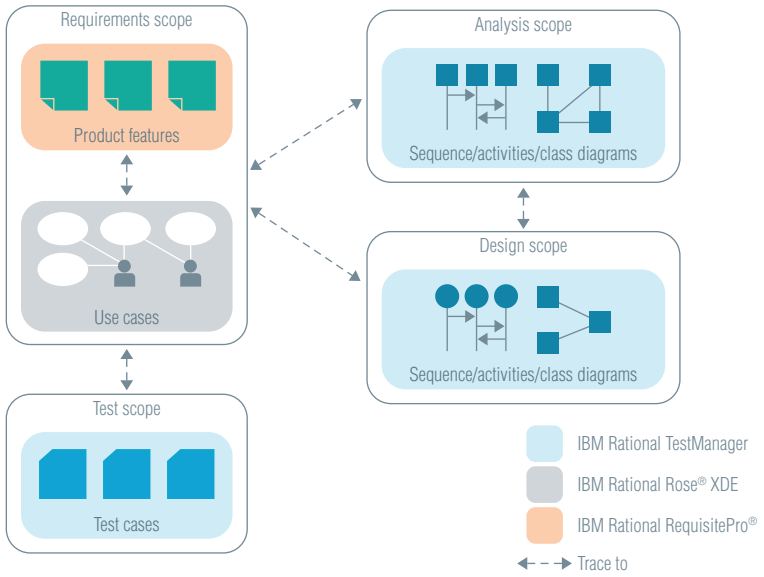

Figure 4: Traceability between disciplines

A framework for IBM Rational solutions

Many activities in RUP can be automated through the use of software development tools, enabling an organization to minimize time-consuming and error-prone tasks. The Rome lab saw the pilot project as an opportunity to try a RUP-driven approach to adopting the IBM Rational Software Delivery Platform, including IBM Rational XDE™ Developer and IBM Rational Software Architect software for visual modeling and model-driven development; IBM Rational Functional Tester software for automated testing; and IBM Rational Method Composer software for delivering, managing and documenting processes and best practices.

The Rome lab viewed the uncoordinated adoption of such tools outside a RUP framework as less than optimally efficient. In contrast, adopting Rational development solutions with the support and coordination of a well-structured set of best practices like those embodied in the RUP help the Rome lab make the most of process and tools in improving the overall quality of its deliverables. To reduce training and startup time, RUP includes a set of tool mentors that provided the Rome lab's team with step-by-step guidance on how to use a particular tool to complete a task. Tool mentors provided the link between the process and the tools used in the projects. By clearly indicating how to use tools within process boundaries, tool mentors simplified the adoption of new tools, while giving the organization freedom to adopt tools on an as-needed basis.

Integrating RUP and TPDM

Introducing a development process in an organization typically involves overcoming cultural and psychological barriers. When an organization has an established development process, such as TPDM, already in place, the difficulty is compounded.

Introducing RUP as another development model required careful positioning with respect to TPDM. Because the Rome lab was not seeking to replace TPDM but to complement it, the team had to customize RUP and adapt it to TPDM through a loosely coupled integration.

A loosely coupled approach reduced the time, effort and cost involved in integrating the two processes, while minimizing the intrusive impact of adopting a new process. The approach does not alter the core activities and artifacts already in place; instead, it introduces activities and new artifacts to improve areas only partially covered by TPDM and areas that are in obvious need of upgrade.

By combining RUP and TPDM, the lab was able to leverage its substantial investment in TPDM, build on its skill and experience, and continue to use its existing development model. At the same time, the lab used RUP to address specific areas that needed improvement. And it implemented a flexible integration pattern that allows changes to either of the two processes to be easily incorporated.

Mapping phases and milestones

As one of the first steps in integrating RUP and TPDM, the Rome lab mapped RUP phases and milestones to existing TPDM activities, checkpoints and development change processes (DCPs). While RUP provides a great deal of freedom in customization, the Rome lab team viewed the architecture-first approach of RUP and the milestones between the RUP elaboration and construction phases as a fundamental necessity. The group also saw milestones, phases and iterations as essential to RUP, and sought to adopt them as soon as possible by mapping them to TPDM as follows:

- ***The inception phase*** *in RUP maps to TPDM's* define the product *activity, including the Concept DCP milestone and interaction design activities. The RUP inception milestone maps to the TPDM requirement checkpoint.*
- ***The elaboration phase*** *maps to TPDM's Plan DCP and some of the stages established during TPDM's* plan the stage *activity. The elaboration phase includes a set of stages addressing the most relevant use cases from an architectural point of view. The elaboration milestone has no corresponding milestone in TPDM; development teams are free to determine their own elaboration milestones based on the stages included in the elaboration phase.*
- ***The construction phase*** *maps to the remaining TPDM stages and requires the development of the remaining use cases.*
- ***The transition phase*** *naturally maps to the* test and release the product *activity of TPDM.*

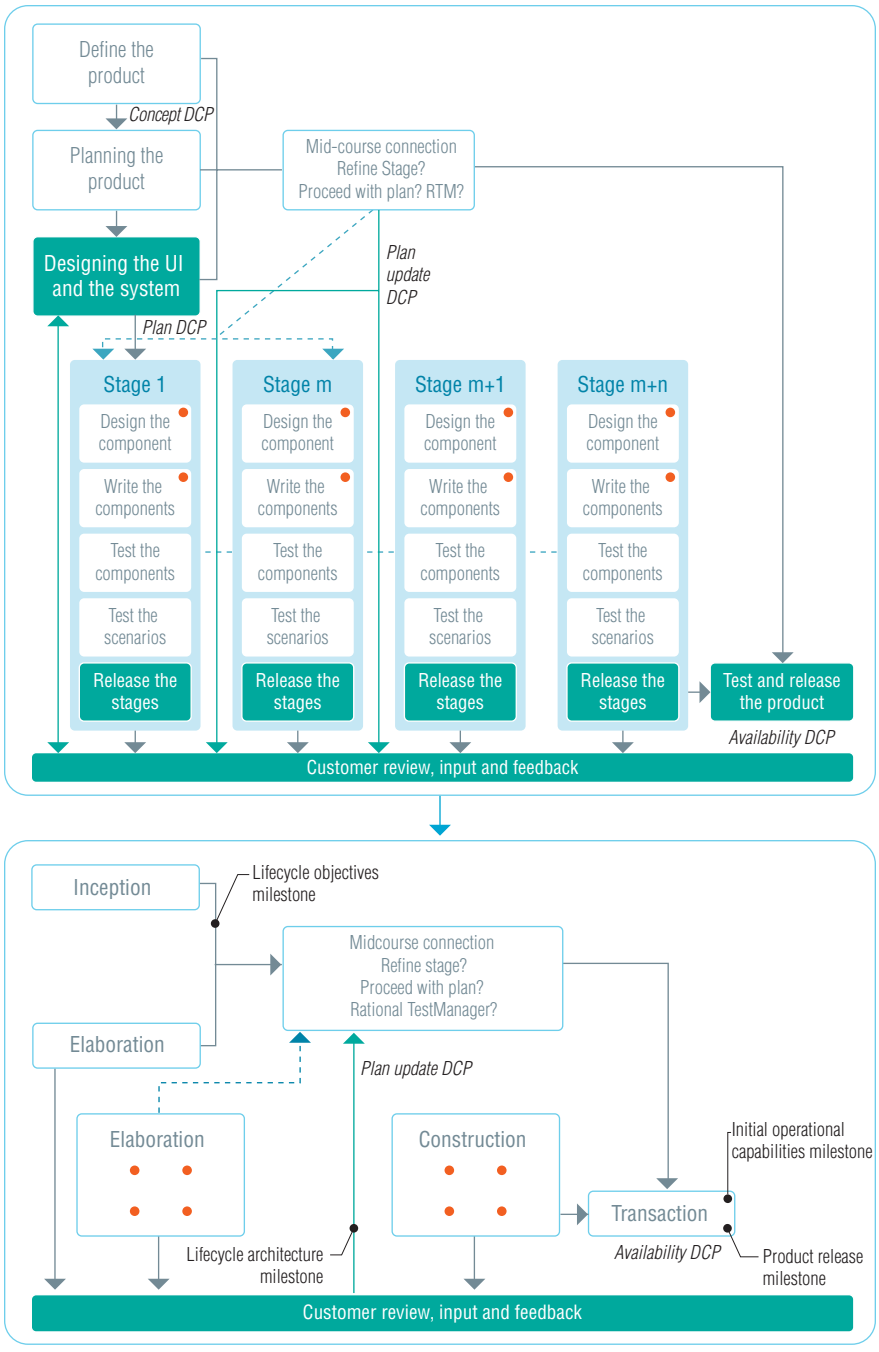

Figure 5: Mapping RUP phases to TPDM

Integration strategy

To adopt RUP into the existing TPDM environment as seamlessly as possible, the Rome lab developed a roadmap for RUP customization, shown in figure 6.

