Unifying Systems and Software Teams: A Holistic Approach to Systems Development

Dave West
Group Manager
IBM Rational Software

Robert A. Maksimchuk
Industrial Solutions Market Manager
IBM Rational Software

Rational software

A whitepaper from IBM Rational

Contents

This whitepaper describes the IBM Rational Software overall approach to systems development, covering both important disciplines that must be undertaken on a systems development project and the specific solution offered by IBM Rational Software. It highlights the motivations for following this approach by describing the principal problems it solves.

The approach to systems development described here draws on the work by Murray Cantor[1] regarding Systems Engineering using the IBM® Rational Unified Process® and by the Rational field organization in applying these techniques and technologies to solve real world problems. This work includes a wide variety of projects, ranging from defense programs to e-government initiatives. The most salient, common feature of these projects has been their size and complexity. In every case, the focus was on creating not just a software product, but instead a complete system for supporting a business or mission.

The objective of the paper is to describe how a combined, integrated approach to building systems, coupled with a platform to build such systems, enables teams to work in an ever increasingly complex environment.

Introduction

A system is “An integrated set of elements to accomplish a defined objective. These include hardware, software, firmware, people, information, techniques, facilities, services, and other support elements.[2]”

Why should we re-evaluate our approaches and techniques for systems development? Below are some of the industry drivers for doing so.

- **Enterprise Architectures.** In 1996, the US Congress passed the Clinger-Cohen Act, which, among other things, mandated that federal agencies develop and maintain an enterprise information technology (IT) architecture. This act brought focus to a growing concern that government organizations had developed applications that, while partially successful in meeting a narrow set of needs, could not be tied together to meet the needs of a broader set of constituencies. Currently, many of those organizations are engaged in Enterprise Architecture (EA) efforts intended to enable the interoperability of a broad range of systems to accomplish expanding missions; these efforts involve defining those missions clearly and describing how the

[1] See Murray Cantor’s articles in *The Rational Edge*, http://www-106.ibm.com/developerworks/rational/rationaledge/ (Search the archives under the category “systems development”.)
[2] © Copyright 1998 International Council on Systems Engineering; http://www.incose.org

set of interoperating systems will accomplish them.[3] Because of the focus on the whole system, which includes hardware, software, and people, traditional data or function-centric approaches to specifying an enterprise architecture are not effective. Their inadequate support for evolving enterprises becomes even more pronounced after the architecture is implemented in a working system.

- **Increasing software capability.** Increased flexibility in software allows new system capabilities to be easily introduced. Services can be provided by hardware, people, or—at an ever increasing rate—software. On a systems development project, it is crucial that the development team has the right approach and tools that will enable them to make these design trade-offs. This has increased the complexity of the environment that analysts, project managers and systems engineers work in. It has introduced, much earlier in the lifecycle, design decisions on how a system capability is going to be realized.
- **Development lifecycles.** While the introduction of iterative and agile methods for developing systems can create a number of associated pressures, traditional approaches—the waterfall method in particular—do not satisfy the need to deliver 'chunks' of the system incrementally.
- **Project and program failures.** Examples of project failure abound. One particularly significant statistic, according to the Standish Group, in 1995 the US government spent some $81 billion on cancelled software projects. This highlights the fact that projects and programs are failing and at the heart of that failure is their systems development approach. This has lead to organizations re-evaluating their systems development capability and looking to standards such as CMMI and ISO to provide benchmarks of their capability.

What is needed in a systems engineering approach?

"Systems Engineering is an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining **customer needs** and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem:

[3] "Enterprise Architecture Use across the Federal Government can be Improved," General Accounting Office, February 2002, GAO-02-6.

