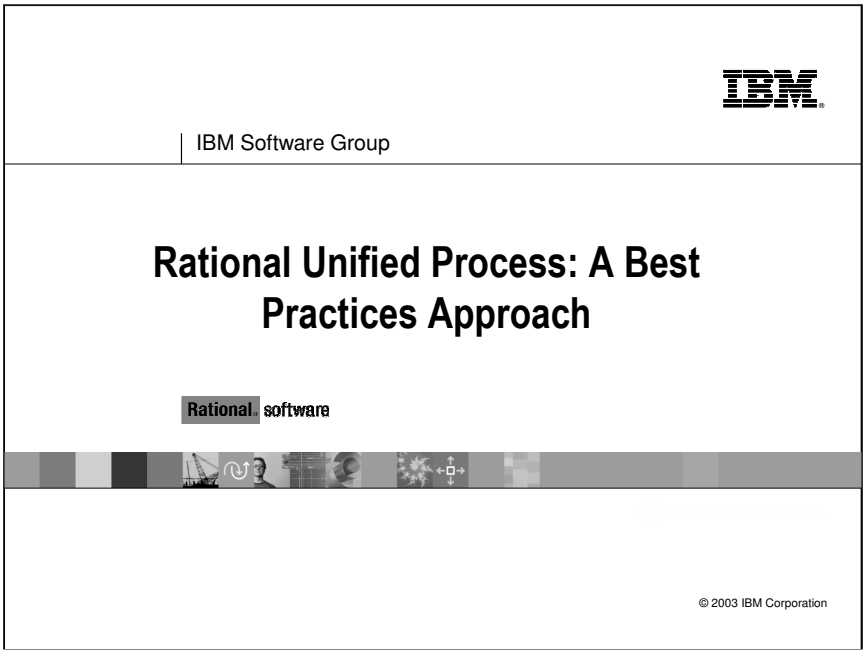

IBM Software Group
Rational Unified Process: A Best Practices Approach
Rational. software
© 2003 IBM Corporation

Topics

- What is RUP?
- RUP best practices
- Software economics
- Adapt the process

IBM

The new RUP language is a unification of different method and process engineering languages such as the SPEM extension to the UML for software process engineering, the languages used for defining content and process for RUP v7.0, IBM Global Services Method, as well as IBM Rational Summit Ascendant.

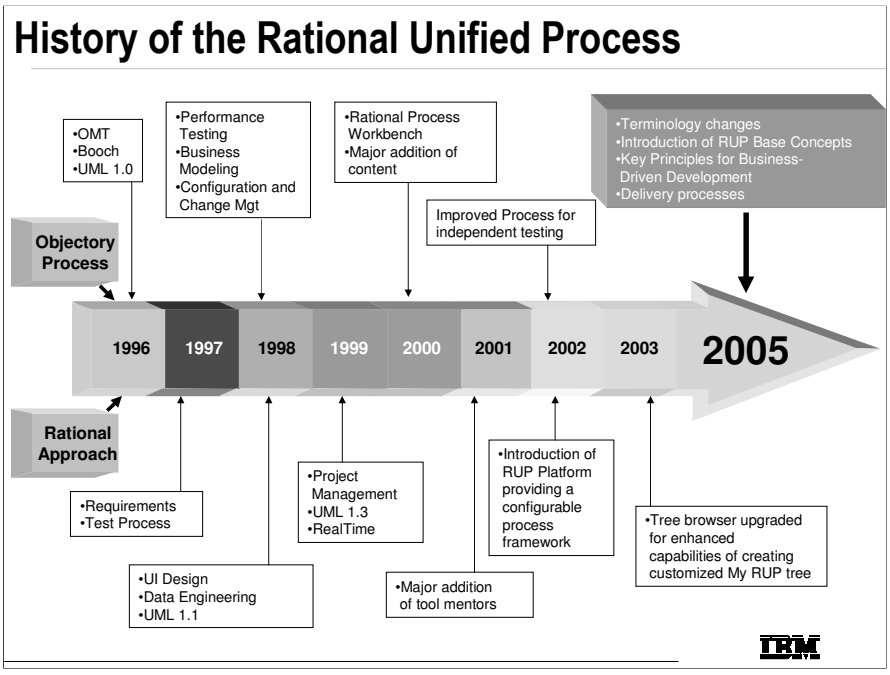
What is the Rational Unified Process (RUP)?

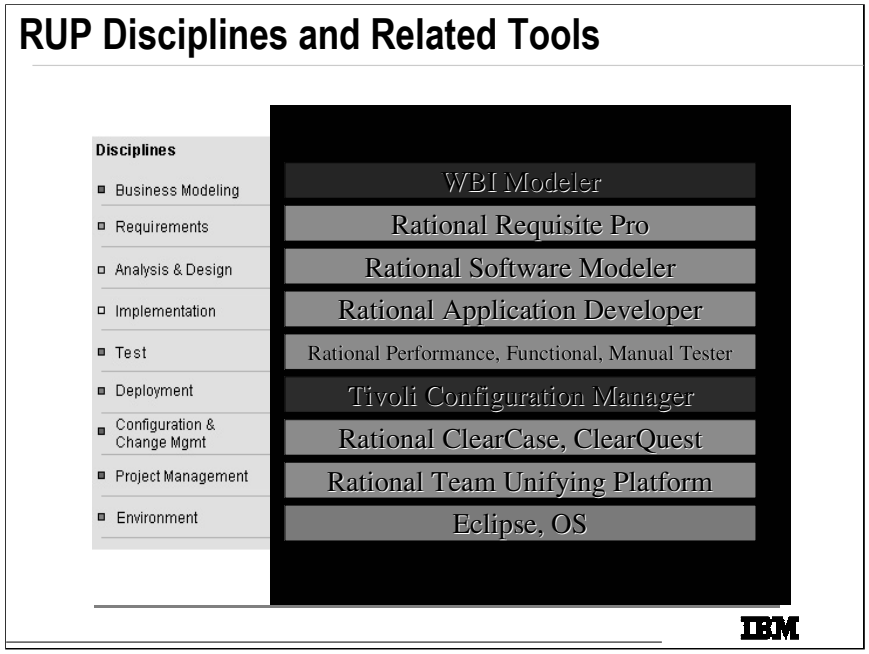
There are three central elements that define RUP:

1. An underlying set of principles for successful software development.
   - These principles are the foundation on which the RUP has been developed.
2. A framework of reusable method content and process building blocks.
   - A family of method plug-ins defines a method framework from which you create your own method configurations and tailored processes.
3. The underlying method and process definition language.
   - A unified method architecture meta-model that provides a language for describing method content and processes.

IBM

The new RUP language is a unification of different method and process engineering languages such as the SPEM extension to the UML for software process engineering, the languages used for defining content and process for RUP v7.0, IBM Global Services Method, as well as IBM Rational Summit Ascendant.