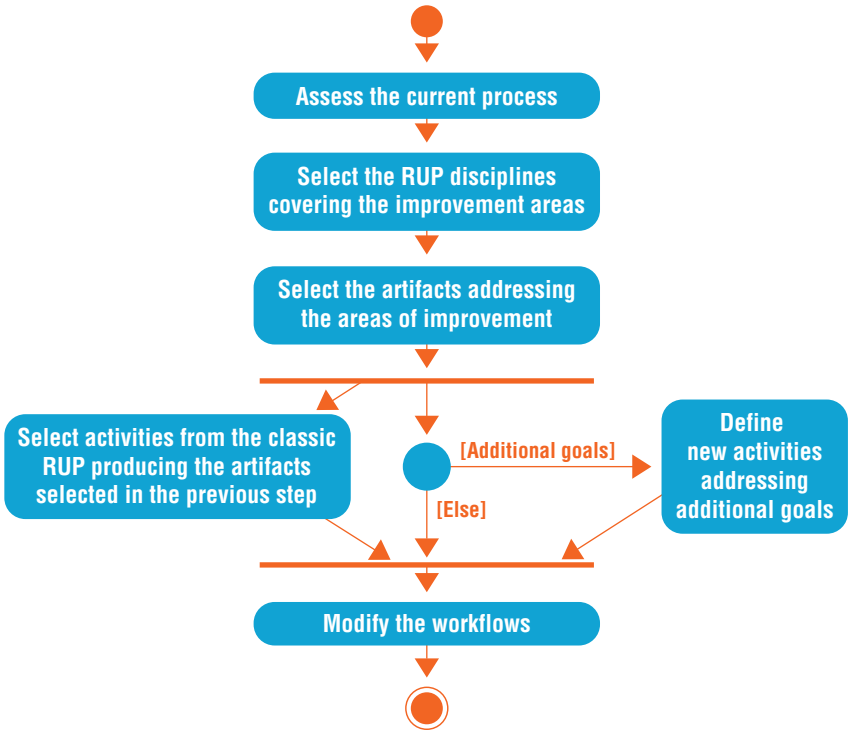

Figure 6: Rome lab's RUP customization roadmap

In the first step, the Rome lab assessed its current process to identify areas for improvement. The objective was to use RUP to address weaknesses, not to change what the team was already doing well.

The lab then identified the RUP disciplines that addressed the areas to be improved, including the requirements, analysis and design, and implementation disciplines. Although TPDM addresses these disciplines, it does not prescribe how to perform activities within them; RUP helps to standardize these activities by providing the *how* and completing the current TPDM.

Next, the team selected a set of artifacts associated with those activities. For the pilot project, the Rome lab selected artifacts directly associated with the developer role.

Based on this set of artifacts, the lab identified the specific RUP activities used to produce the artifacts. In parallel, the group defined a set of new activities to address additional goals of the customization—for example, end-to-end traceability from requirements to analysis, design and implementation artifacts.

Finally, the lab modified the classic RUP workflows to include only the activities identified in the previous two steps. In this step, RUP and TPDM are coupled with TPDM artifacts serving as inputs to RUP activities—for example, the marketing theme document in TPDM is an input to the RUP activity that produces a use-case model.

An example: customizing the requirements discipline

To gain a clearer picture of how this customization was performed, let's examine the requirements discipline.

In this discipline, the use-case model serves as the bridge between RUP and TPDM. The Unified Modeling Language (UML) and the *model visually* best practice of RUP require a reasonable level of formality to produce an effective use-case model.

In figure 7, the diagram on the left shows macro activities of standard RUP associated with the requirements discipline. These activities are mapped to the specific activities the Rome lab chose to implement in the pilot customization on the right. The standard RUP activities for analyzing the problem and understanding stakeholder needs were already covered by TPDM in the creation of the interaction design (IaD). As a result, the customization of RUP includes just two workflow actions executed in sequence:

- *Define the system.*
- *Refine the system definition.*

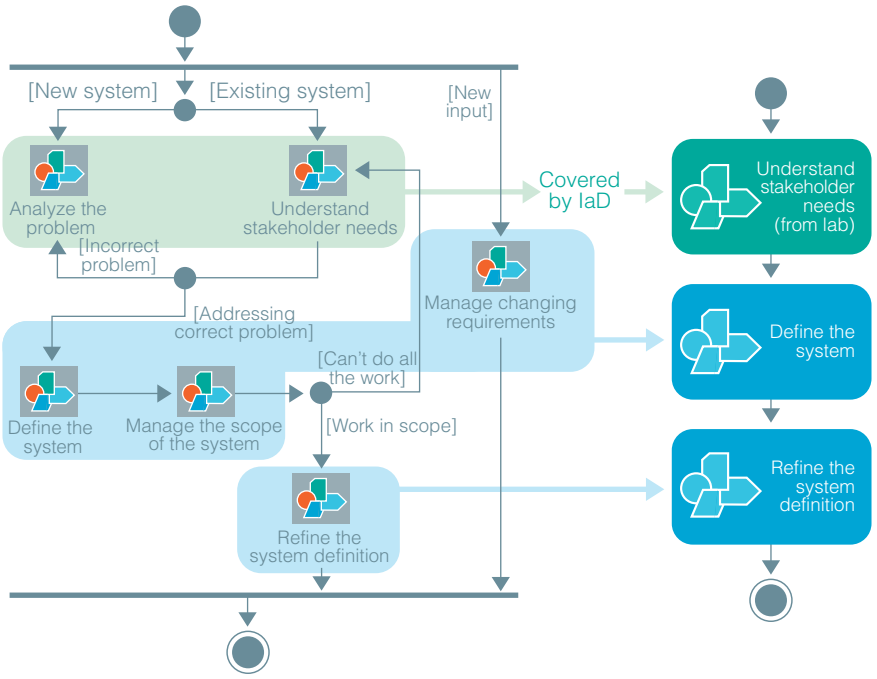

Figure 7: Customizing the requirements discipline

Defining the system

The workflow for defining the system is detailed in figure 8, which illustrates not only the activities required by the workflow, but also the input and output artifacts. This workflow produces the first use-case model refinement of the system, which is used to drive the creation of other artifacts in the analysis, design, deployment and data models.

In this workflow, the *Find actors and use cases* activity receives as input the TPDM artifacts that collect information on system requirements and produces the use-case model. By taking a TPDM artifact and producing a RUP artifact, this activity implements the link between TPDM and RUP.

The vision document collects all the requirements that are distributed across different TPDM documents. This document is also used to create and maintain the traces that link requirements to use cases.

The *Prioritize use cases* activity is needed to select the most relevant use cases from an architectural point of view, and to determine the order in which they will be realized during the elaboration phase.

During the *Review the requirements* activity, a quality check is performed to evaluate the completeness of the produced artifacts—in this case the use-case model and vision document. In particular, the requirement reviewer verifies that each requirement is traced to at least one use case.

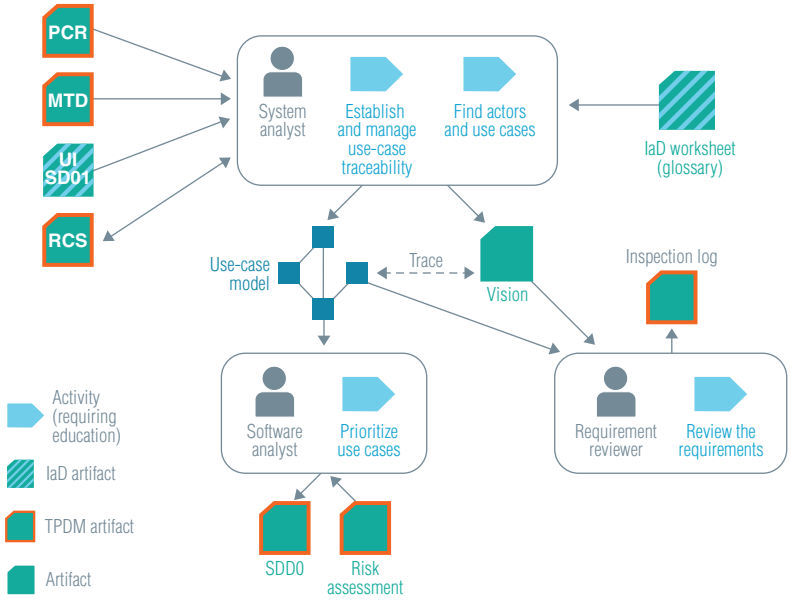

Figure 8: Defining the system

Refining the system definition

The workflow for refining the system definition is shown in figure 9. Like figure 8, this diagram illustrates the activities required by the workflow, as well as the input and output artifacts.

As part of this workflow, the *Detail a use case* activity focuses on describing one or more of the use case's flow of events in sufficient detail to enable software development to begin working on it.

For the *Structure the use-case model* activity, a system analyst refines the use-case model to include new use cases or extend existing ones.

During the *Review the requirements* activity, a quality check is again performed to evaluate the correctness and traceability links of the artifacts that have been refined. This activity is one of the most important, because traceability forms the foundation of many of the quality improvements that can be achieved through RUP and the UML.

As with the *Defining the system* workflow, the *Refining the system definition* workflow incorporates both TPDM and RUP artifacts, and it serves as another point at which the two methodologies are coupled.

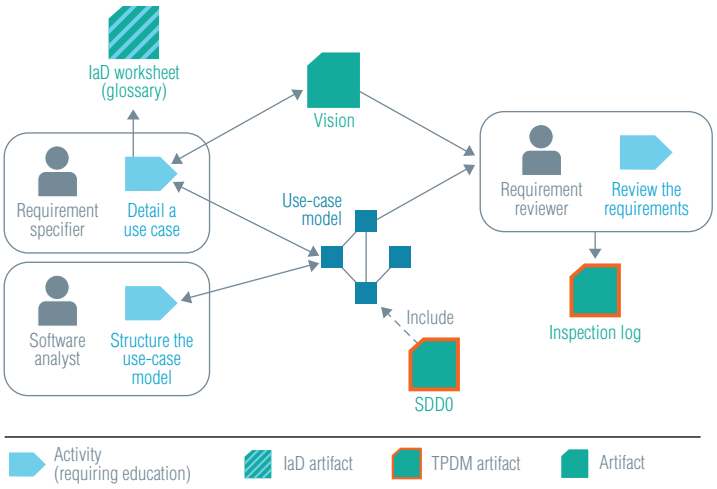

Figure 9: Refining the system definition

Assessing the pilot

Often, one of the major obstacles to the adoption of a software process improvement is executive management's initial reluctance to invest in it. This natural tendency is frequently due to the lack of convincing evidence that the process improvement will deliver a meaningful ROI.

