

Organizing RUP SE projects

by [Murray Cantor](#)

Principal Consultant
Rational Software
IBM Software Group

IBM RUP SE[®] is an extension of IBM Rational Unified Process[®] or RUP,[®] for addressing systems development. RUP SE can be applied not only to software development and integration projects, but also to projects that include hardware development or acquisition and require specification of worker roles. This article provides a brief overview of how to extend generic RUP project management principles to RUP SE projects.



As a RUP extension, IBM RUP SE specifies that projects adhere to certain fundamental principles:

- The project lifecycle model includes the four RUP phases-- Inception, Elaboration, Construction, and Transition -- and the RUP disciplines: business modeling; requirements; analysis and design; implementation; test and assessment; deployment; configuration and change management; project management; and environment. In fact, the familiar RUP diagram (see Figure 1) applies unchanged to RUP SE.
- Project activities are not serialized; instead, teams evolve artifacts -- including the project plan -- in parallel, and detail them as their understanding of the project problem and solution increases.
- The system is developed through a series of iterations, each satisfying more of the functional requirements. The focus is on finding and eliminating problems early, thereby reducing risk.

[▶ subscribe](#)[▶ contact us](#)[▶ submit an article](#)[▶ rational.com](#)[▶ issue contents](#)[▶ archives](#)[▶ mission statement](#)[▶ editorial staff](#)

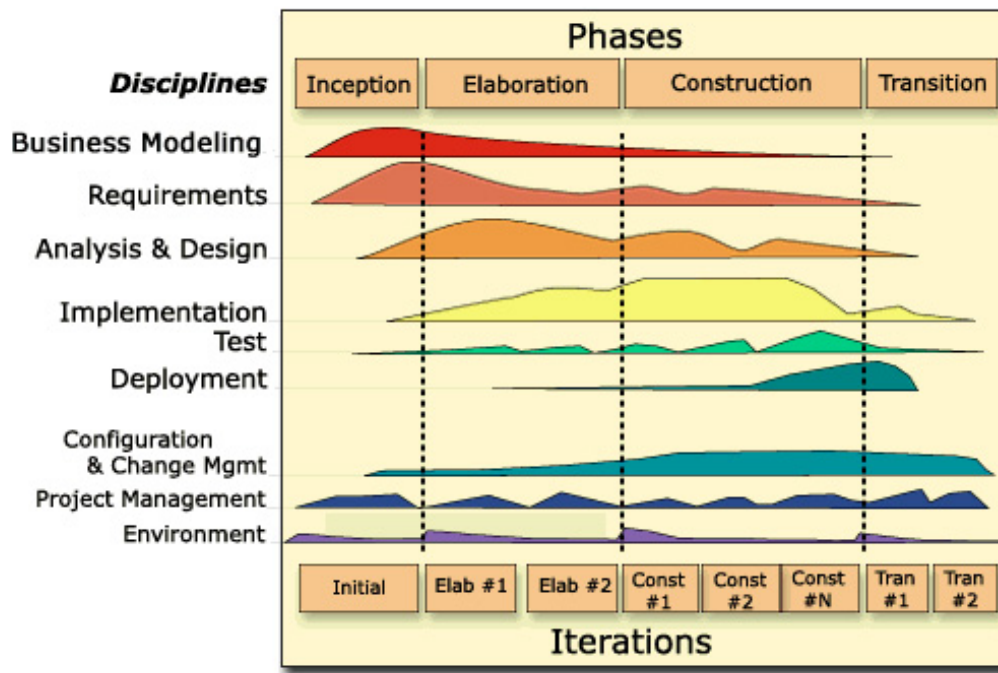


Figure 1: The Rational Unified Process lifecycle phases focus on risk reduction

Applying these principles to systems development raises two major issues:

- **Team structure.** How do we partition the staff into development teams and assign staff roles and responsibilities?
- **Iteration planning.** For large IBM RUP SE projects, how do we coordinate multiple development teams? When hardware is involved, how do we apply iterations to the hardware development?

This article addresses these issues, by showing how to apply RUP principles to RUP SE projects. It starts with a general discussion of the concepts underlying development project organization: basing the team organization around the project architecture, and the role of requirements analysis in partitioning effort. A brief overview of the RUP SE architecture framework follows that discussion.