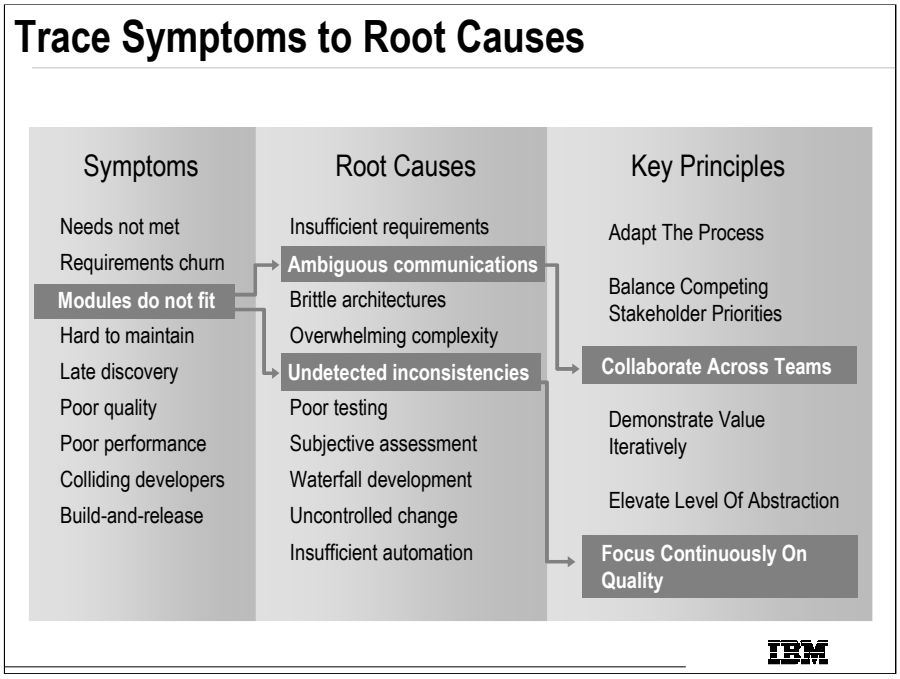

History of the Rational Unified Process
•OMT
•Booch
•UML 1.0
•Performance Testing
•Business Modeling
•Configuration and Change Mgt
•Rational Process Workbench
•Major addition of content
Improved Process for independent testing
•Terminology changes
•Introduction of RUP Base Concepts
•Key Principles for Business-Driven Development
•Delivery processes
Objectory Process
1996
1997
1998
1999
2000
2001
2002
2003
2005
Rational Approach
•Requirements
•Test Process
•UI Design
•Data Engineering
•UML 1.1
•Project Management
•UML 1.3
•RealTime
•Major addition of tool mentors
•Introduction of RUP Platform providing a configurable process framework
•Tree browser upgraded for enhanced capabilities of creating customized My RUP tree
IBM


RUP Disciplines and Related Tools
Disciplines
Business Modeling
Requirements
Analysis & Design
Implementation
Test
Deployment
Configuration & Change Mgmt
Project Management
Environment
WBI Modeler
Rational Requisite Pro
Rational Software Modeler
Rational Application Developer
Rational Performance, Functional, Manual Tester
Tivoli Configuration Manager
Rational ClearCase, ClearQuest
Rational Team Unifying Platform
Eclipse, OS
IBM

Symptoms of Software Development Problems

- ✓ User or business needs not met
- ✓ Requirements churn
- ✓ Modules do not integrate
- ✓ Hard to maintain
- ✓ Late discovery of flaws
- ✓ Poor quality or end-user experience
- ✓ Poor performance under load
- ✓ No coordinated team effort
- ✓ Build-and-release issues

IBM



Treat these root causes, and you will eliminate the symptoms. Eliminate the symptoms, and you will be in a much better position to develop quality software in a repeatable and predictable fashion.

The Key principles are a set of commercially-proven approaches to software development, which when used in combination, strike at the root causes of software development problems. These are so-called “key principles,” not because you can precisely quantify their value, but rather because we observe their common use in the industry by successful organizations.

The key principles are harvested from thousands of customers on thousands of projects and from industry experts.

Best Practices of Software Engineering

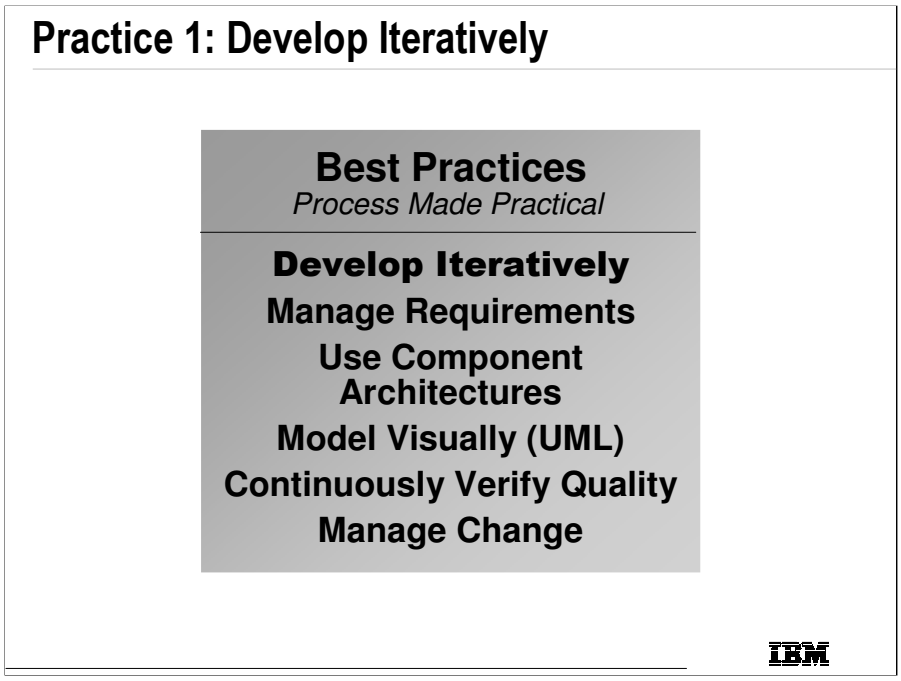
Best Practices
Process Made Practical

Develop Iteratively
Manage Requirements
Use Component Architectures
Model Visually (UML)
Continuously Verify Quality
Manage Change

IBM

Developing iteratively is a technique that is used to deliver the functionality of a system in a successive series of releases of increasing completeness. Each release is developed in a specific, fixed time period called an “iteration.”

Each iteration is focused on defining, analyzing, designing, building and testing some set of requirements.