- Operations
- Performance
- Test
- Manufacturing
- Cost & Schedule
- Training & Support
- Disposal

Systems Engineering integrates all the disciplines and specialty groups into a **team effort** forming a structured **development process** that proceeds from concept to production to operation. Systems Engineering considers both the business and the technical needs of all customers with the goal of providing a **quality product that meets the user needs.**[4]"

This definition embodies the following important notions:

- **Customer Needs** - A system engineering approach must encompasses everything that needs to be considered when describing how a system supports an organization's mission or business process.
- **Communication** - A system engineering approach is about and communicating ideas, information, and processes across the team. A systems engineering approach must provide a method of abstraction that allows communication about each system element to all its stakeholders, as well as a mechanism to map the requirements of each business process to that abstraction.
- **Process and project realities** – Because systems are being developed in the "real" world, with pressures of time and cost, ensuring your development approach does not incur massive overhead to projects is very important.
- **Conformance to system specification** – One vital objective in systems development is that the components and elements of your system conform to the system specification and architecture.
- **Team collaboration** – Any approach to systems development must enable a large or small group of people to work together effectively and to work effectively with other such groups in a distributed environment.

[4] © Copyright 1996-1999 International Council on Systems Engineering; http://www.incose.org/whatis.html

Given all these areas to be considered, the case for a "holistic" approach to systems engineering is compelling. By *holistic* we mean an approach that considers the full range of disciplines involved in systems development: requirements, analysis, design, and implementation and that integrates those disciplines into one process. Moreover, this approach must include a mechanism that allows the systems to be tested and the results to be communicated to stakeholders.

Customer Needs - Use Cases and Requirements

The primary responsibility of any system is to fulfill the mission or business goals of the organization. A system must therefore be developed in the context of a set of requirements from different stakeholders, including:

- Users concerned with functionality and performance (e.g. radar operators for a military system).
- Decision makers concerned with the cost of deployment and ownership, including maintenance (e.g., the program management office for a military system).
- Investors concerned with how a given system will better serve their business and provide competitive advantage (e.g., a government department sponsoring a military system).

The requirements from each of these stakeholders must be factored into the systems architecture, and they must be verified in the final design. These will include various non-functional requirements such as:

- **Usability** – How easy is the system to use?
- **Maintainability** – What is the system maintenance cycle? How quickly can a defect be resolved?
- **Extendibility** – How flexible and how "changeable" should it be?
- **Scalability** – How many users? How many locations?
- **Reliability** – How often is the system going to work as planned?
- **Performance** – What response time/throughput/etc. should be expected in different situations?

- **Supportability** – What is needed to maintain the system and in what environment?
- **Cost of manufacture and deployment** – What is the acceptable cost of production and then replication?
- **Operational cost** – How much will it cost to keep the system working?

The answers to these questions become requirements that must be satisfied in the context of system behavior. And this system behavior is described in a set of use cases.

Use cases describe the behavior of a system as it applies to the various users who interact with that system; in use-case terms, these users are called the "actors." With its clear focus on actors, the use case has become the de facto mechanism for describing the functional requirements of computer systems for a number of reasons:

- The use case describes the behavior of the system from the customer's (i.e. actor's) viewpoint. (In fact, the customer may be another external system or system operator.) This approach allows the functional requirements to be described *as requirements only*, thus avoiding the tendency to describe how the system realizes these requirements. Because use cases describe requirements from the customer's perspective, it is easy to review them with the customer.
- Because a use case comprises a set of paths ranging from the normal transaction, when everything works correctly, to a description of the infrequent 'very unhappy' path, it is natural to deliver the system in increments that support these scenarios. This approach lends itself very nicely to iterative development, with the development team delivering chunks of system that support particular scenarios, which make sense to the customer.
- The use case describes an end-to-end set of transactions that are of value to the customer. The key phrase here—"of value"—means that the use case is not an arbitrary partitioning of functionality, but is instead a transaction that provides value to the customer (see Figure 1).