Note that this article assumes an understanding of the fundamentals of IBM RUP SE:

- The RUP SE architecture framework [1](#)
- UML (logical) subsystems [2](#)
- Localities
- Requirements flowdown

For a more complete description of these concepts, see the IBM Rational whitepaper, "Rational Unified Process® for System Engineering, RUP SE 1.2," [3](#) and the RUP SE Rational Unified Process Plug-In. [4](#)

Partitioning strategies

One lesson learned from software development that carries over into system development is that **there is a diseconomy of scale related to effort**. As a project grows, more and more effort is spent on communication among the developers. As Fred Brooks⁵ points out, the number of conversations required has the potential to grow quadratically with the number of team members, and this growth can actually occur in poorly organized projects. Even in well-organized projects, the effort can grow to the 1.2 power with the number of staff.⁶ Hence *individual productivity falls off as the size of the effort increases*. **A fundamental management task is to partition the effort so as to manage communication among the developers**. One strategy is to partition the effort into teams and then minimize communication among those teams. This divide-and-conquer strategy can prevent a quadratic loss of individual productivity.

There are two approaches for partitioning the effort:

- **Partitioning by requirements.** Partition the system requirements into sets, and then assign teams to take responsibility for developing the parts of the system that deliver each of those sets.
- **Partitioning by architecture.** Assign teams to develop subsystems or subcomponents of the system. The requirements for these architectural elements are derived from the role they play in meeting the system requirements.

Let's explore each of these approaches.

Using requirements to partition effort

The requirements approach is common in systems development. *The perceived advantage of partitioning by requirements is that it sharply decouples the effort, which simplifies the management problem*. Different teams build different components that meet different requirements. These teams do their work in relative isolation, and, at the end of the project, bring their component to be integrated into the system.

This means of decoupling the development seems very attractive on the surface: There is little need for the teams to communicate. The belief is that each team can enhance productivity by using its own processes. In the end, what was originally a big, unmanageable project becomes a set of smaller, manageable programs.

However, **this apparent simplicity often comes at a price**. First, note that decomposing the system by requirements does, in the end, impose architecture: The components built in isolation become an implicit architecture. However, since *this architecture was not explicitly constructed to address such quality issues as extensibility and maintainability, the resulting system tends to suffer from high service costs and may be incapable of evolving to meet changing mission needs*.

A second problem with the requirements approach is that *complications can arise when the team efforts are coupled at integration*. Even though the components meet different requirements, they often need to communicate. Interface documents specify how the components access each other's services, and *before they can proceed with their work in isolation, each team needs a fully documented, frozen interface document*. For a new system, it is unlikely that such a document will be available early on, since such interface specifications result from a detailed design effort -- and that will not yet have occurred. So each team needs something it cannot have. In practice, this leads to premature attempts to freeze the interfaces in an Interface Control Document (ICD) and then interminable interface control meetings, as the shortfalls of the ICD become evident. Each team may go into an ICD meeting more focused on preserving its schedule than on finding a good technical solution.

Finally, a third problem is that *modern systems require more integration than did past systems*. This places more attention on internal reuse of hardware and software components, and on the ability to redeploy the components as you add more capability to the system. With all of these new demands, the architecture must be managed more closely than is possible when the effort is decomposed along requirements.

Using architecture to partition effort

This second strategy assumes that the system is partitioned into parts. In other words, it assumes an architecture whose elements were chosen to achieve an optimal design while balancing stakeholder needs. Teams are assigned to build each grouping of parts.

As we noted, the requirements method implies an architecture, but not a good one. The implied architecture does not account for many factors that need to be considered in specifying a quality architecture: maintainability, extensibility, supportability, overall responsiveness, and cost of ownership. *Partitioning the effort along such a quality architecture results in some managerial complexity, but the extra effort results in a superior system.*

In the following sections we will address the following challenges that result from allocating along architecture lines:

- Project management needs to understand the architecture to staff the team accurately.
- Since optimal architectures are more coupled than the implicit functional-allocation architectures resulting from requirements-based project staffing strategies, optimal architectures require more team communication.
- The architecture needs to evolve along with the team's understanding of the system design and implementation. Some team members must be responsible for maintaining the integrity of the architecture throughout development.
- The requirements that each team manages are *derived* from system requirements, not from an *allocation* of system requirements. We

will discuss the distinction between derived and allocated requirements below.