The key questions most frequently asked by executive management and project managers are focused squarely on ROI:

"Why should we implement RUP?" And, *"How much is this going to save us?"*

After selecting the RUP pilot project, customizing RUP, integrating RUP with TPDM and executing the pilot project for almost six months, the Rome lab produced answers to these questions by fulfilling a principal goal of the pilot project: evaluating the ROI of the RUP adoption. This evaluation included short- and long-term benefits and risk assessment in two main areas: quality and productivity.

The pilot project was run for six months, and then put on hold after the requirements analysis. Design phases had been largely completed while the implementation phase was under way. At this point the team felt it had enough experience in the specific disciplines that the RUP pilot project was designed to address and could evaluate progress in quality and productivity with a high degree of confidence.

The quality evaluation focused on the quality of the produced artifacts—product quality—and not the quality of the process. Because the full development cycle was not completed for the pilot, quality measurements could not be made based on number of defects present at the end of the transition phase and defects reported by customers. However, the pilot could assess quality throughout the early phases, and no design defects were found following the design phase. In addition, the Rome lab subsequently measured quality improvements on a later RUP project (see the *Beyond the pilot: RUP for test* section, later in this paper).

In the area of productivity, the Rome lab used the Constructive Cost Model (COCOMO II) to frame the evaluation. Using this model, the team calculated a forecast of the expected productivity gains before starting the pilot. When the pilot project was put on hold, the team compared actual results of the pilot with the forecasted improvements.

Quality evaluation

The pilot project team continually assessed the quality of the artifacts they produced with respect to both functional and nonfunctional requirements. As the artifacts matured, the team performed several quality assessments in the project's lifecycle. The team evaluated artifacts as they completed the activities that produced them, and also at the conclusion of each iteration.

In addition, every time the team produced a new version of executable software, the development team demonstrated and tested it using relevant scenarios to gain a more tangible understanding of design trade-offs and to help identify and eliminate architectural defects. This approach contrasts a more traditional approach that leaves the testing of integrated software until late in the project's lifecycle, when defects are more difficult and costly to fix.

In the requirements phase, the quality management process included analyzing the requirements artifacts for:

- *Consistency between use cases and requirements.*
- *Clarity, and the artifacts' ability to clearly communicate information to all stakeholders and team members.*
- *Precision to ensure accuracy and the appropriate level of detail.*

Similarly, in the analysis and design phases, the team assessed the design artifacts to ensure the consistency of the design and analysis models. And the Rome lab assessed the end-to-end traceability of its approach by analyzing the artifacts' links to requirements artifacts and to implementation artifacts.

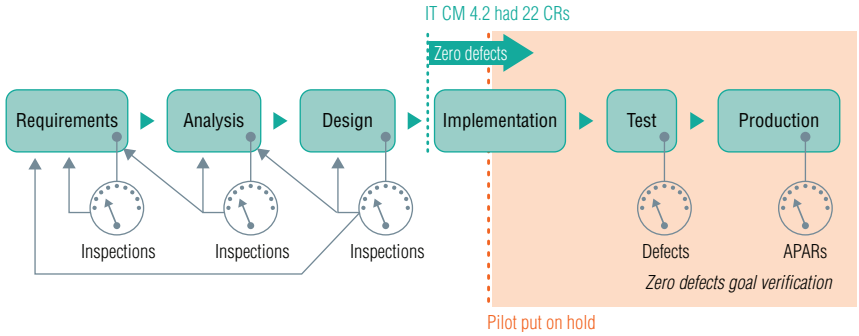

Figure 10: Quality inspections were performed after requirements, analysis and design.

Because the pilot project was placed on hold during implementation, the team did not verify the quality of the analysis performed in the requirements, analysis and design phases during the transition phase. Had the project been carried to completion, the team could have categorized defects found in the transition phase to determine whether any design defects had slipped undetected into the final product. In addition, the team also would have measured, analyzed and categorized field defects (reported in Authorized Program Analysis Reports [APARS]) to reverify that the goal of no defects in requirements artifacts and analysis and design artifacts had been achieved.

Continuously verifying quality

In pursuing a quality goal, RUP itself suggests the best practices of continuously measuring and assessing quality. At the Rome lab, verification of artifact quality was performed throughout development using a variety of techniques.

Mentors and technical leads performed informal reviews and walk-throughs on a daily basis. Similarly, team members regularly held informal meetings to review produced artifacts.

Once a week, the mentors or technical leads conducted formal inspections to assess and approve the artifacts produced by each of the project's five two-person subteams. In addition, all team members participated in a formal inspection of the overall project, again on a weekly basis.

External reviewers participated in formal reviews at the main milestones, including the completion of phases and iterations. All reviews and formal inspections produced comprehensive documentation and inspection logs that detailed the problems found.

Productivity evaluation

The Rome lab selected COCOMO II as a framework for projecting productivity improvements. As an objective cost model for planning and executing software development projects, COCOMO II supports ROI estimates with a credible basis. It defines more than 25 parameters affecting productivity, 5 of which depend heavily on the development process in place:

- *Management of complexity*
- *Use of software tools*
- *Architecture and risk resolution*
- *Team cohesion*
- *Process maturity*

The Rome lab's analysis determined that the adoption of RUP provided a positive effect on each of these parameters.

Parameter	How using RUP may improve the parameter	Productivity range improvement declared by the COCOMO model	Expected productivity increase
Management of complexity	• Introducing UML* • Change and configuration management** • End-to-End traceability*	15–30%	15%
Architecture and risk resolution	• Early architecture definition* • Component based architecture using proven design patterns* • Architecture centric approach*	0–5%	2.5%
Team cohesion	• Using RUP as common language*	0–5%	2.5%
Use of software tools	• RUP and tools are adopted in parallel*	15–30%	10%
Process maturity	• Adoption of RUP to provide an integrated process that focusses on the end product***	0–5%	0%

*Pilot goal **Already in place ***Out of the pilot scope

Figure 11: Project productivity improvements for RUP

Figure 11 lists aspects of RUP that have a direct impact on the improvement of these parameters, including the introduction of UML, end-to-end traceability, component-based architectures and the adoption of RUP and supporting development tools in parallel, among others.

The estimate does not include the contribution of change and configuration management because the Rome lab team already had best practices and tools in place to support this discipline. Likewise, the process maturity parameter was not included because the scope of the pilot project was limited to developer activities, and it did not reflect a broad adoption of RUP throughout the software lifecycle.

Summing the conservative estimates for each parameter, the Rome lab estimated a productivity increase from RUP of approximately 30 percent on the pilot project.

Measuring productivity on the pilot project

To assess the actual productivity increase, the Rome lab used source lines of code (SLOC) as the key metric to calculate the thousands of lines of code (KLOC) produced per person year (PY).

KLOC per person year (KLOC/PY) is found using the formula $[(\text{SLOC} \div p) \div m] \times 12$, where p is the number of people on the team, and m is the duration of the project in months. On the pilot project, a team of 10 developers worked for 5.5 months and produced 40,000 source lines of Java code.

As a result the productivity was measured at $[(40{,}000 \div 10) \div 5.5] \times 12 = 8.7$ KLOC/PY

On the previous release of the same product—a project that did not use RUP—the team produced 7.5 KLOC/PY. Comparing the two efforts, the Rome lab measured an average productivity increase of 16.4 percent on the RUP pilot project. To meet the COCOMO II estimate of a 30 percent productivity increase, the pilot team would have needed to produce 9.7 KLOC/PY.

When creating the pilot team, the Rome lab was careful to include a skill mix that was representative of the overall development team. This step enables a fair comparison of the productivity of the pilot team to the productivity of the entire development team.

The temporary productivity dip

When a software development organization introduces a process change, it typically experiences a temporary drop in productivity as the team adjusts to the new way of working. On the pilot project, the Rome lab team experienced such a dip in productivity and attributed it to two main factors. First, the developers were spending time learning and adopting new activities and artifacts. Second, the developers were dedicating more time to the design and analysis activities that were needed to produce artifacts—including use-case models and analysis models—which was not required by TPDM alone.