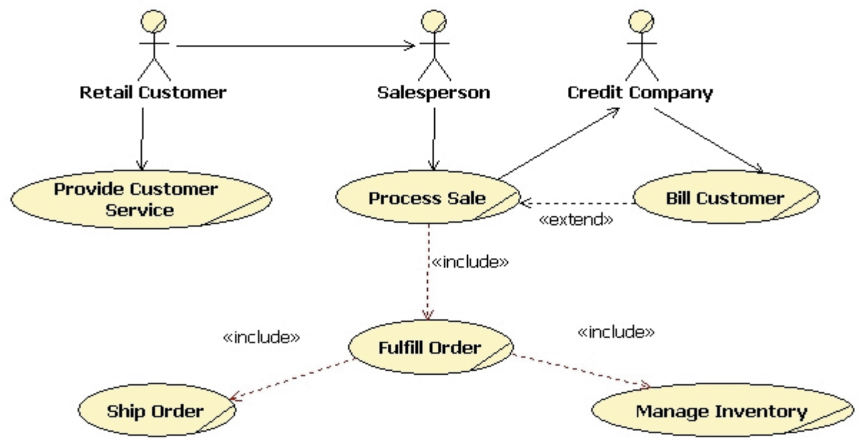

***Figure 1**: A Typical Business-Level Use-Case Diagram for a Retail System Project*[5]

- A use case may span time periods and include many subsystems and components. It will act as the "context glue," ensuring that all system elements fit together to deliver value. One major benefit of this approach is that the use cases define the system integration tests, enabling the system to be developed from that same context.

Combining use cases with traditional textual requirements allows systems projects to exploit the benefits of both of these requirements representations. A use case gives "transactional" context for the textual requirements and provides a great mechanism to structure the delivery of meaningful system increments. The textual requirements permit stakeholders to describe the system in their natural language.

Communication – Architecture, Visual Modeling, and the UML

Enterprise and Systems Architecture

The system architecture is "the fundamental and unifying system structure defined in terms of system elements, interfaces, processes, constraints, and behaviors.[6]" This definition can be used to plan development, manage risk, and drive that development to completion. Notice that the description of a

[5] *UML For Mere Mortals*; Robert A. Maksimchuk and Eric J. Naiburg; Addison-Wesley; 2004
[6] System Architecture Working Group; INCOSE, 1996; http://www.incose.org

system's architecture is very closely linked to the definition of an enterprise architecture as "... a 'blueprint' that documents all the information systems within the enterprise, their relationships, and how they interact to fulfill the enterprise's mission.[7]" This is intentional. An enterprise architecture is always a system, but a systems architecture is not always an enterprise. The term enterprise defines the scope of the system and its completeness. Thus to create an enterprise architecture requires an organization to create a complete definition of the system(s), for a particular context.

One of the primary challenges with enterprise architectures is the amount of time required up front in their creation. When trying to create an architecture that serves so many stakeholders' needs, the final product often takes too long to create and does not sufficiently serve anyone. Experience would indicate that an iterative and incremental approach to EA creation is crucial—and at the heart of that creation is the system architecture.

The system's architecture needs to provide enough detail to enable the system to be created. Some would argue that, realistically, a system should be described in enough detail to enable enterprise architecture activities to be undertaken on it; activities such as resource planning, IT portfolio management, etc. But, not all projects require such activities and thus a system's architecture should be complete enough to satisfy its stakeholders and to enable a system to be created.

Modeling Architectural Views

There are two primary dimensions to systems architecture:

- **Viewpoint** – A view of the architecture for a particular set of stakeholders that addresses a set of quality concerns.
- **Model level** – UML models that describe the various levels of design maturity.