As we shall see below, in the end, each of these challenges sets the stage for optimizing project performance.

RUP SE architecture and requirements

As mentioned earlier, this article assumes some understanding of RUP SE artifacts and workflows. This section highlights RUP SE concepts pertinent to our discussion: derived requirements; maintaining traceability between system requirements and model element requirements; and separation of concerns between the logical, physical, and information aspects of the system architecture.

Probably the most important notion is that of **derived requirements**. For reasons alluded to above, generating requirements for analysis elements and their implementations (subsystems, localities, etc.) is not merely a matter of sorting the system requirements. Instead, these are newly derived requirements based on the roles and responsibilities of the elements and how they work together to meet the system requirements. The workflow for deriving the analysis model requirements is detailed in the RUP SE whitepaper and RUP SE Plug-In mentioned earlier.⁷

One of the RUP SE disciplines is to **maintain traceability between the system requirements and the analysis model element requirements**. This traceability is usually not a decomposition tree, as found in functionally decomposed architectures; it is a many-to-many mapping. As discussed below, maintaining the traceability between system and analysis model requirements is important for iteration planning.

Another important aspect of the RUP SE framework is that, **at the analysis level, the decomposition is not hardware and software, but rather logical and physical, based on separation of concerns**. *Given the flexibility of modern technology hardware/software choices for product development (e.g., whether the logical capability is realized in VHDL, in an ASIC, in firmware driving a DSP, or code running in a CPU) are typically price/performance trade-offs that may change over the commercial life of the product.* In fact, at a given point in time, a single analysis model may be realized in a variety of products, with different choices as to how to physically provide the logical capability, meeting different price/performance points. Further, given modern embedded system development environments, the distinction between hardware and software is somewhat blurred.

Also note that the software and hardware cannot be developed in isolation. The structure of physical software components (executables, etc.) is based on the physical architecture. The workflow for determining software components based on the logical and physical architecture is provided in the same RUP SE whitepaper and RUP SE plug-in mentioned earlier.⁸

RUP SE project organization

In setting the organization, it is useful to keep some fundamental principles in mind. Many of these principles apply to any large RUP-based development effort:

- Cover all of the RUP core workflows.
- Balance communications by assigning developers who need to communicate frequently to the same team whenever practical.
- Maintain architectural coherence through cross-team membership.
- Plan and deploy resources so that all teams function throughout the development lifecycle.
- Ensure that implementation includes integration of the separate components throughout the lifecycle.

Note that the last two principles follow from using RUP's iterative strategy. As mentioned above, the system development activities (requirements analysis, architecture, design, implementation, integration, and test) are not serialized in RUP projects; instead, they are carried out in parallel as the team delivers the system as a series of iterations with increasing capability.

We apply these principles at the system level, using associated RUP SE artifacts as the basis for defining team deliverables. As we shall see below, *the analysis model serves as the architectural basis for organizing the effort.*

Team structure

The RUP SE project organization consists of the following teams with overlapping membership.

- **Enterprise/business/mission modeling team.** Develops enterprise business/context models in order to set system context and derive system requirements.
- **System analysis team.** Builds the analysis model, including the UML subsystem and localities. This team also develops and maintains the derived requirements to the analysis, and it may build the analysis-level process and data models.
- **Design and implementation teams.** Responsible for the detailed design and implementation of components within a given viewpoint.
 - **Subsystem teams.** Develop detailed class design and associated software modules for one or more subsystems.
 - **Locality teams.** Develop detailed hardware specifications, design, and hardware components for one or more localities.
 - **Other teams.** Might include **data modeling and computer/human interaction.**

- **Build and integration team.** Receives components developed by the development and implementation teams and builds system iterations.
- **System test team.** Plans, executes, and reports on system tests.
- **Operation and maintenance team.** Performs field delivery, tracks problems, prioritizes change requests, and delivers updates and patches.
- **Project management team.** Performs ongoing iteration planning, context management, and stakeholder communications.

Figure 2 shows the relationships among the various teams, with the arrows indicating communication paths between teams.