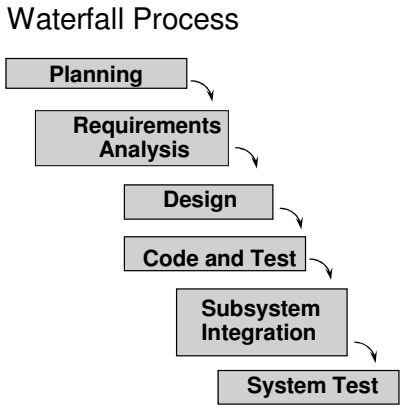


Developing iteratively is a technique that is used to deliver the functionality of a system in a successive series of releases of increasing completeness. Each release is developed in a specific, fixed time period called an “iteration.”

Each iteration is focused on defining, analyzing, designing, building and testing some set of requirements.

To illustrate a problem with the waterfall model: Suppose I estimate that the project will take two years, and it really takes three years. At the end of two years, what do I have? Nothing useful works. No partial delivery is possible. Diagrams and models are great, but they can not execute.

Waterfall Development Characteristics



- Delays confirmation of critical risk resolution.
- Measures progress by assessing work-products that are poor predictors of time-to-completion.
- Delays and aggregates integration and testing.
- Precludes early deployment.
- Frequently results in major unplanned iterations.

IBM

Waterfall is conceptually straightforward because it produces a single deliverable for each step (requirements, analysis model, design model, code, etc), resulting in a single release. The fundamental problem is that it pushes risk forward in time, where it's costly to undo mistakes from earlier phases. An initial design will likely be flawed with respect to its key requirements, and furthermore, the late discovery of design defects tends to result in costly overruns or project cancellation. The waterfall approach tends to mask the real risks to a project until it is too late to do anything meaningful about them.

Enable Feedback by Delivering Incremental User Value

- Divide the project into a set of *iterations*
  - In each iteration, we perform some requirements, design, implementation, and testing of the application, producing a deliverable that is one step closer to the solution.
- Obtain feedback from stakeholders to find out:
  - Are we moving in the right direction?
  - Are stakeholders satisfied so far?
  - Do we need to change the features implemented so far?
  - What additional features need to be implemented to add business value?

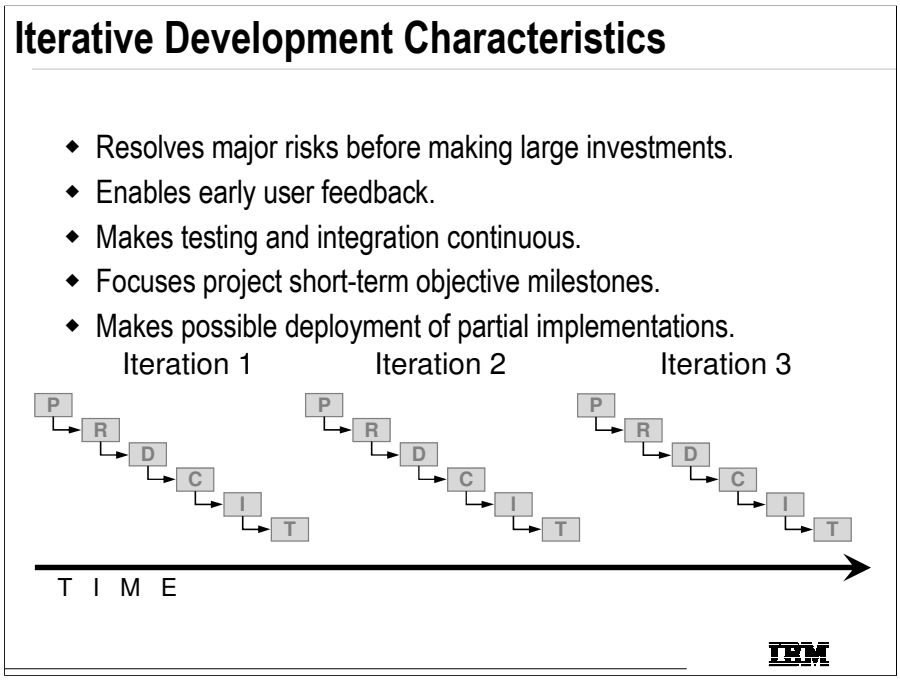
IBM

By assessing stakeholder feedback we are more likely to:

- build trust among stakeholders
- ensure that the system we are developing will address their needs

And less likely to:

- over-engineer our approach
- add capabilities that are not useful to the end user.



Iterative processes were developed in response to these waterfall characteristics. With an iterative process, the waterfall steps are applied *iteratively.* Instead of developing the whole system in lock step, an increment (that is, a subset of system functionality) is selected and developed, then another increment, and so on. The selection of the first increment to be developed is based on risk, the highest priority risks first. To address the selected risk(s), choose a subset of use cases. Develop the *minimal* set of use cases that will allow objective verification, for example, through a set of executable tests, of the risks that you have chosen. Then select the next increment to address the next highest risk, and so on. Thus you apply the waterfall within each iteration, and the system evolves incrementally.