The development of the system's architecture typically begins with modeling. The systems architect initially creates the context model, then moves on to logical analysis and design, and finally into implementation (the various model levels). Throughout this progression, it is possible to view the system model from different perspectives, such as enterprise workers, information,

[7] Ingredients for Building Effective Enterprise Architectures; Dave West, Kurt Bittner, and Eddie Glenn; Nov. 2002. http://www.therationaledge.com/content/nov_02/f_enterpriseArchitecture_dw.jsp

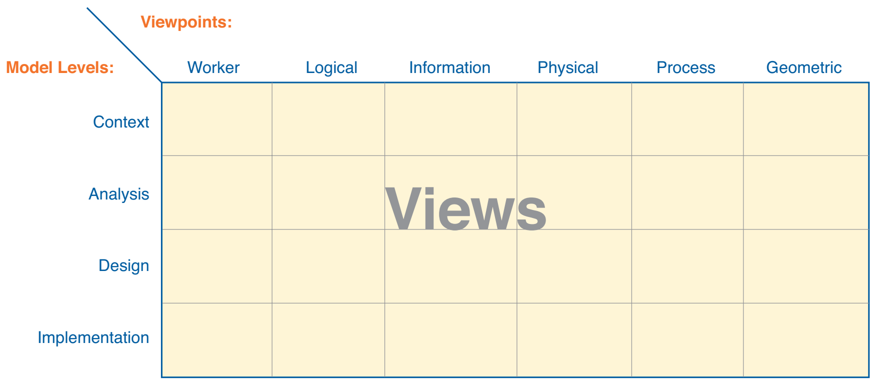

***Figure 2**: Views. Different views exist depending on your viewpoint and level of your model*

physical structure, and so forth (see Figure 2). This approach enables the architect to define and structure the system and at the same time allows stakeholders to view the architecture as it relates to their concerns.

Subsystems

A subsystem is a logical chunk of the system with a defined set of services. These services are the only thing that another subsystem can depend on. When used collectively, they provide the functionality described in the use cases. In other words, use cases are realized by a collection of subsystems providing services (see Figure 3).

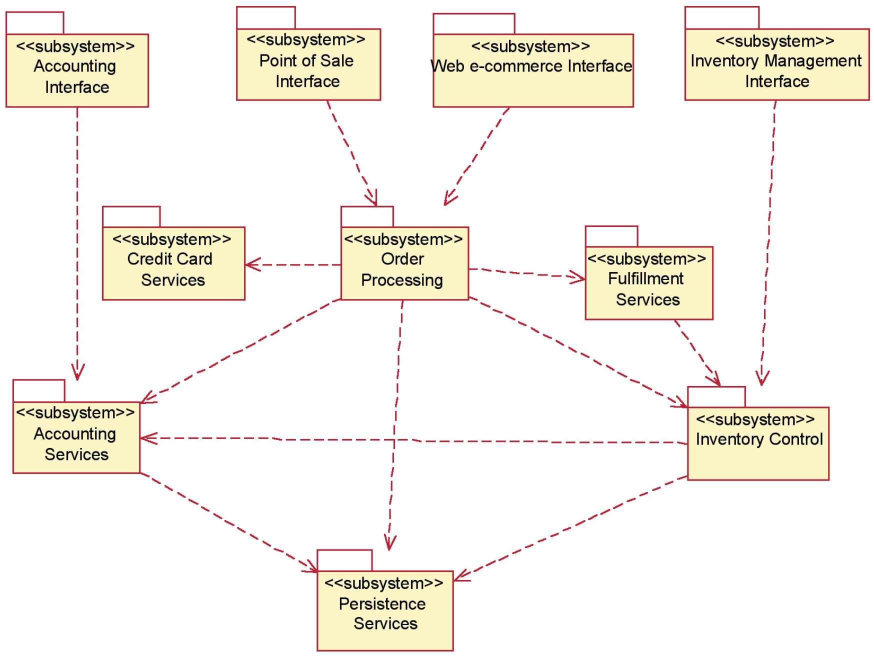
<<subsystem>> Accounting Interface
<<subsystem>> Point of Sale Interface
<<subsystem>> Web e-commerce Interface
<<subsystem>> Inventory Management Interface
<<subsystem>> Credit Card Services
<<subsystem>> Order Processing
<<subsystem>> Fulfillment Services
<<subsystem>> Accounting Services
<<subsystem>> Inventory Control
<<subsystem>> Persistence Services

***Figure 3**: Diagram of a Subsystem and its Dependencies Each subsystem defines a set of services that collectively provide the functionality described by the use cases.*

Localities

The "locality" is a mechanism to describe the logical locations to which the system processing is deployed, from an engineering viewpoint. Subsystem services are allocated to particular localities, which, in turn, can be realized via physical processes. Locality diagrams consist of two elements:

- **Localities** – A collection of computing and storage resources that can host processing.
- **Connections** – Information paths between physical localities.