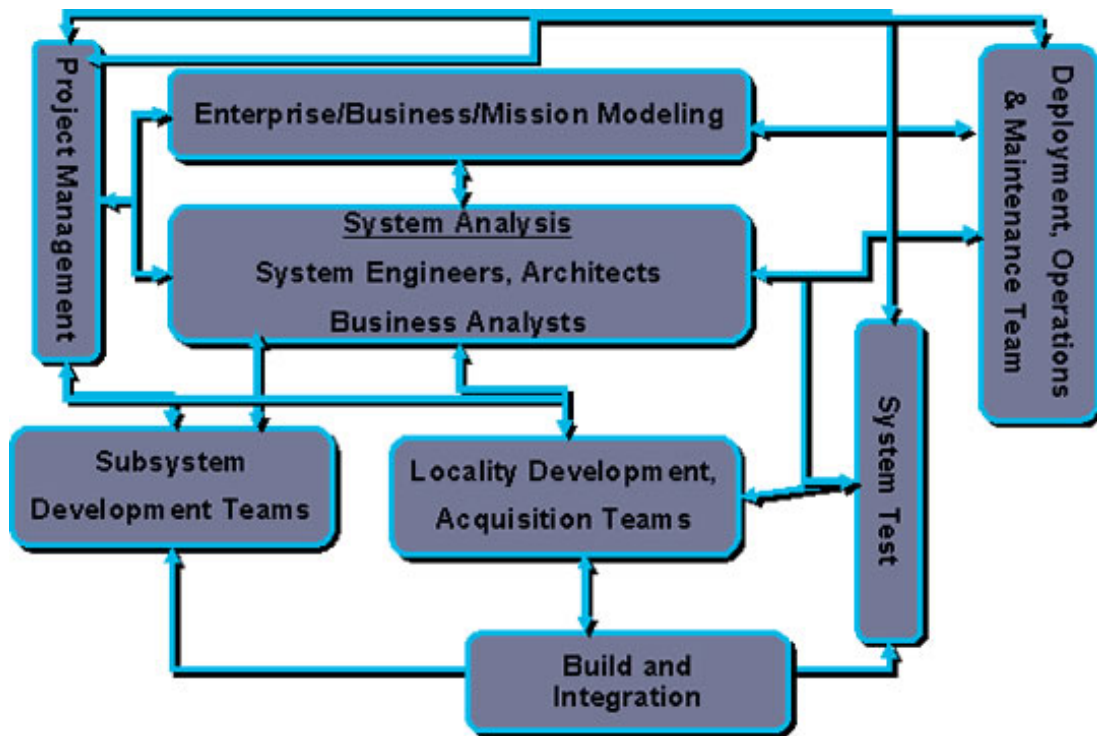


Figure 2: RUP SE project team organization

Let's explore in detail what each of these teams does.

Enterprise/business/mission modeling team

This team is responsible for establishing system requirements by defining the role that the system under development will play in the overall enterprise. This team applies flow methods to create system use cases, system services, and system supplemental requirements. Over the course of development, this team may update the enterprise model as the system specification evolves. In addition, the team serves as an authoritative source for dealing with requirement issues that emerge during development.

Ideally, this team should consist of business analysts and/or domain or process experts, along with some or all members of the system analysis team.

System analysis team

This team builds and maintains the system analysis model, including each of the views. In addition, this team carries out joint flowdown activities to derive the system use-case survey and services, survey of hosted services for each locality, and so on. This team looks after the integrity of the architecture, resolves interface issues, and addresses discovered needs for changes as the development evolves.

This team includes the lead architect(s) and some of the business analysts or domain experts. Over time, the lead architects of the design and development team, the build and integration team, and the test team will join the system analysis team.

Design and implementation teams

These teams create the modules that are integrated by the build and integration team. Once the architecture is reasonably stable, the elements of the analysis model are partitioned to the design and development teams. In particular, some teams (subsystem teams) are given one or more UML subsystem(s) to design and develop in code modules. Other teams (locality teams) are assigned localities to design; then they either develop these localities or acquire hardware modules for them.

The teams are organized around core competencies. Good logical and physical decompositions are generally aligned with technical competencies.

Subsystem teams. Note that *the derivation of requirements for subsystems results in a clean set of specifications (requirements and interfaces) that permits subcontracting of work if necessary.* Subcontracting is frequently necessary for large systems projects, and IBM RUP SE allows for independent development that maintains synchronization with other development plans. In many instances the subsystems are software based, but in others the work of UML subsystem development teams could be instantiated as firmware (e.g., EPROM-hosted code for an embedded controller) or hardwired into an ASIC. In still other cases, these teams may work with VHDL⁹ designers for a full hardware implementation.