P: Planning

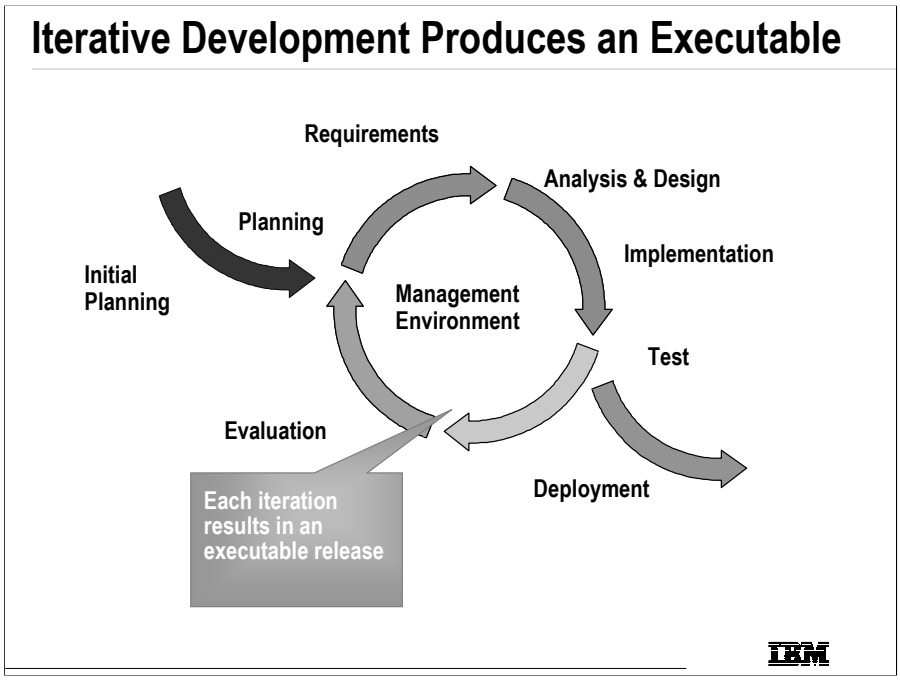
R: Requirements analysis

D: Design

C: Code and unit test

I: Integration

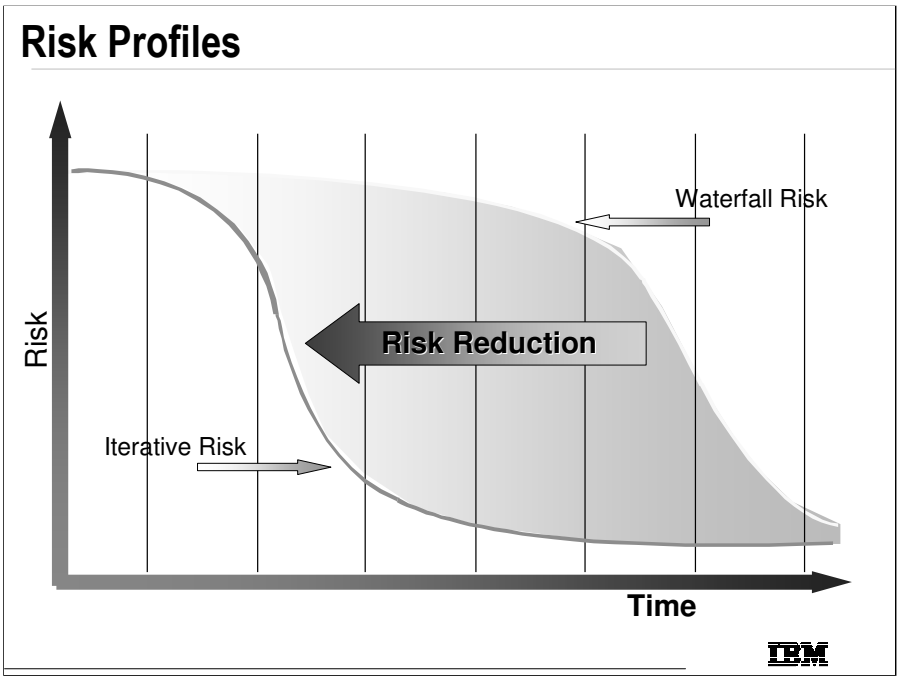
T:Test



The earliest iterations address greatest risks. Each iteration produces an executable release. Each iteration includes integration and test. Iterations help:

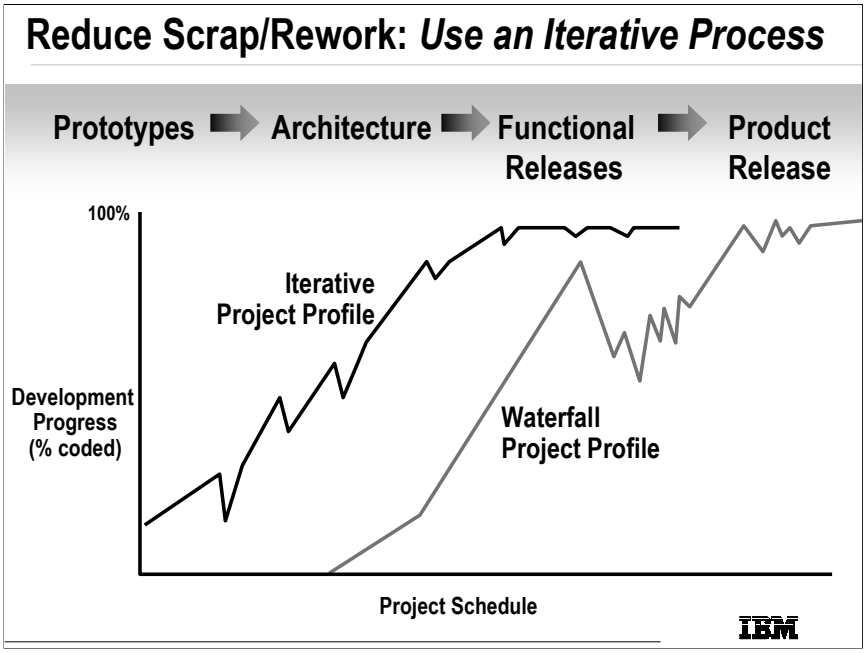
- resolve major risks before making large investments.
- enable early user feedback.
- make testing and integration continuous.
- focus project short-term objective milestones.
- make possible deployment of partial implementations.

Iterative processes were developed in response to these waterfall characteristics. With an iterative process, the waterfall steps are applied iteratively. Instead of developing the whole system in lock step, an increment (i.e. a subset of system functionality) is selected and developed, then another increment, etc. The selection of the first increment to be developed is based on risk, the highest priority risks first. To address the selected risk(s), choose a subset of use cases. Develop the *minimal* set of use cases that will allow objective verification (that is, through a set of executable tests) of the risks that you have chosen. Then select the next increment to address the next highest risk, and so on. Thus you apply the waterfall within each iteration and the system evolves incrementally.



Iterative development produces the architecture first, allowing integration to occur “as the verification activity” of the design phase and allowing design flaws to be detected and resolved earlier in the lifecycle. Continuous integration throughout the project replaces the “big bang” integration at the end of a project.

Iterative development also provides much better insight into quality, because of system characteristics that are largely inherent in the architecture. For example, performance, fault tolerance, and maintainability are tangible earlier in the process. Thus, issues are still correctable without jeopardizing target costs and schedules.

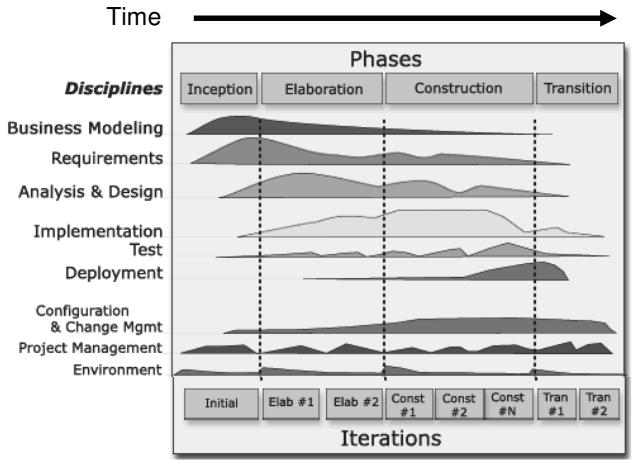


How you can avoid late life-cycle design breakages and project plan slippages? Get your teams to use an iterative development process.