By associating non-functional requirements such as quality, reliability, and performance to a locality, it is then possible to logically describe the specification for each location. In the case of connections, the locality is a great place to attribute throughput and management requirements to the system's architecture. Figure 4 shows an example of locality design, indicating specific connections between localities.

Use-Case Flowdown

This is an analysis/logical design-level activity used to derive functional requirements and associate them to the system elements. The primary outcomes of this activity are:

- A use-case survey for the subsystems

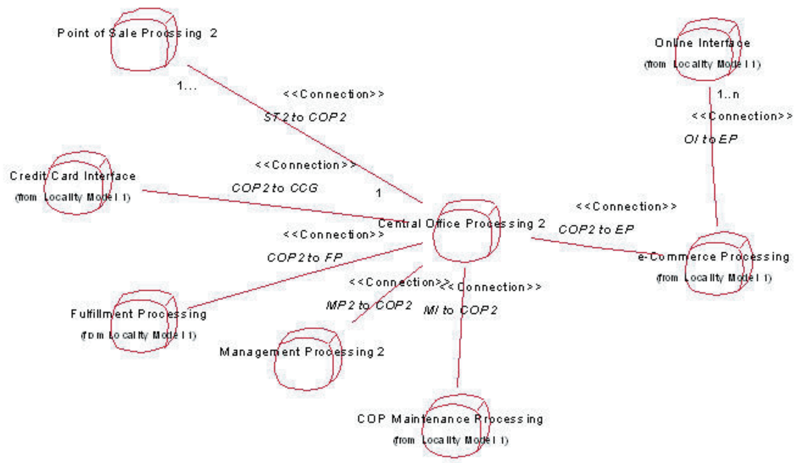

***Figure 4**: A Locality Diagram Indicates Specific Connections Between Localities*

- A survey of the hosted subsystem use cases for localities
- A survey of the realized subsystem use cases for processes.

The first step in performing the use-case flowdown is to develop the use-case descriptions, in which the system is treated as a black box. Then using typical object-oriented analysis and design techniques, candidate subsystems are established. Next, the subsystems are examined with regard to *how* they fulfill the use case, adding white box descriptions of what the subsystem needs to. The additional elements of subsystem, locality, and process are added to the flow, describing what the subsystem should do, where it should be done, and on what process. This process is repeated to elaborate further details as you proceed through the different model levels.

An important point to note is that the approach described above "builds up" the candidate architecture by adding in more information. It does not employ use cases to decompose requirements. Do not confuse use case flowdown with functional decomposition.

Visual modeling provides the development team with a way of both abstracting the problem and communicating its solution to other key stakeholders, and the UML provides an industry standard language for these models. The UML encourages a component-based approach to structuring the architecture. Components in the form of subsystems enable the development team to clearly partition the system into chunks that exhibit high cohesion, meaning that the system chunks make sense from a change perspective, and low coupling meaning that each component is not heavily dependent on other components to perform its function, which ensures a robust architecture for the future.

Process – System Engineering with the IBM Rational Unified Process

The IBM Rational Unified Process, or RUP, provides a framework for systems development. The key principles of RUP for systems development are:

- **Architecture** - RUP fosters a modular, component-based architecture. It advocates use of the UML for defining the systems architecture, which encourages methodical system design, development, and validation.