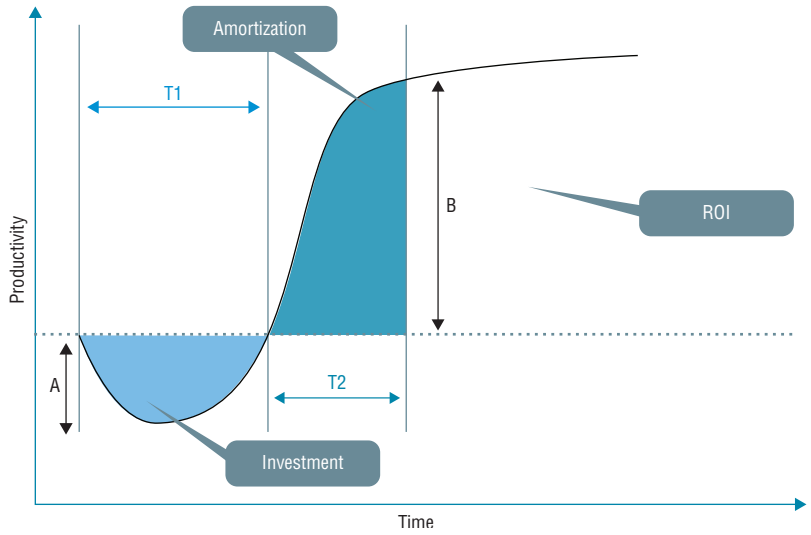

Figure 12a: A typical productivity dip

Figure 12a shows the productivity trend over time that typically results from the introduction of a process change in an organization. When the area labeled "amortization" becomes equal to the area marked "investment," the organization has reached the breakeven point, and the business case becomes profitable from there on.

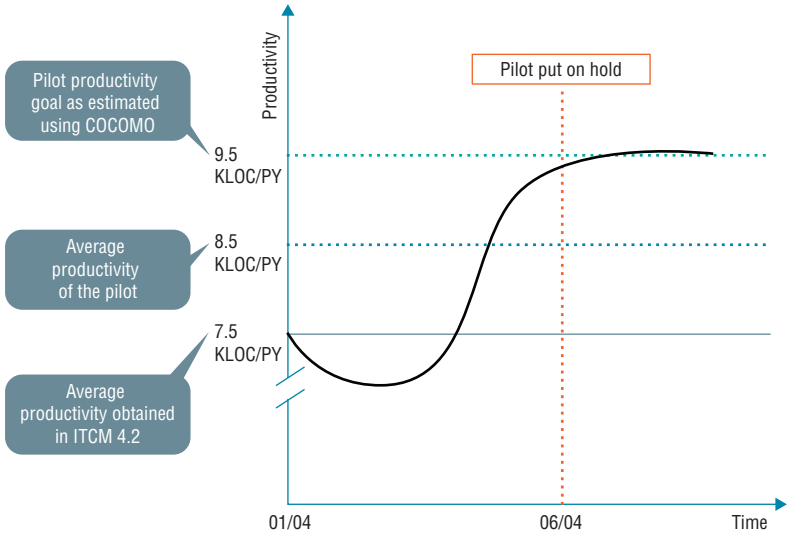

Figure 12b: Observed productivity trend on pilot

The actual productivity trend observed during the pilot is shown in figure 12b. After the initial dip, the Rome lab observed a productivity increase peak that amortized the initial loss of productivity and eventually resulted in a positive ROI.

The team attributed this later productivity increase to several factors. Once the use-case realizations had been created and the interactions between different components defined, it was easier for developers to understand the use cases and implement them. Also, the pilot team was able to reduce edit-compile-test-debug cycles, and clear communication enabled by UML led to more efficient teamwork.

The Rome lab also successfully applied a set of strategies suggested by the PEP to minimize the productivity dip. First, the team did not try to change everything—instead, it kept TPDM in place and complemented it with RUP.

Second, the team customized RUP to minimize intrusiveness and leave core TPDM activities and artifacts intact. New activities and artifacts were introduced only in areas that clearly needed improvement. Finally, the organization focused on growing institutional knowledge by conducting formal reviews and workshop meetings on RUP, producing detailed training schedules and performing daily mentorship on real cases.

Ultimately, the initial expense of adopting RUP—including costs, time spent and occasional frustration—led to significant benefits in terms of increased productivity, better quality and a more efficient team.

A more thorough calculation of ROI for RUP adoption is discussed in the section below.

Beyond the pilot: RUP for test

Following the pilot project, the Rome lab carefully analyzed and identified the project's successes and lessons learned. Gains in quality and productivity apparent in the pilot project led to the Rome lab's expanding the use of RUP to other projects, and, more important, to extending the scope of the initial RUP customization to other disciplines.

After successfully applying the initial RUP customization that covered requirements, analysis and design, and implementation, the Rome lab began an initiative to incorporate the test discipline at the same level of detail.

Goals

The Rome lab identified several primary goals for this follow-up round of customization.

First, the lab wanted to formalize a process through a graphical UML representation to make it easier to understand. In this way, employees who are new to the test discipline need only know their role and the point in the project that they are to address to see exactly what artifacts they should produce and how to produce them. More experienced testers can use the same resource as a reference, to verify detailed aspects of their tasks.

Second, the team wanted to formalize the interlock between test and development activities by defining checkpoints for improved synchronization.

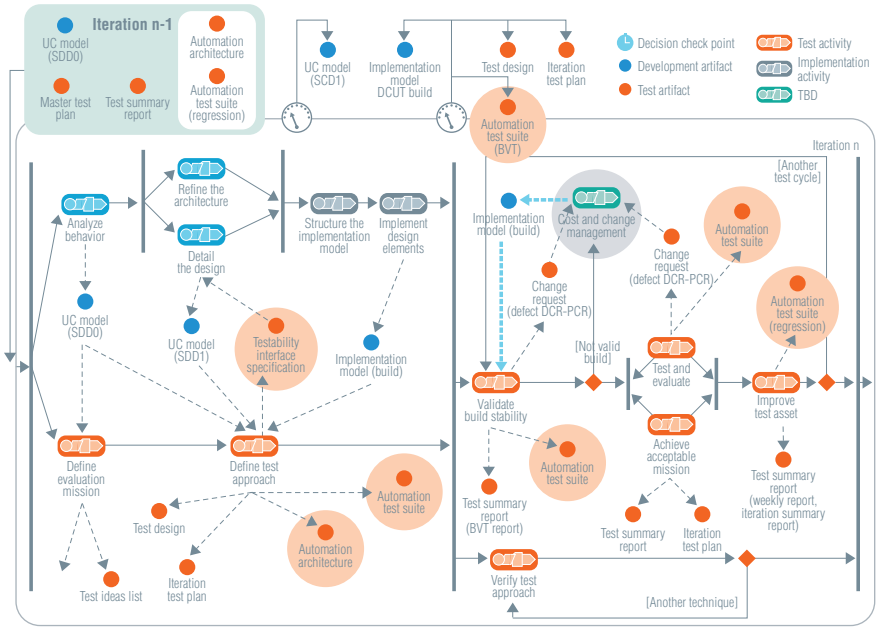

Figure 13: Formally linking test and development activities was an important goal of the customization.

The third goal of the customization was to establish a single, centralized location where all relevant test information is gathered, and to make the information accessible to all testers.

Fourth, the Rome lab sought to further streamline RUP for its specific needs, based on the experiences gained in prior projects. While continuing to leverage best practices already in use at the lab, the team looked to improve test effectiveness and test productivity by adopting new best practices suggested by RUP, specifically use-case-based testing and automation.

And finally, the team wanted to implement these new RUP for test best practices on a pilot project and measure their effect on productivity and quality.

Customizing RUP for test

In the RUP for test customization, the Rome lab followed the same integration strategy it had used for the initial RUP customization, shown in figure 6. The goal was to merge RUP with the TPDM by mapping the test roles, artifacts and activities in both processes, and by using RUP to describe how individual actors in specific roles perform activities to produce the artifacts.

Roles

To minimize changes in the way the testing team worked, the team began by analyzing the testing roles in RUP and TPDM, looking for similarities. In RUP, the roles include test manager, test analysts, test designer and tester. In TPDM, the roles are test manager, team leader, technical leader/test automation leader and tester.

The Rome lab observed that the RUP test manager roles performed some of the activities typically performed by the team leader in TPDM. Further, the team noted that in RUP the test analyst is a very technical role in which the practitioner must have significant knowledge of the product and testing techniques to evaluate overall quality based on the results of testing activities. In TPDM, these tasks usually fall to the technical leader.

The RUP test designer, as the test analyst, must also have strong technical skills. This role also involves identifying the appropriate test automation techniques, defining the test design and ensuring its successful implementation.

Based on these observations, the Rome lab created a custom set of roles in RUP for test that include test manager, team leader, test designer, test automation leader and tester.

Artifacts

As it did for roles, the Rome lab also produced a mapping between artifacts proposed by RUP and those used in the TPDM process. The team then identified those RUP artifacts that were missing from TPDM. From that set, the team selected artifacts that it believed would improve the quality of the tests and add value to the product verification process.

Although there are many artifacts in RUP that lack a corresponding artifact in TPDM, in the end, the team selected two artifacts from RUP: the test ideas list and the test interface specification.

The test ideas list was included to capture ideas that testers have at the earliest stages of the development process. These are ideas that should be explored later in the verification activities to improve the quality of target test items. Used in writing test scenarios, the test ideas list is an artifact owned by the test designer, but all testers can contribute to it.

The test interface specification artifact is used to document special requirements on the design of the product that are needed for test. This specification is used in situations where aspects of the system that are not normally visible must be observed, or where the software must be controlled in a way that is not available through its standard interface.

The Rome lab moved the production of this artifact from the test designer to the test automation leader, because the skills required by that role are helpful in defining the specification.

Another significant departure from standard RUP is the splitting of artifacts produced by the test designer. All artifacts associated with test automation activities are instead assigned to the new role of test automation leader.

Activities

The customization also required a change to the RUP test activities workflow to fit with the newly established set of integrated roles and artifacts.