In the case of UML subsystems, teams develop the subsystem context diagrams, and the detailed class design and code modules. They need to maintain coordination with the locality teams to ensure the code modules are suitable for compilation and deployment on the target hardware platforms. The subsystem teams conduct unit tests.

Locality teams. The locality teams take into account the derived supplemental requirements and surveys of hosted services to determine the needed hardware resources for deployment of the logical services. This team will then develop the hardware specifications and look after the

delivery of the needed components.

Build and integration team

The development teams develop modules for integration, and the build and integration team assembles them to create versions of the system. Note that IBM RUP SE employs iterative builds to manage content and technical risk. The build and integration team is responsible for the ongoing system integration effort and includes build environment experts. Each of the development teams has liaisons who are responsible for coordination with this team.

This team develops and maintains the make files, and -- working with the configuration management system -- creates the labeled software builds.

If hardware is not available, the development teams may need a simulated or scaffolded hardware platform to build interim system iterations. The build and integration team, working with the locality development teams, are responsible for providing those resources.

System test team

This team plans, executes, and reports on the system tests for each of the iterations.

Operation and maintenance team

This team tracks field experience, prioritizes change requests, and performs other similar activities.

Project management team

A key project management tool is the evolving iteration plan. As we will discuss below, IBM RUP SE iteration planning consists not only of creating and maintaining a system-level iteration plan, but also of deriving iteration plans for each of the teams. The project management team, consisting of the system development project manager, the lead architects/engineers, and the development, integration, and test team leads, is responsible for iteration plans and standard project management tasks such as staffing and status collection and reporting.

Staffing curve

RUP projects are typically not fully staffed at their onset. For larger projects, there is not enough work to keep everyone busy during the Inception phase; a core group of key staff should carry out Inception activities. Once they meet the Inception lifecycle goals, then managers should have enough confidence about the scope and required effort to fully staff the project.

The same is even more true for RUP SE projects. Note that creating the analysis model is primarily an Elaboration activity, so in principle, the

development teams should not be fully staffed until the end of Elaboration. In practice, however, managers can typically identify core competencies among staff members at the beginning of a project, so the Inception team can include development team leaders. As their understanding of the project deepens, managers can better estimate the optimum size for these teams and begin staffing up at the beginning of the Elaboration phase. By the end of Elaboration the teams should be fully staffed.

Key roles

Note that each team works throughout the entire project. Hence, each system team (project management, enterprise modeling, analysis, test, and build and integration) needs a lead responsible for maintaining the system perspective, delivering results, and coordinating with development team representatives.

Iteration planning

A key feature of the RUP lifecycle is that the project team builds the product through a series of iterations of increasing capability. At a system level, the standard iteration principle applies: *Iteration content is described by an increasing set of use-case scenarios. **Early iterations address technical risk; later iterations address content risk.*** Since there is a large body of literature describing the advantages of iterative development and ongoing iteration planning, we will not describe those benefits here.

In addition to the standard iteration planning concerns, system development involves other critical issues:

- Each development team needs its own derived iteration plan.
- Hardware delivery dates may not support iteration delivery.

The notion of derived iteration planning is not new, and with every RUP product, IBM Rational delivers a whitepaper by Maria Ericsson that discusses the principles of derived iteration planning. In the following section, we will see how to apply those principles to RUP SE projects.

Derived iteration plans

Recall that an iteration plan consists of specifying a sequence of partially functional versions of the system, which culminates in a completely functional system. So each iteration requires a set of test cases to verify the functionality and the hardware and software components required to provide that functionality. Each development team needs the means for determining what logical and physical modules it must develop to support each iteration, and the integration team needs to know what pieces to expect from the development teams for integration. The specification of these pieces for each iteration, based on the system iteration plan, results in a derived iteration plan for each of the teams.

The IBM RUP SE requirements flowdown workflow provides the means for deriving these team iteration plans. Recall that the use-case flowdown workflow produces surveys:

- **Derived use-case surveys** are produced for each of the UML systems, along with UML subsystem services that enable the use cases.
- A **survey of hosted subsystem services** is produced for each locality.