- **Concurrent Design** – The ability to develop a system with many different people in parallel is crucial for success. By providing a clear set of artifacts and process steps, it is possible to organize large groups of individuals. The architecture focus of the RUP encourages an environment that has structure and a team that mirrors that architecture, thus providing clear lines of responsibility among the components.
- **Iterative Development** – RUP advocates an iterative approach. Each iteration[8] mitigates risk by delivering functionality. As a key "best practice" advocated by RUP, risk management starts with the Inception phase, where risks regarding scope and functionality requirements are identified. Then, as the project moves through its lifecycle, additional risks regarding architecture, manufacturing, and deployment are identified and managed. This iterative approach enables a team to work effectively on incrementally building a system, while decreasing the overall risk profile.
- **Change Management** – By combining an architecture-centric approach with an iterative lifecycle, RUP provides a framework that supports change. On all projects changes will occur, and a good process actively manages those changes, providing guidance on how to document, implement, and test them. Given the size and complexity of most systems, it is important to manage the impact of change and to continually evaluate what components are affected and how that change will play out across the entire system architecture.

Conformance to Specification – Testing

Once you have defined the requirements of the system, designed and developed the system components in the context of the system architecture, it is crucial that you test these components. This is made even more important when you are working in an iterative and incremental fashion. Incremental development hastens testing, validation and verification in the lifecycle by

[7] "Iteration" refers to a working version of the system under development, however early or late in the development process. For a general introduction to the principles of iterative development in terms of RUP, see "What is the Rational Unified Process?" at http://www-106.ibm.com/developerworks/rational/rationaledge/ (Search the archives under the category "Rational Unified Process".)

making it happen early and happen throughout the development. As defined in the RUP testing applies to a number of core practices:

- Finding and documenting defects in system quality
- Generally advising about perceived system quality
- Proving and validating the assumptions made in the systems architecture and the requirements through concrete demonstration
- Validating the system functions as defined
- Validating that the requirements have been implemented

Defect Observation and Reporting

Because of the iterative nature of the Rational approach to systems development, the system is tested throughout the lifecycle. Even in early iterations, aspects of the system are tested. It is important that the observations that are found are documented and that every deliverable is considered to be worth reporting upon. Many times an aspect of the system does not work at the end of the project because it did not work at the beginning and it was assumed that a defect was documented and was being worked on.

The process of defect reporting must be made a natural process and not a massive overhead impact. The Rational approach provides, with the application of ClearQuest, a mechanism to enter defects on a web form or via a native application. These observations, which then may become defects, can happen in two forms of testing, formal testing against some sort of specification and exploratory testing. In the case of formal testing it is crucial to associate any defects against the specification that is being tested against, and in the context of, a particular test plan. For exploratory testing it is vital to have the ability to capture as much information as possible about the situation.

Measuring System Quality

Typically, the thoroughness with which systems are tested is proportional to the consequences if the system fails. There are diminishing returns of testing every system path in every context. It is therefore important to consider what aspects of the system to test and from that decide what the associated risk is of not testing all aspects. There are two primary artifacts that are crucial in helping the Quality Assurance professional make those decisions: the use case and the systems architecture.

The use cases describe the system in terms of the 'black box' behavior. It is written from the actor's perspective and does not describe how the system implements that behavior. Because use cases and the flows that they comprise are used as the primary driver for an iteration, testing the 'end to end' flow enables a QA professional to have a good idea about system quality. It enables them to judge how much functionality is being delivered and how well it works. Because the iteration occurs in the context of the risk, it is possible to map the risks to the quality measures and thus get a good feel for the system. For example, we have delivered x number of use cases and got y number of defects, but there are r number of risks still outstanding, thus the we really do not know what the quality is because we have not addressed all the risks.

The system architecture defines services and components that comprise the system. Testing those services in isolation, outside the context of the use case, provides a certain measure of quality. But having these services exercised in the context of a use case, enables those services to be tested against a specification of the customer's requirements.