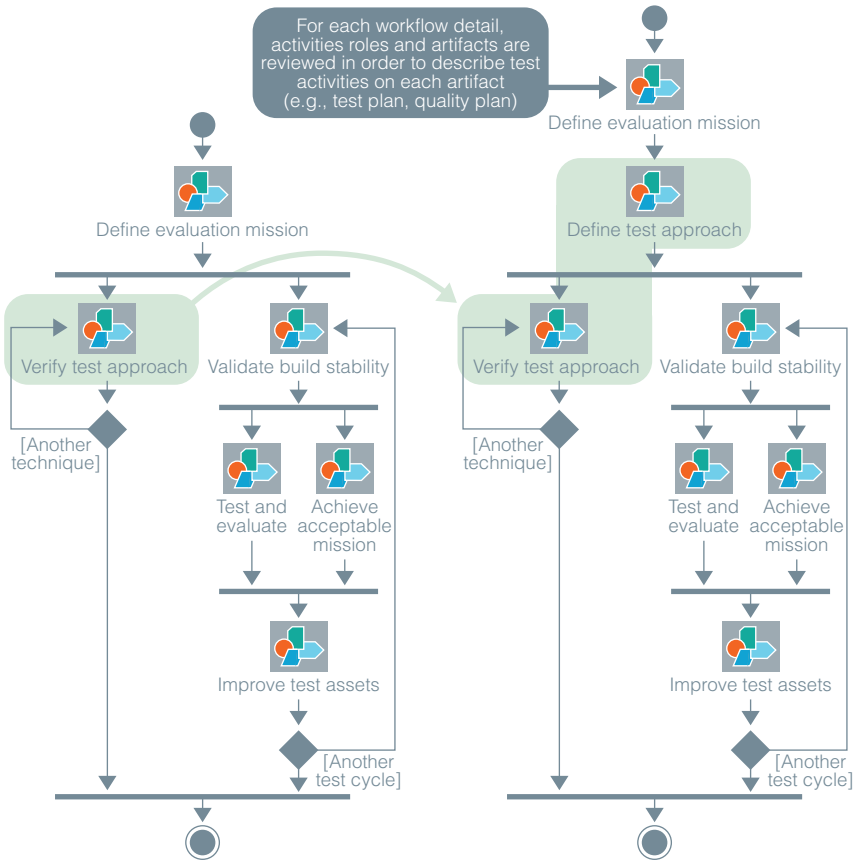

Figure 14: The customized RUP for test workflow

The Rome lab split the original *Verify test approach* activities, which enabled the team to write and review test design documents before test execution cycles start as required by the TPDM process and the integrated development process (IDP). The design preparation activities have been grouped in a new workflow element called, *Define test approach*. Activities that focus on continuous verification of the test strategy and approach still cycle in parallel with the test execution activities.

A high-level view of the mapping of TPDM to the customized RUP for test process is shown in figures 15a–15c.

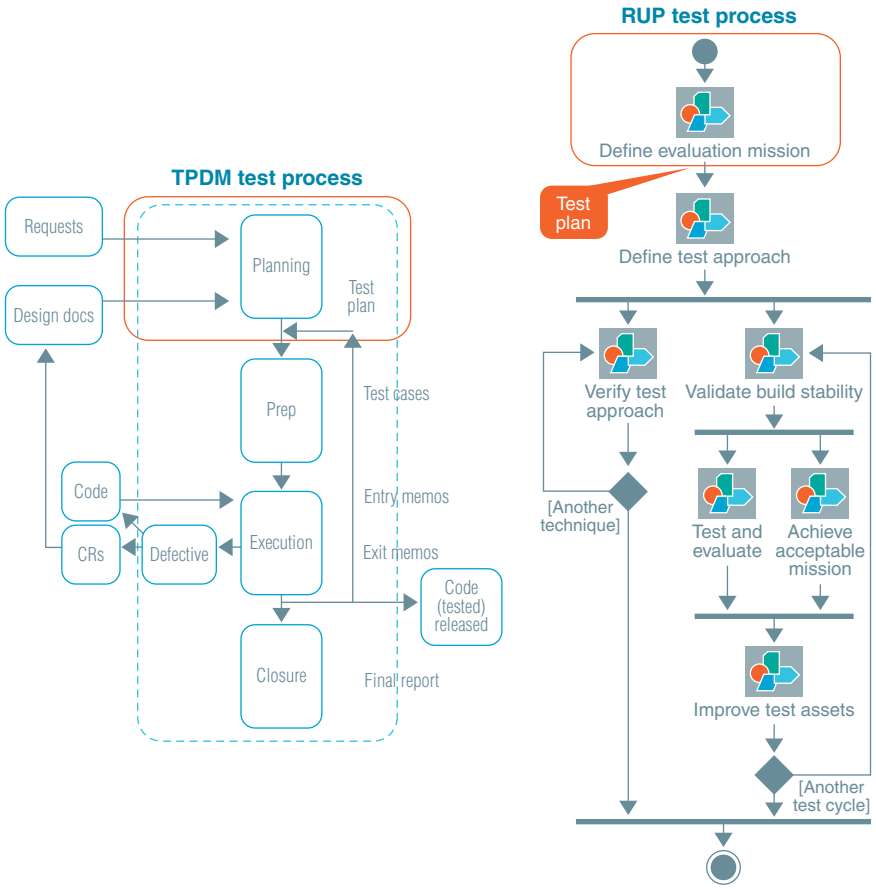

Figure 15a: Mapping TPDM to RUP for test—planning

In the TPDM planning phase, shown in figure 15a, the team defined the evaluation mission to identify the proper focus for the testing effort, and produced a test plan.

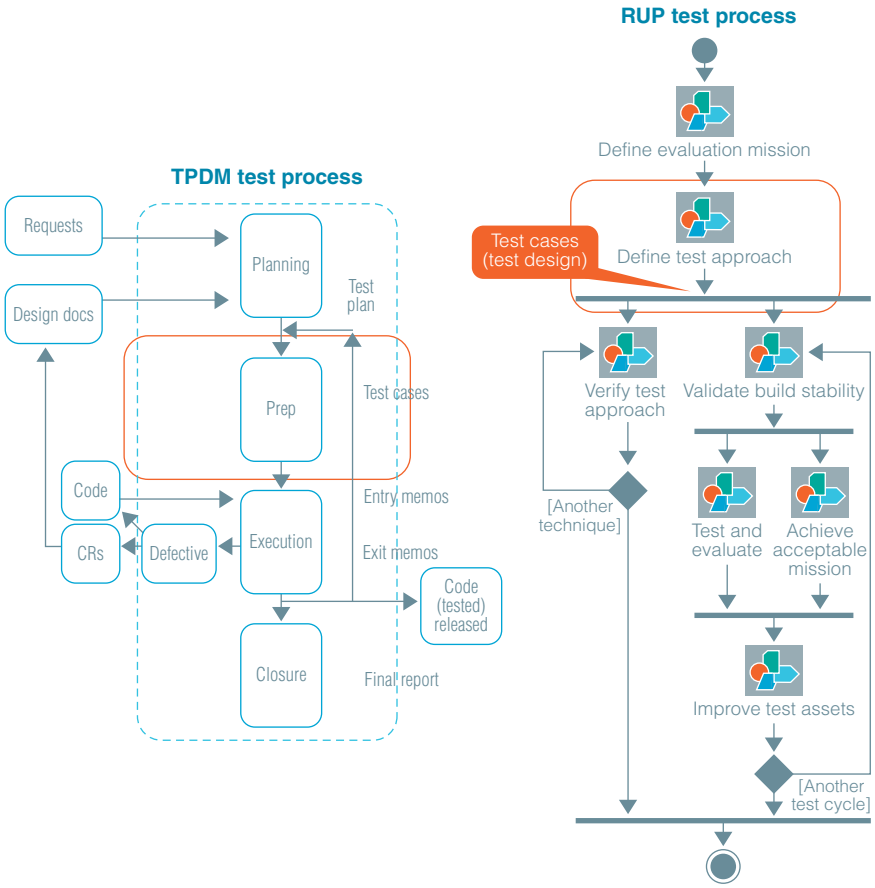

Figure 15b: Mapping TPDM to RUP for test—preparation

The TPDM preparation phase, shown in figure 15b, called for the team to determine the appropriate depth of testing. In this phase the team produced a set of test cases.