This graphic depicts how an iterative development would compare to a conventional project from the development progress perspective and how the project schedule can actually be shortened.

The architecture-first approach forces integration into the elaboration phase, with demonstrations validating the design and requirements. The demonstrations do not eliminate the design breakage, they just make it happen in the design phase where it can be fixed more efficiently. This reduces the risk of the project by confronting it early in the development process. In an iterative process, the system is "grown" from an immature prototype to a baseline architectural skeleton to increments of useful capabilities and then finally, complete product releases. The downstream integration nightmare is avoided along with late patches and shoe-horned software fixes, thereby resulting in a more robust and maintainable design.

RUP Organization Along Time



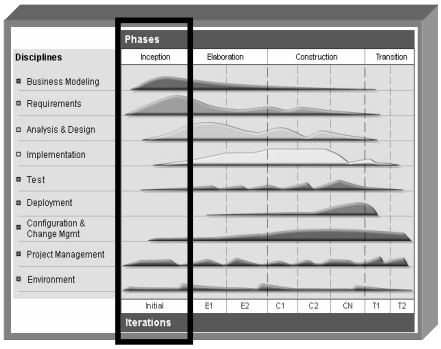
Organization by phases helps minimize the risks of resource allocation.

The time aspect of the process is enacted through phases, iterations, and milestones (end of phase objectives).

Progressing by meeting milestones helps minimize wasted resources since they are allocated only on a firm basis.

Inception Phase: Objectives

- Establish project scope and boundary conditions
- Determine the use cases and primary scenarios that will drive the major design trade-offs
- Demonstrate a candidate architecture against some of the primary scenarios
- Estimate the overall cost and schedule
- Identify potential risks (the sources of unpredictability)
- Prepare the supporting environment for the project

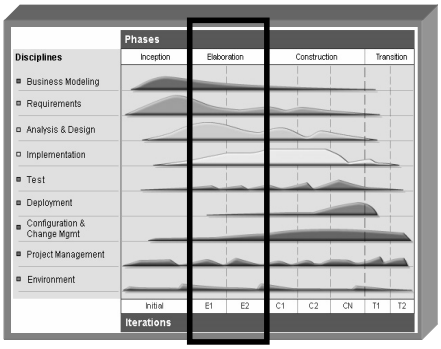


17

Note: In addition to an estimate for the Elaboration phase, an overall estimate of cost and schedule to complete the project is usually made at this time. However, this estimate must be recognized as being very rough (low confidence) and subject to revision at the end of Elaboration.

Elaboration Phase: Objectives

- Define, validate, and baseline the architecture as rapidly as is practical
- Address architectural significant risks
- Baseline the vision
- Baseline a detailed plan for the Construction phase
- Demonstrate that the baseline architecture will support the vision at a reasonable cost in a reasonable period of time
- Refine support environment



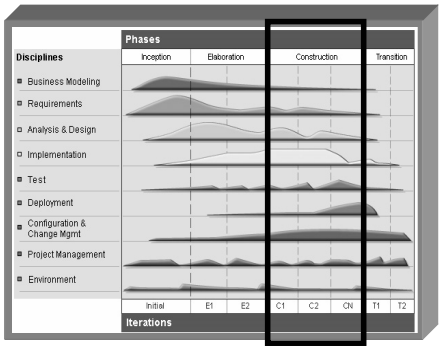
18

IBM

The cost and schedule to complete are re-estimated at the end of this phase. At this point, they are considered stable (high confidence), and firm commitments can be made.

Construction Phase: Objectives

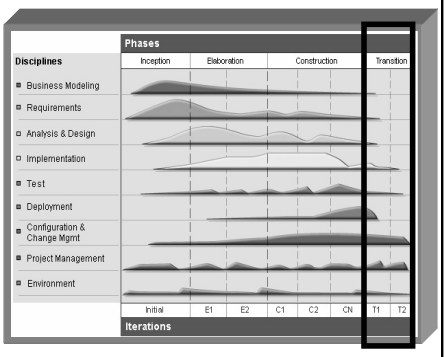
- Complete the software product for transition to production
- Minimize development costs by optimizing resources and avoiding unnecessary scrap and rework
- Achieve adequate quality as rapidly as is practical
- Achieve useful versions (alpha, beta, and other test releases) as rapidly as possible



19

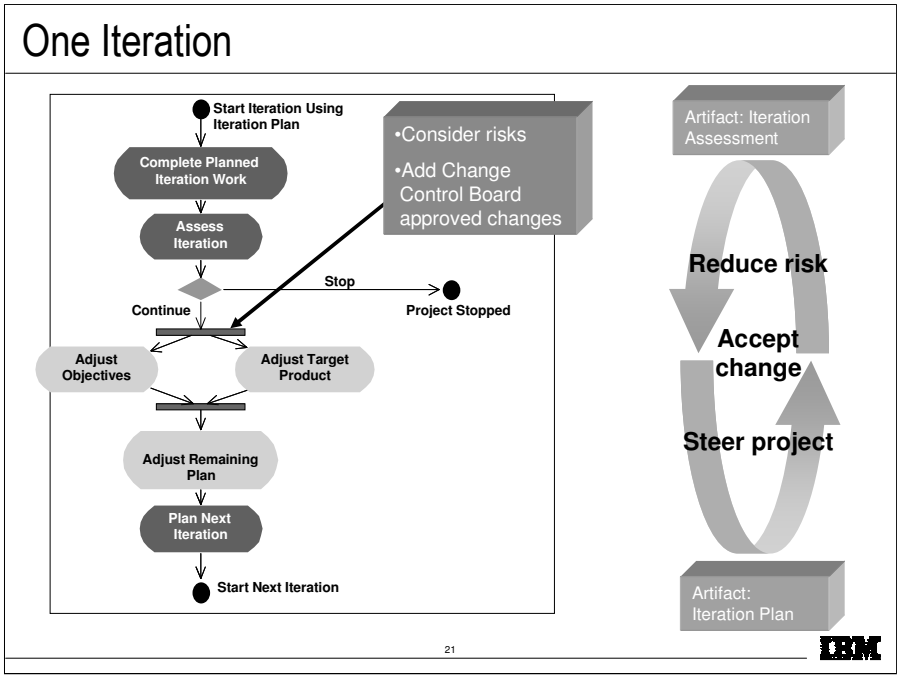
Transition Phase: Objectives

- Achieve user self-supportability
- Achieve stakeholder concurrence that deployment baselines are complete and consistent with the evaluation criteria of the vision
- Achieve final product baseline in a rapid and cost-effective manner



20

During the Transition phase, a decision is made whether to release the product.