A Process for Testing

Testing should be an integrated part of the overall systems development process. Having a clearly defined mission for testing, coupled with the right input artifacts from the other disciplines, enables testing to be a natural part of the process. Iterative development drives testing earlier in the lifecycle and encourages metrics on quality to be part of the reporting process even in the early phases of the lifecycle. Figure 5 depicts the testing discipline in the RUP. It highlights that the process of testing must be continually measured as well as the actual testing of the system and its components.

Independent Verification and Validation (IV&V)

Verification is traditionally the process that assesses whether the system was built correctly against its specification and answers the system question 'was the system built right?'. Validation is concerned whether the system solves

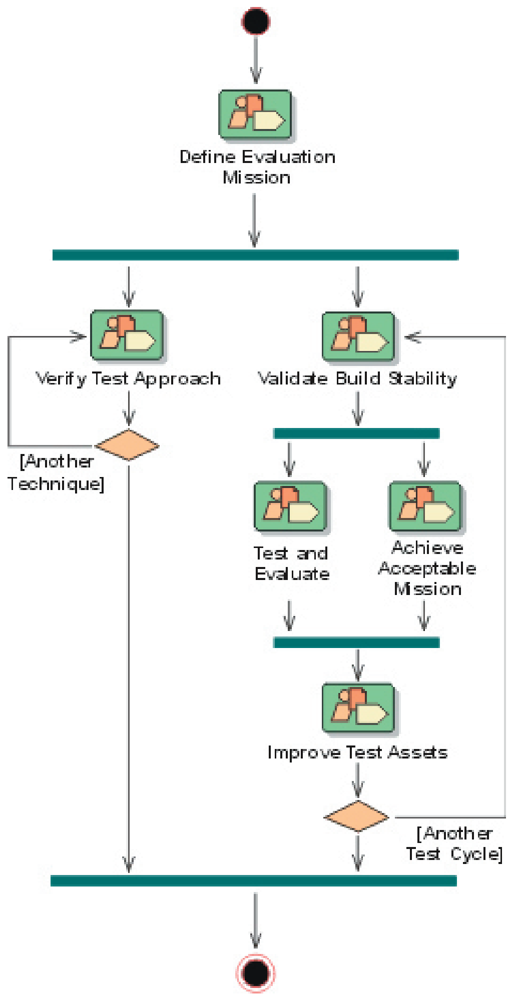

***Figure 5**: The testing discipline from the RUP*

the problem that it was intended to and answers the simple question ‘was the right system built?’.

As increments are delivered, each is tested in the context of its specification (verification) with defects being described against that specification. By making the iteration available to the customer and because the delivered functionality is a somewhat complete ‘slice’ of the system, as defined in a use case flow, it is possible for them judge if the iteration supports their particular need. Step by step, the system is built thus enabling management to have a true picture of the results of IV&V throughout.

Team collaboration – Configuration and Change Management

At any given time, a system can be described as being in various states: for example, its current state (how the system is working today) or various future states, often including a long term and a short term profile. Being able to manage this myriad of configurations for system definition is crucial, since it becomes harder and harder for the systems developer to keep a clear picture of the working configuration in mind.

Since the development of a system is a team activity, this challenge increases geometrically. The team may include people that are both inside and outside the boundary of one organization, in many locations, and involved in many activities. It is crucial that each team member can be working both in the same development process and from the same 'virtual' repository. We have already talked about process, and how having the whole team work from one development process, in the form of the RUP for Systems Engineering (a.k.a. RUP SE), is crucial in terms of knowing what their role is responsible for, what each artifact looks like, and what the dependencies are.

RUP SE is a configurable process, a process description, templates, guidelines and examples. It does not 'force' a team to abide by its tenets. For many development teams that is a good thing, allowing a team to make decisions themselves as to what they should do based on their experience and expertise, but still having the foundation of a shared knowledgebase. But, in one area this can prove problematic. That area is change management. It is important that team work to the same change process, that the states an artifact goes through are well known, and that state model is 'enforced' on the project. Coupling this state model with a shared repository that every team member can see, is at the heart of the IBM Rational solution for systems development.