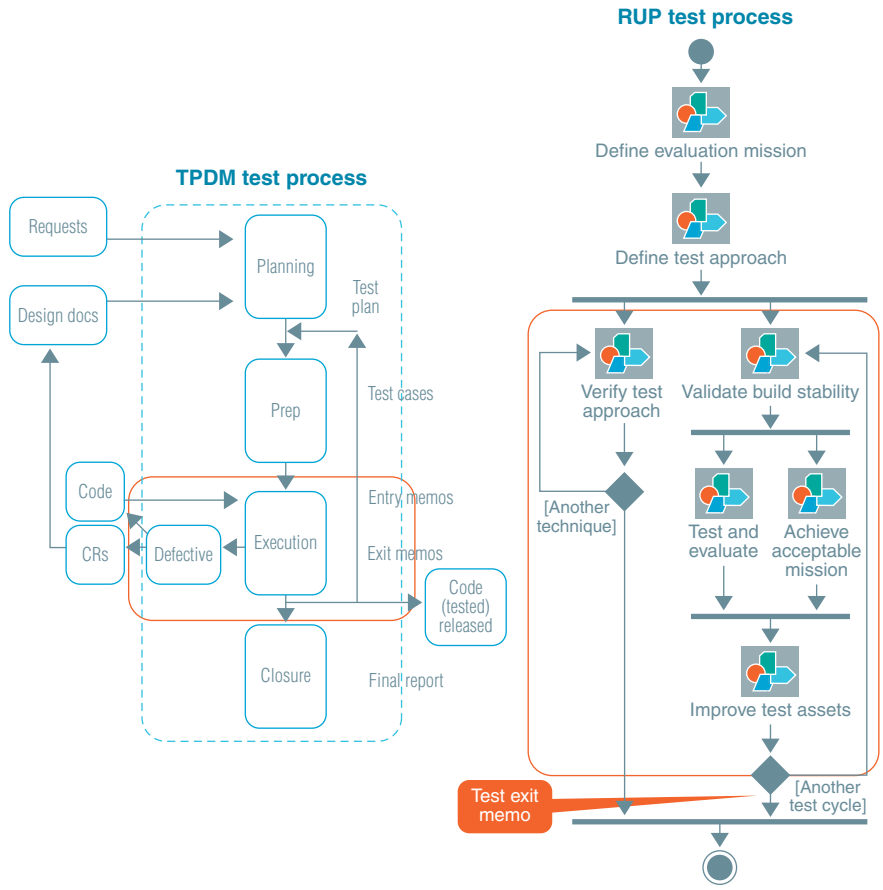

Figure 15c: Mapping TPDM to RUP for test—execution

The activities of TPDM's execution phase, highlighted in figure 15c, are vital to an iterative process such as RUP or TPDM. In this phase, results analysis enabled the team to verify test assumptions and introduce improvements in test strategy.

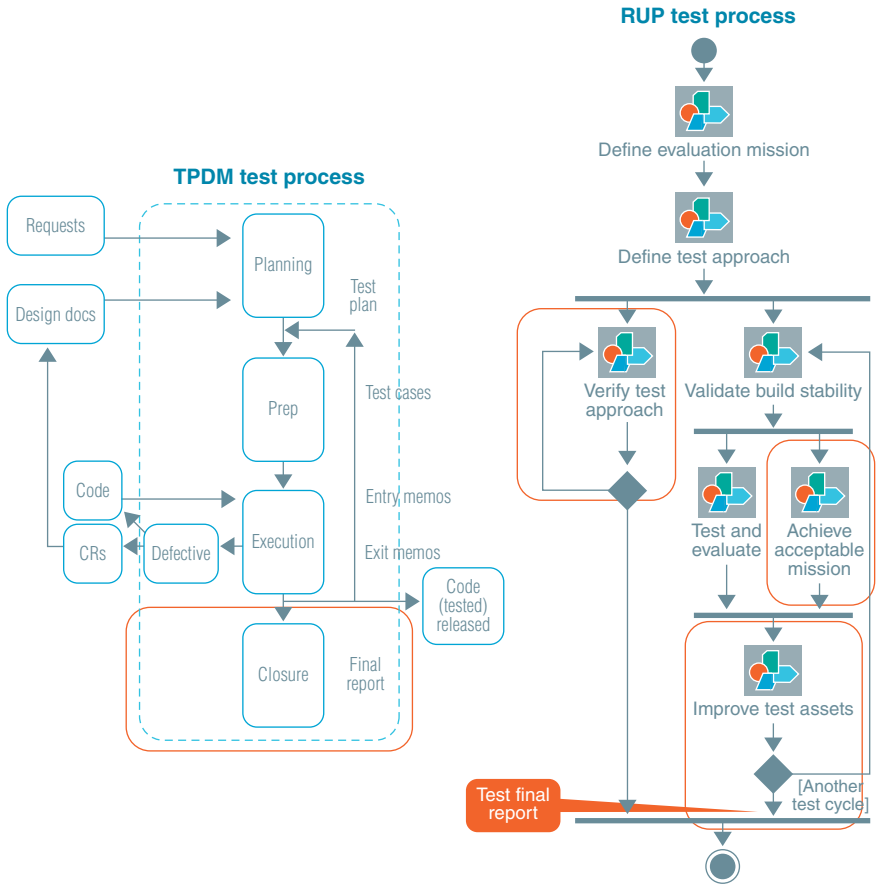

Figure 15d: Mapping TPDM to RUP for Test—closure

As seen in figure 15d, in the final phase of TPDM, the team verified that the test approach was appropriate, improved test assets and produced a test final report.

The team managed and delivered the customizations—including the addition of the test discipline—using IBM Rational Method Composer software. Upon formal review and approval of the new process, the Rome lab published the customized RUP to an internal Web site where it could be accessed by the development team, including testers, in their everyday activities. The Web site is also available to other development teams throughout the organization that may be considering the adoption of RUP.

Use-case-based testing and automation

Building on the best practices implemented in the pilot project—including iterative development as well model-driven architecture and development—the Rome lab focused on two testing best practices for this RUP customization: use-case-based testing (UCBT) and automation.

In functional testing, UCBT ensures validation of what the system is *supposed* to do. When used as a part of system testing, UCBT provides a way to perform end-to-end testing according to use-case flows defined by the business analyst; explore behavior that flows through multiple use cases; and explore integrations with external systems.

In addition, UCBT enables traceability of use cases, enabling testers to identify the impact to test cases when a use-case requirement changes, and produce test coverage metrics. UCBT also helps the Rome lab manage complexity in large systems by decomposing the problem into major functions, and it enables better prioritization of the testing effort. By writing test cases from use cases, the Rome lab encouraged collaboration between testers and developers, leading to a thorough review of use cases. Further, UCBT helped the Rome lab identify more high-level design defects during the first project phases (inception and elaboration), resulting in fewer design defects discovered during formal test phases (functional and system test).

The Rome lab sees improved automation as a way to quickly detect destabilizing changes on each new product build. More importantly, automation would enable the lab to reduce regression testing time in all test phases, and reduce the number of defects in the released product. UCBT and automation complement each other because automation test cases can be built from the design use case to cover, at a minimum, basic flows.

Choosing the pilot project

For the pilot project of RUP for test, the Rome lab selected IBM Tivoli Workload Scheduler, Version 8.3 software, a workload management solution that automates and controls cross-platform scheduling. The decision to use this project as a pilot was based on several criteria. First, the project schedule had already been created following the best practice of iterative development, which simplified the task of implementing an iterative test discipline. Also, the development team was already using the customized RUP process integrated with TPDM for the analysis and design discipline and the implementation discipline. This enabled the project team to use the test discipline to iteratively monitor and validate the earlier RUP customization implementation and to assess its ability to meet objectives.

Second, the size and scope of the project enabled the Rome lab to try the process on a larger project, one involving more than 40 developers and 30 testers.

Implementation

In implementing RUP for test on the Tivoli Workload Scheduler project, the Rome lab wanted to introduce the test discipline gradually and iteratively. This approach was taken to avoid a major disruption that can accompany a process change on a large team. Also, because the project timeline had already been defined when it was selected as the pilot, the Rome lab needed an iterative approach that could be introduced starting in the second project iteration during the construction phase, as shown in figure 16.

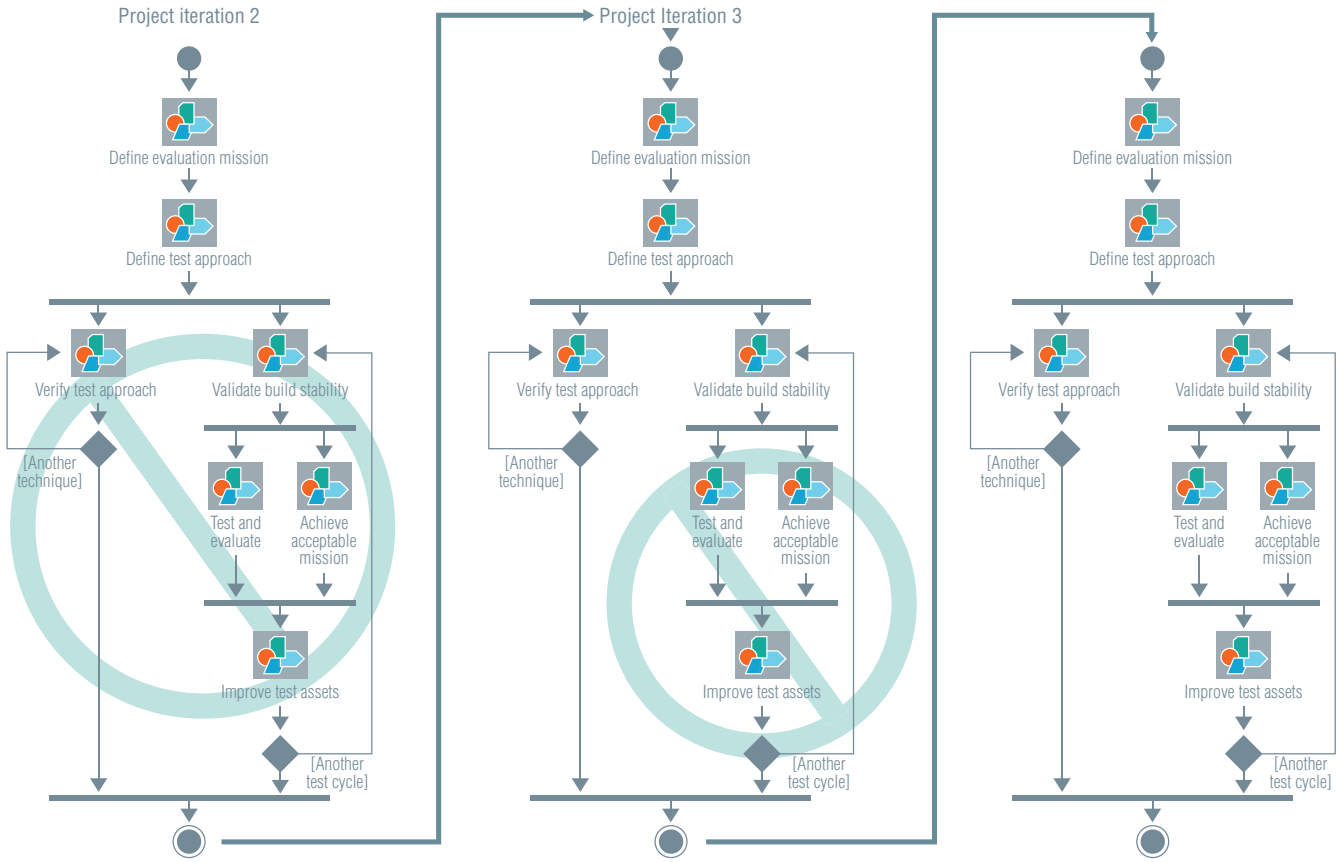

Figure 16: RUP for test was implemented in multiple steps.