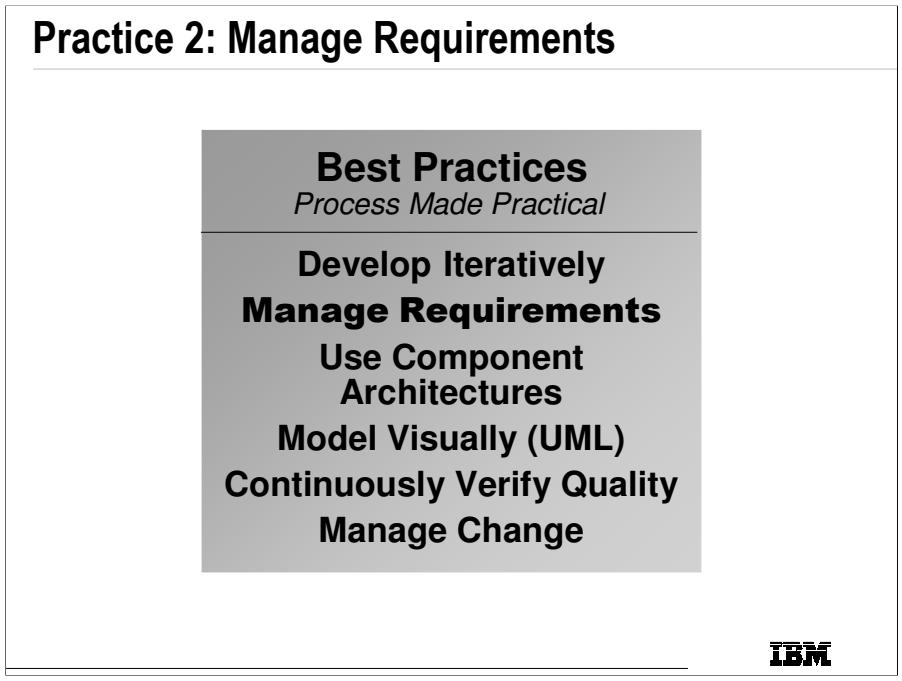
Definition of Artifact – a subtype of Work Product: A piece of information that: 1) is produced, modified, or used by a process, 2) defines an area of responsibility, and 3) is subject to version control. An artifact can be a model, a model element, or a document. A document can enclose other documents.

Two iteration planning artifacts in RUP are the Iteration Plan and the Iteration Assessment. In combination, they facilitate decisions that allow you to reduce risk, accept change, and steer the project through each iteration.

After an iteration starts, the teams complete the work specified in the Iteration Plan.

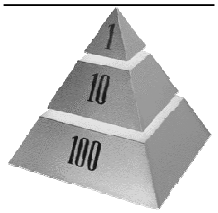
When work is complete, the Iteration Assessment is performed to determine if and how the iteration goals were achieved, using as many objective measurements as possible. Based on the assessment, a determination is made whether or not to continue the project.

If you decide to continue, you have to analyze Change Control Board (CCB)-approved project changes, revise your risk list and possibly modify the product's objectives (requirements) or the specifications of the product itself (architecture and design). This revised specification for the project becomes the new target and allows you to steer the project, taking into consideration requirements and product changes. Based on this adjusted set of objectives, you can plan the next iteration, creating a new Iteration Plan.



A report from the Standish Group confirms that a distinct minority of software development projects is completed on-time and on-budget. In their report, the success rate was only 16.2 percent, while challenged projects (operational, but late and over-budget) accounted for 52.7 percent. Impaired projects (canceled) accounted for 31.1 percent. These failures are attributed to poor requirements management, incorrect definition of requirements from the start of the project, and poor requirements management throughout the development lifecycle. (Source: Chaos Report, http://www.standishgroup.com)

Requirements Management



Making sure you

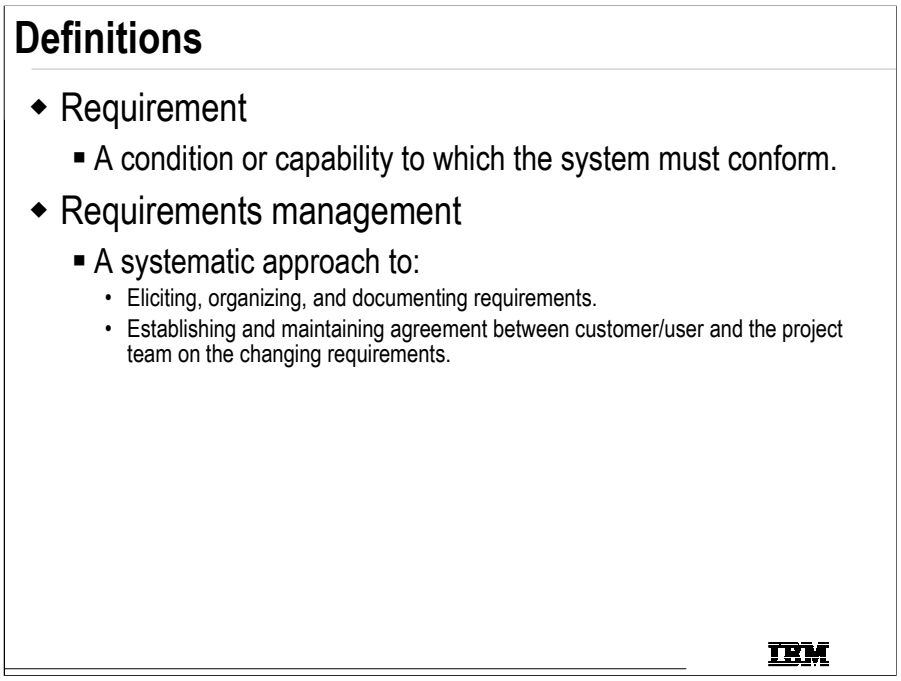
- solve the right problem
- build the right system

by taking a systematic approach to

- eliciting
- organizing
- documenting
- managing

the changing requirements of a software application.