Each element of these surveys traces from one or more system use-case scenarios. Following the traceability, the subsystem iteration plan is derived from the system iteration plan by including the subsystem use-case scenarios traced from the included system use-case scenarios. Subsystem services define the interface requirements for a subsystem, and they must also be supported by the hardware realization of the localities. Following this traceability path results in the locality iteration plans. The integration of the modules that make up the realizations of the development team iterations must be included in the build and integration iteration plan so that each team's iteration plans may be derived from the system iteration plan.

If hardware is not yet available to pursue a locality team's derived iteration plan, project management is faced with an important decision: Is the benefit of risk reduction worth the extra investment in a simulated hardware platform? The derived iteration plan provides the necessary information to scope the simulator capabilities requires and conduct a cost/benefit analysis. Often the analysis points toward doing a simulation, in which case, the derived iteration plan provides the necessary requirements for it.

Resource balancing and smoothing

One challenge of managing a large-scale iterative development is to ensure that the effort required from each team to support an iteration is reasonable. Each of the teams' project managers needs to ensure that the team resources are adequate to support their derived iteration plans. In addition, managers need to ensure that staff is not idle during iterations that place only a "light" demand on their team. For example, it is generally not possible for a major subsystem to become fully functional in the first iteration; only some of the subsystem's functionality can be delivered that early on. The system project manager and subsystem lead typically negotiate and revise the system iteration plan to reflect what each of the subsystem teams can actually deliver within the iteration schedule. This activity, a standard project management responsibility, is the often called *resource smoothing*.

Resource smoothing is the responsibility of the project management team. They achieve the balancing by jointly performing the following workflow:

1. Set the system iteration plan using the usual RUP principles of addressing technical and content risk.

2. Use requirements traceability to set the derived iteration plans.
3. Assess team resources for the derived iteration plan and propose an achievable alternative plan (performed by each project manager for each team).
4. Unify alternative proposals by shifting functionality in the system iteration plan, and occasionally planning to shift resources between teams.

As in standard RUP projects, RUP SE project iteration plans are never frozen; they continually evolve as managers examine the results of the delivered iterations. Maintaining the system plan and derived iteration plans is an ongoing responsibility of the project management team.

A final word: How hard is this?

As we mentioned above, using architecture rather than requirements to organize systems development may, at first, seem overly complex and difficult to put into practice. However, the complexity of the organization scales with the complexity and size of the effort. For smaller, simpler projects, you can combine roles and teams so that the organization is more streamlined. Even for larger teams, the difficulty is more perceived than real. For these projects, with their inherent diseconomies of scale, the RUP SE management approach tames the nonlinear growth in interteam communications, yielding productivity increases. For very large projects, applying RUP SE at several levels simplifies the management process, and synchronizing the plans from these various levels validates overall system planning. At every level, the teams have clear roles, areas of concern, and responsibility. In the end, process adoption can proceed smoothly, allowing all teams to focus on the system, and not on the process itself.

Notes

¹ The RUP SE architecture framework can be found in the RUP SE extension to the RUP available on RDN. This framework is further explained in the draft whitepaper "The Rational Unified Process for System Engineering 2.0" (in press) by Murray Cantor, and available from the author.

² RUP SE relies on a way to express logical decomposition. UML 1.4 characterizes the elements of the logical decomposition of the system as subsystems. At this writing, the UML 2.0 drafts use different semantics to express the logical elements. When UML 2.0 is adopted, we will change the semantic representation of logical elements to reflect the current standard.

³ Rational Software TP 165, April, 2000, (<http://www.rational.com/products/whitepapers/wprupsedeployment.jsp>). A new whitepaper on RUP SE 2.0 is currently in production.

⁴ Available through Rational Developer Network (<http://www.rational.net>); authorization required.

⁵ Fred Brooks, *The Mythical Man Month*, Addison Wesley, 1997.

⁶ Barry Boehm et al., *Software Cost Estimation with COCOMO II*, Prentice Hall, 2000.

⁷ See notes 3 and 4.

⁸ See notes 3 and 4.

⁹ Very High Speed Integrated Circuit (VHSIC) Hardware Description Language



For more information on the products or services discussed in this article, please click [here](#) and follow the instructions provided. Thank you!

Copyright [Rational Software](#) 2003 | [Privacy/Legal Information](#)