The customized workflow for the test discipline, shown in figure 16, is repeated multiple times.

The discipline was introduced for the first time in the second construction iteration. At this point, a relatively small group of three testers began working on the test planning activities as described in the *Define evaluation mission* workflow. The group also worked on a draft version of the test case/test design document described in the *Define test approach* workflow detail, leveraging use-case specifications as an input to enable UCBT. Third, this group of testers began work on the Automation Architecture artifact described in the *Define test approach* workflow.

In the third construction iteration, the development team conducted the first tests on available product builds, including unit testing and initial performance testing. In this iteration the team expanded to five testers, who continued to develop the test case/test design document for all use cases involved in the current iteration and in subsequent iterations. This group also reviewed and approved the subset of test case/test design documents required for test execution in the current iteration.

In this iteration, the testers also started testing product builds as described in the *Verify build stability* workflow, and began building the automation test suite for the currently available use cases as described in the *Define test approach* and *Verify test approach* workflows.

In the fourth and all subsequent construction iterations, the entire RUP for test discipline was being implemented by a team of approximately 20 testers. This team tested and evaluated new product functionality and product performance as described in the *Test and evaluate* workflow. It also added tests to expand the automation test suite, and maintained and improved test assets as described in the *Improve test assets* workflow. Lastly, the team monitored test execution, as described in the *Achieve acceptable mission* workflow.

A time line of the entire process is shown in figure 17.

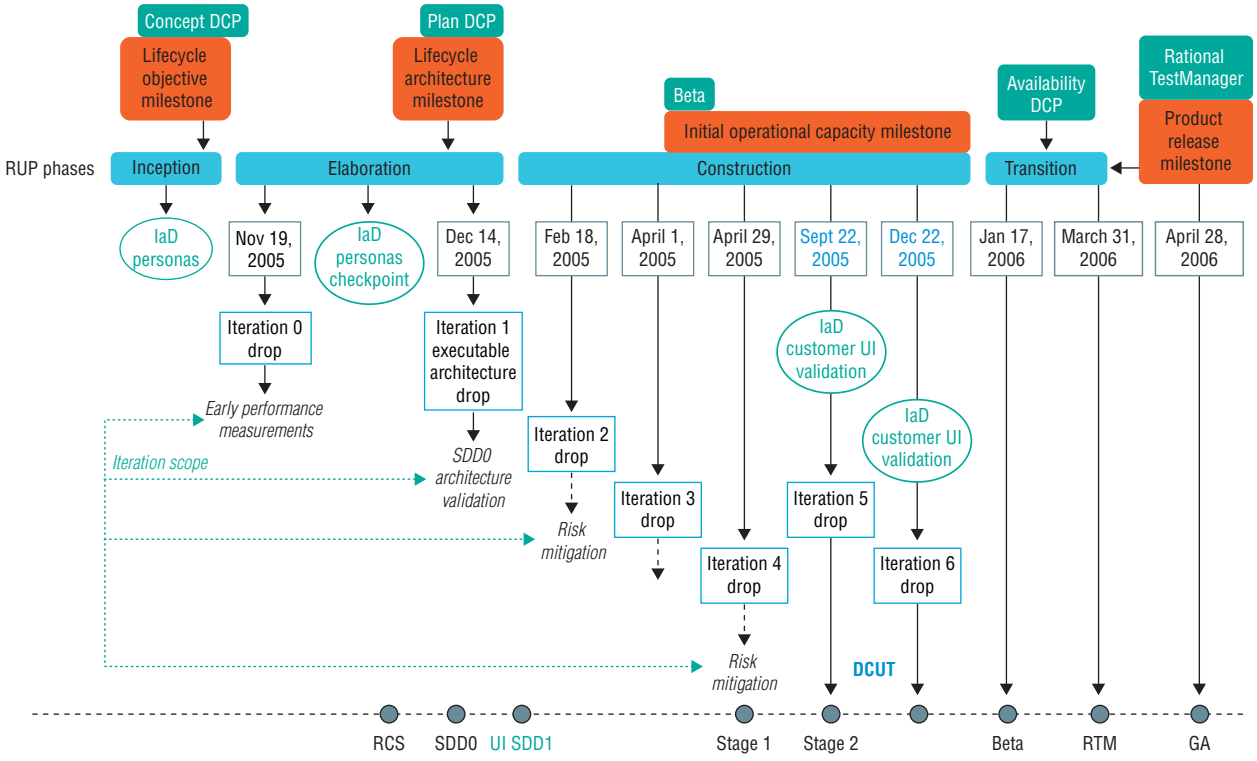

Figure 17: A time line of RUP implementation for Tivoli Workload Scheduler, Version 8.3 software

Results

While the initial pilot project for RUP was placed on hold during implementation, the RUP for test pilot project was completed, enabling the Rome lab to fully assess the quality and productivity gains of RUP adoption.

Leveraging ODC

The Rome lab's ability to quantify quality improvements was enabled not only by carrying the project out to completion, but also by the group's previous adoption of another industry best practice, Orthogonal Defect Classification (ODC).

ODC provides a method for defining and capturing attributes of software defects that facilitates subsequent mathematical analysis. Because the Rome lab used ODC to classify defects on earlier development projects, the team was able to accurately quantify the improvements afforded by the customized RUP methodology and supporting IBM Rational Software Delivery Platform solutions.

Productivity improvements

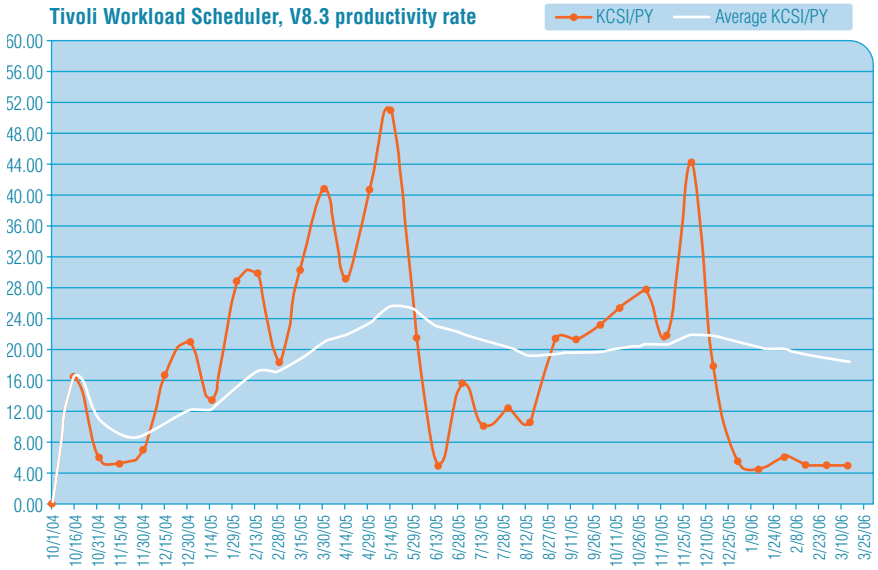

Figure 18: Productivity rates on the RUP for test pilot project

On the Tivoli Workload Scheduler project, the team measured 331,400 changed source instructions, or a KCSI of 331.4. Productivity for development phases (i.e., not test activities) was measured at an average of 18.7 KCSI per person year (KCSI/PY) with a peak of 51, as shown in figure 18.

This result was compared against two baseline projects. The first was an earlier minor release of the same product line, which had a KCSI/PY of 9.5. The second project, similar in size and scope, had a KCSI/PY of 14.4. In both cases, RUP contributed to a significant increase—30 percent or more—in development productivity.

Test productivity increased as well, with the Rome lab measuring an increase of 20 percent on the RUP for test pilot project. The productivity increase was attributed to UCBT and improved automation, a combination that also increased overall functional test coverage by 30 percent.

During component verification testing (CVT), the testing team effectively implemented its test strategy, and highlighted a high percentage of the ODC triggers coverage and variation as shown in figure 19a.

Figure 19a: Planned and actual signatures of CVT

Because of increased code stability and the ability to base test cases on use cases, the team was able to improve the granularity of its tests, resulting in very few blocked test points during execution, as shown in figure 19b.