IBM



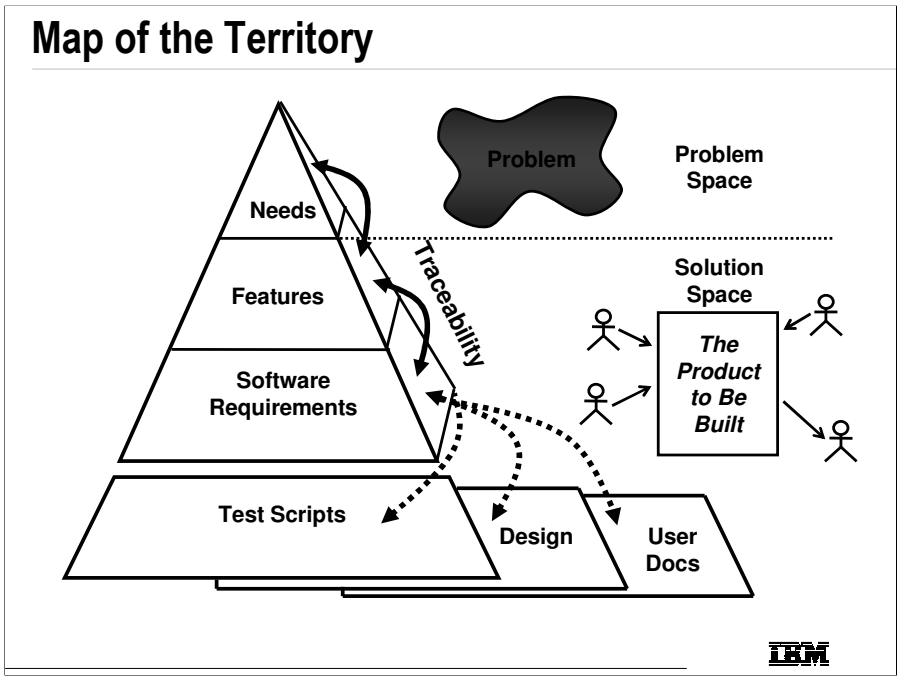
Definitions:

- **RUP:** A requirement describes a condition or capability to which a system must conform, either derived directly from user needs, or stated in a contract, standard, specification, or other formally imposed document.
- **UML:** A desired feature, property, or behavior of the system.

Requirements specify what the system must do rather than how the system does it.

There are many kinds of requirements. For example, *feature* requirements are requirements that are directly tied to user needs, and *software* requirements give the detailed requirements for the software. These different types of requirements are discussed later in the course.

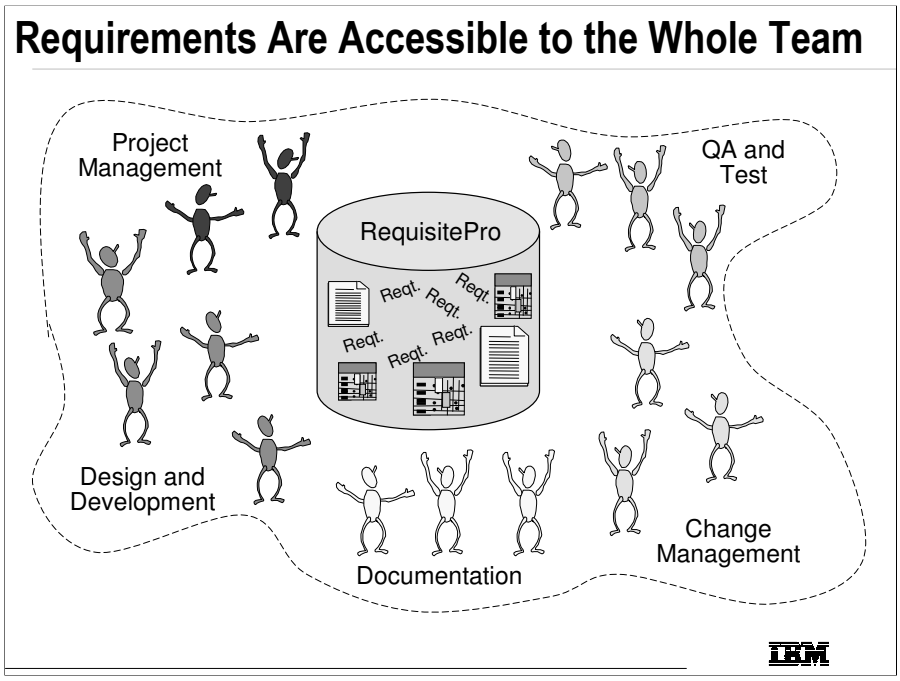
Do not expect to “get it right first time”. Requirements management is successful only if it allows for uncertainty of the requirements early in the project. However, requirements management also ensures that requirements become better defined over time.



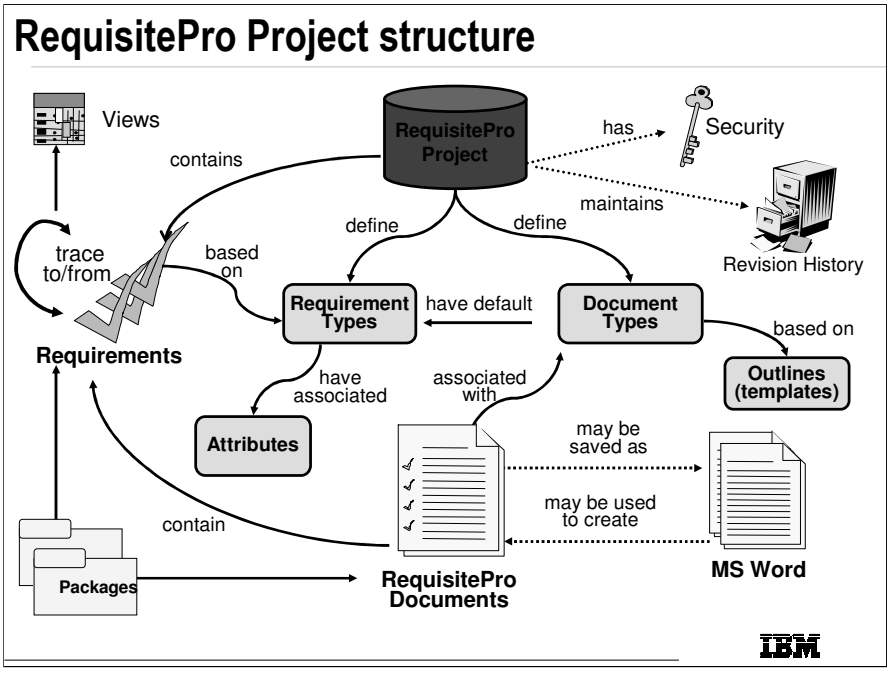
Managing requirements involves the translation of stakeholder requests into a set of key stakeholder needs and system features. These in turn are detailed into specifications for functional and nonfunctional requirements. Detailed specifications are translated into test procedures, design, and user documentation.

Traceability allows us to:

- Assess the project impact of a change in a requirement
- Assess the impact of a failure of a test on requirements (that is, if the test fails, the requirement may not be satisfied)
- Manage the scope of the project
- Verify that all requirements of the system are fulfilled by the implementation
- Verify that the application does only what it was intended to do
- Manage change



Effective requirements management requires you to organize your requirements so that they are available to the whole team. It also requires you to control change and ensure that your project does not spiral out of control as changes occur. When a change occurs, it must be communicated effectively, and the impact of the change must be fully understood.



Your Requirements Management Plan dictates your project structure in RequisitePro. Your RM Plan specifies the types of requirements you want to capture, the relationships between the requirement types, and the attributes you want to capture with each requirement.