IBM Rational provides an integrated solution that integrates change management of artifacts with a clearly defined and enforced process. Unified Change Management (UCM) is a workflow for automating change across the software lifecycle and across distributed, multi-functional development teams. At the heart of UCM is the ability to associate steps with a particular change request. Then when you check in and check out artifacts you can associate them with that step. Thus you work in the context of a "change record".

UCM helps managers reduce risk by coordinating and prioritizing the activities of the team and by ensuring that they work with the right sets of artifacts. Extending across the lifecycle to accommodate all project domain information—requirements, visual models, code, and test artifacts—it helps teams effectively "baseline" requirements along with code and test assets.

Once you have the infrastructure of a repository and an integrated activity-oriented change management process, the next step is to link this change process into the development process. By linking change requests to requirements it is possible for a program management office to manage change on a baselined set of requirements. By making the change management recording system available to users of the system, this allows them to add comments on a particular release (or iteration). This works particularly well when considering a system that has many user stakeholders. Managing their input is crucial for improving the quality of the final system and reducing the chaos of 1000s of change requests that are never mapped to new functions or releases.

Change management is often thought of as the boring part of systems development—a necessary evil. Having an automated and integrated change management process that fits into your whole systems development approach makes change management a real enabler for a successful iterative and incremental process.

Summary

A system is more than just software; it is a combination of people, software, and hardware that is unified in pursuit of a common objective. Proper development of a system requires thinking about how things should operate to support a particular business objective or mission. As described above, IBM Rational's approach to systems development is holistic, providing a unified process along with integrated tools. This approach can be broadly split into several areas: use cases and requirements; architecture, visual modeling and the UML; process for systems engineering; and configuration and change management. In each of these areas, IBM Rational provides thought leadership and tools, leveraging modern solutions to solve modern problems.

Building systems is not an easy activity. With the increased flexibility demanded by customers, coupled with a large team, many stakeholders and the pressure of numerous industry drivers, life on a systems project has a unique set of challenges. This whitepaper introduces a holistic approach to systems development. This holistic approach enables development projects to reduce the overall risk of delivery by having clearly integrated artifacts throughout the development process. With the clear set of best practices embodied in RUP—practices that have been used on numerous systems and software projects—this approach can yield success for even the most complex system development projects.

About the Authors

Dave West is a Group Manager at IBM Rational Software based in Lexington Massachusetts. At Rational, Dave has held a number of positions including technical engagement manager and product manager responsible for the Rational Unified Process product. Dave has worked as a consultant in many large organizations and programs providing practical support in the areas of project management, software architecture, and systems design. Dave has authored many whitepapers and articles on subjects ranging from developing component based systems to re-engineering the business. In his current role Dave runs a team responsible for developing industry solutions.

Robert A. Maksimchuk is a veteran systems engineer with over 25 years of hardware and software systems development experience in a widely diverse group of industries. For most of his career, Mr. Maksimchuk's focus has been using his object-oriented (OO) expertise to help numerous companies employ OO techniques to solve their business problems. He is co-author of the book *UML for Database Design* (ISBN 0-201-72163-5) and the upcoming *UML for Mere Mortals*, and has also written articles for various trade magazines. As an Evangelist, Product Manager, and Industrial Solutions Market Manager for IBM Rational, Mr. Maksimchuk has traveled worldwide, speaking in numerous technology forums and leading workshops and seminars on UML and OO development.

IBM software integrated solutions

IBM Rational supports a wealth of other offerings from IBM software. IBM software solutions can give you the power to achieve your priority business and IT goals.

- DB2® software helps you leverage information with solutions for data enablement, data management, and data distribution.
- Lotus® software helps your staff be productive with solutions for authoring, managing, communicating, and sharing knowledge.
- Tivoli® software helps you manage the technology that runs your e-business infrastructure.
- WebSphere® software helps you extend your existing business-critical processes to the Web.
- Rational® software helps you improve your software development capability with tools, services, and best practices.

www.ibm.com