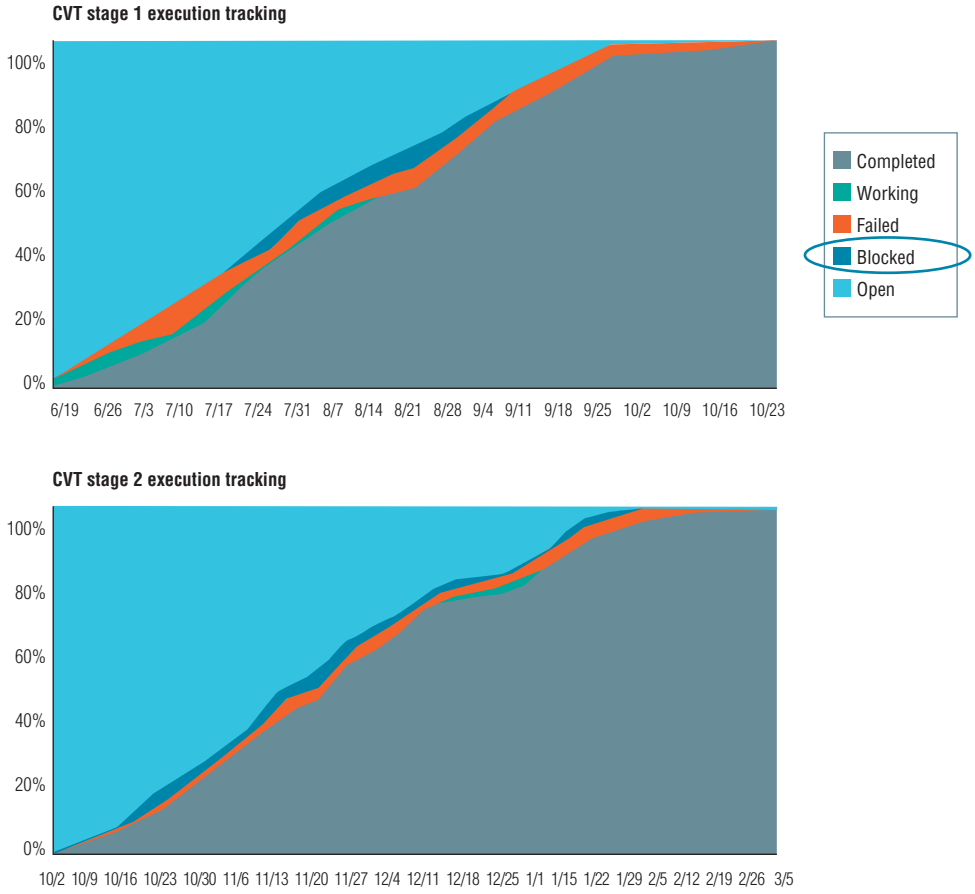

Figure 19b: More granularity resulted in a low percentage of blocked test points.

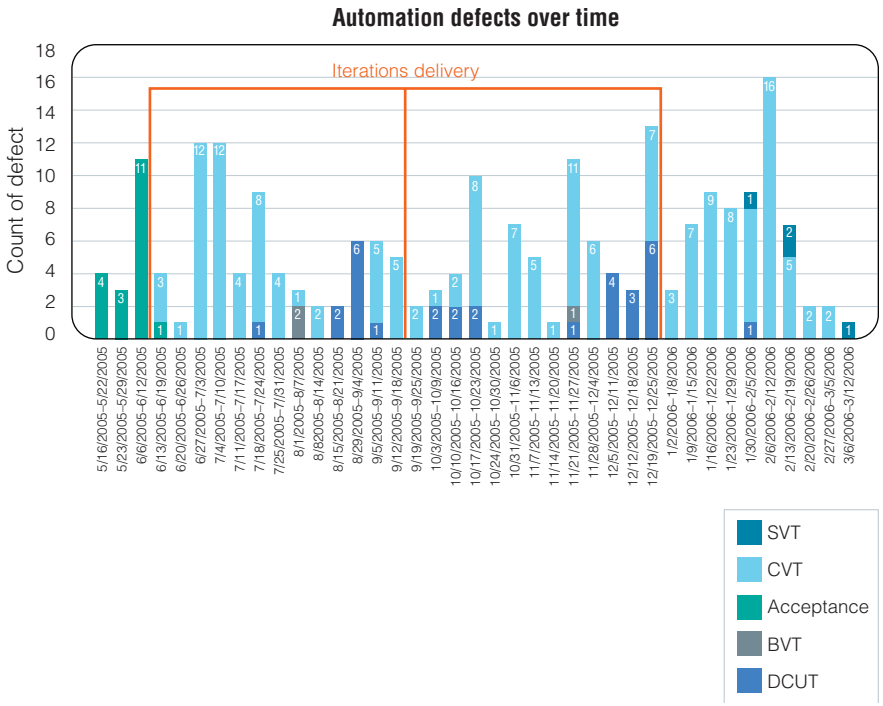

Figure 20: Defects found by test automation throughout the project

Test automation of 12,000 points covered 19 percent of CVT—including 100 percent of API and Web services testing—on the Tivoli Workload Scheduler project, compared to 0 percent on the previous release of the same product. Further, test automation uncovered 222 defects, including 53 in the *early* phases of design, code and unit testing (DCUT) and acceptance testing, as shown in figure 20.

Quality improvements

The Rome lab also quantified quality improvements, again by comparing the Tivoli Workload Scheduler, Version 8.3 project against a baseline project of similar scope. The lab found that the team had removed defects much earlier in the development process when following RUP best practices, including UCBT and use-case reviews. Critical problems were found in requirements and high-level designs much earlier, resulting in a 50 percent decrease of high-level design and requirements defects found in formal test phases.

Figure 21: The Rome lab found and removed defects earlier in the development process.

The combination of UCBT and automation using IBM Rational Functional Tester software, JUnit and the open source Software Testing Automation Framework (STAF) also improved the test effectiveness, enabling the lab to find 10 percent of all functional defects through automated tests. In later project iterations, 32 injected defects were found by automated test suite execution during regression testing. Less than 10 percent of overall defects were found during system testing on the RUP project, compared to more than 30 percent on the baseline project.

Calculating ROI

In the end, the decision to go forward with the adoption of a new software development process will usually be based on the simple question, "*How much will we save by doing this?*"

Although the Rome lab could not accurately predict cost savings before its pilot projects, the group was able to calculate ROI based on precise measurements it had taken during the Tivoli Workload Scheduler development effort. Again, these measurements were compared to a similarly sized project. The benefits analysis incorporated savings resulting from productivity improvements in development and testing, as well as savings due to earlier discovery of requirements and high-level design defects. The cost analysis included training, mentoring and skills ramp-up, as well as product installation and configuration, but it did not include the cost of the IBM Rational Software Delivery Platform solutions.

Activity	Saving (K$)
Productivity improvement development	533
Productivity improvement test	125
Quality (high-level design and requirements defects reduction)	183 (pre-GA)
Total	**841**

Activity	Saving (K$)
Training	27
Skills ramp-up	217
Cost of the Rational tools software licenses	0
Mentoring	25
Total	**269**

Figure 22: Cost and benefit analysis of the adoption of RUP at the Rome lab

Figure 22 shows the results of a thorough analysis of these costs and benefits, based on detailed calculations of the value of productivity and quality improvements, as well as time costs associated with learning the new process.

The ROI calculated for the effort is:

$$ROI = (841 - 269) \div 269 = 196\%$$

Extending the same exercise and assuming that the project team will be involved in multiple releases, the Rome lab estimates the cumulative ROI will increase to 373 percent, 491 percent and 576 percent for the second, third and fourth releases, respectively, as shown in figure 23.

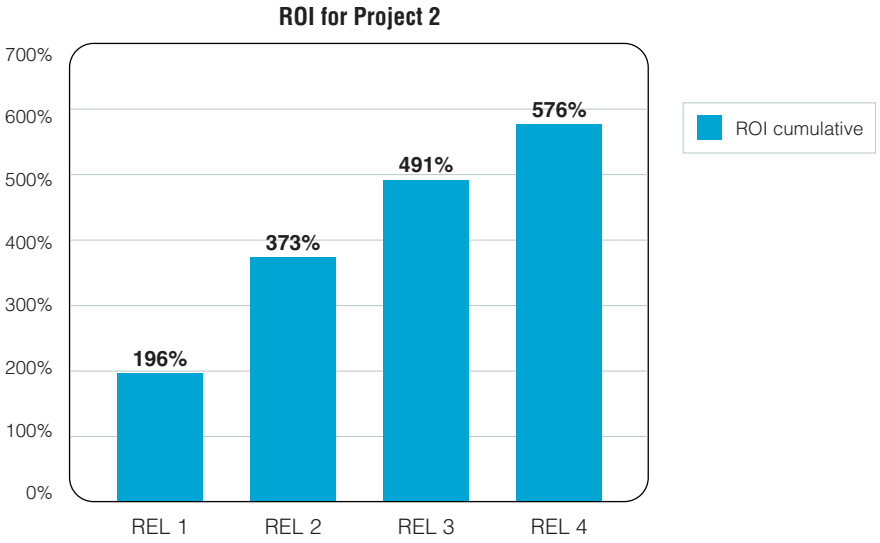

Figure 23: Estimated ROI for future projects

Summary

RUP has yielded a significant return on the Rome lab's investment, while complementing the group's established software development practices.

For more information

To learn more about how the IBM Rational Unified Process methodology can improve your business operations, contact your IBM representative or visit:

ibm.com/software/rational

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
06-07

RAW11013-USEN-00