This slide shows a high-level overview of all the components in a RequisitePro project and how they relate to each other. Requirement types, attributes, and document types define RequisitePro project structure.

Every requirement is associated with a requirement type. All requirements are maintained in the project database but can be located in documents as well. Requirements may have relationships among or dependencies upon one another. They can be traced from one requirement to another.

Practice 3: Use Component Architectures

Best Practices
Process Made Practical

Develop Iteratively
Manage Requirements
Use Component Architectures
Model Visually (UML)
Continuously Verify Quality
Manage Change

IBM

Software architecture is the development product that gives the highest return on investment with respect to quality, schedule, and cost, according to the authors of *Software Architecture in Practice* (Len Bass, Paul Clements & Rick Kazman [1998] Addison-Wesley). The Software Engineering Institute (SEI) has an effort underway called the Architecture Tradeoff Analysis (ATA) Initiative to focus on software architecture, a discipline much misunderstood in the software industry. The SEI has been evaluating software architectures for some time and would like to see architecture evaluation in wider use. By performing architecture evaluations, AT&T reported a 10 percent productivity increase (from news@sei, Vol. 1, No. 2).

Focus on Architecture First

- Design, implement, and test the architecture early in the project.
- Focus on the following goals:
  - Define the high-level building blocks and the most important components, their responsibilities, and their interfaces.
  - Design and implement architectural mechanisms.

Getting the architecture right early-on makes it easier to manage complexity. It also helps to identify what reusable assets we can leverage, and what aspects of the system need to be custom built.

IBM

Resilient Component-Based Architectures

- Resilient
  - Meets current and future requirements
  - Improves extensibility
  - Enables reuse
  - Encapsulates system dependencies
- Component-based
  - Reuse or customize components
  - Select from commercially available components
  - Evolve existing software incrementally

IBM

Architecture is a part of design. It is about making decisions on how the system will be built. But it is not all of the design. It stops at the major abstractions, or in other words, the elements that have some pervasive and long-lasting effect on the system's performance and ability to evolve.

A software system's architecture is perhaps the most important aspect that can be used to control the iterative and incremental development of a system throughout its lifecycle.

The most important property of an architecture is resilience—flexibility in the face of change. To achieve it, architects must anticipate evolution in both the problem domain and implementation technologies to produce a design that can gracefully accommodate such changes. Key techniques are abstraction, encapsulation, and object-oriented analysis and design. The result is that applications are fundamentally more maintainable and extensible.

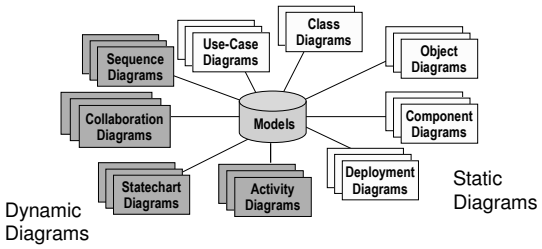
Practice 4: Model Visually (UML)

IBM

A **model** is a simplification of reality that provides a complete description of a system from a particular perspective. We build models so that we can better understand the system we are modeling. We build models of complex systems because we cannot comprehend any such system in its entirety.

Why Model Visually?

- Capture structure and behavior.
- Show how system elements fit together.
- Keep design and implementation consistent.
- Hide or expose details as appropriate.
- Promote unambiguous communication.



UML provides one language for all practitioners.

IBM

Modeling is important because it helps the development team visualize, specify, construct, and document the structure and behavior of a system's architecture. Using a standard modeling language, such as the Unified Modeling Language (UML), different members of the development team can communicate their decisions unambiguously to one another.

Using visual modeling tools facilitates the management of these models, letting you hide or expose details as necessary. Visual modeling also helps you maintain consistency among a system's artifacts—its requirements, designs, and implementations. In short, visual modeling helps improve a team's ability to manage software complexity.

Use Higher-Level Tools and Languages

- Leverage higher-level tools, frameworks, and languages:
  - Standard languages such as UML and EGL facilitate collaboration around high-level constructs while hiding unnecessary details.
  - Design and construction tools can automate moving from high-level constructs to working code.
  - Portfolio management tools allow you to manage financial and other aspects of multiple projects as one entity versus as a set of separate entities.

High-level tools capture key information graphically, which is a powerful and attractive way to present this information.

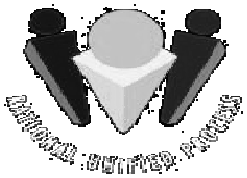
IBM

UML: Unified Modeling Language

EGL: Enterprise Generation Language

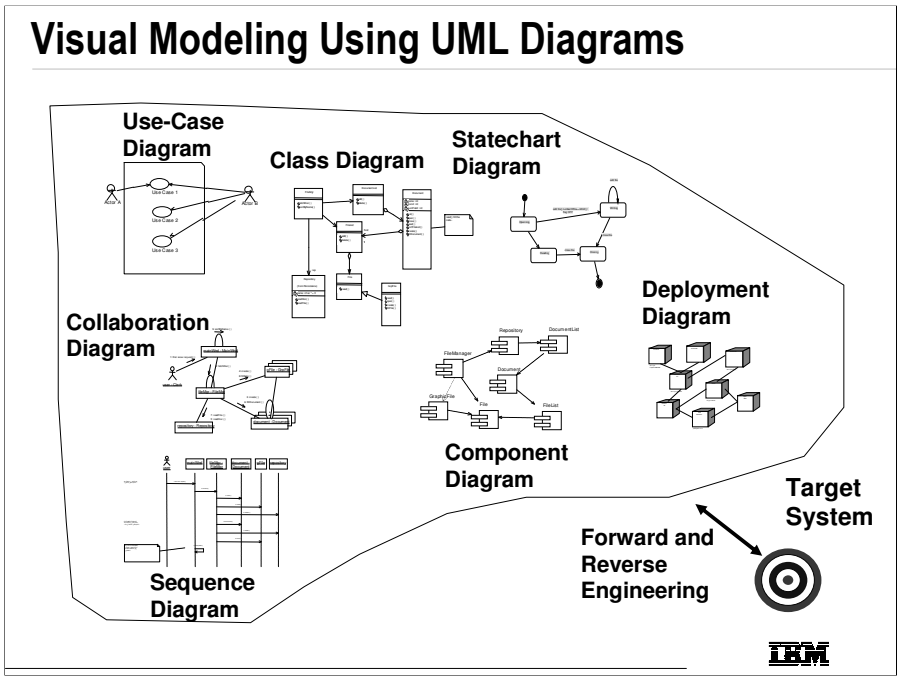
Common Language for Process and Design

- The UML is a common language for
  - Visualizing
  - Specifying
  - Constructing
  - Documenting
- RUP is a common language for
  - Process
  - Roles
  - Activities
  - Workflows
  - Iterations



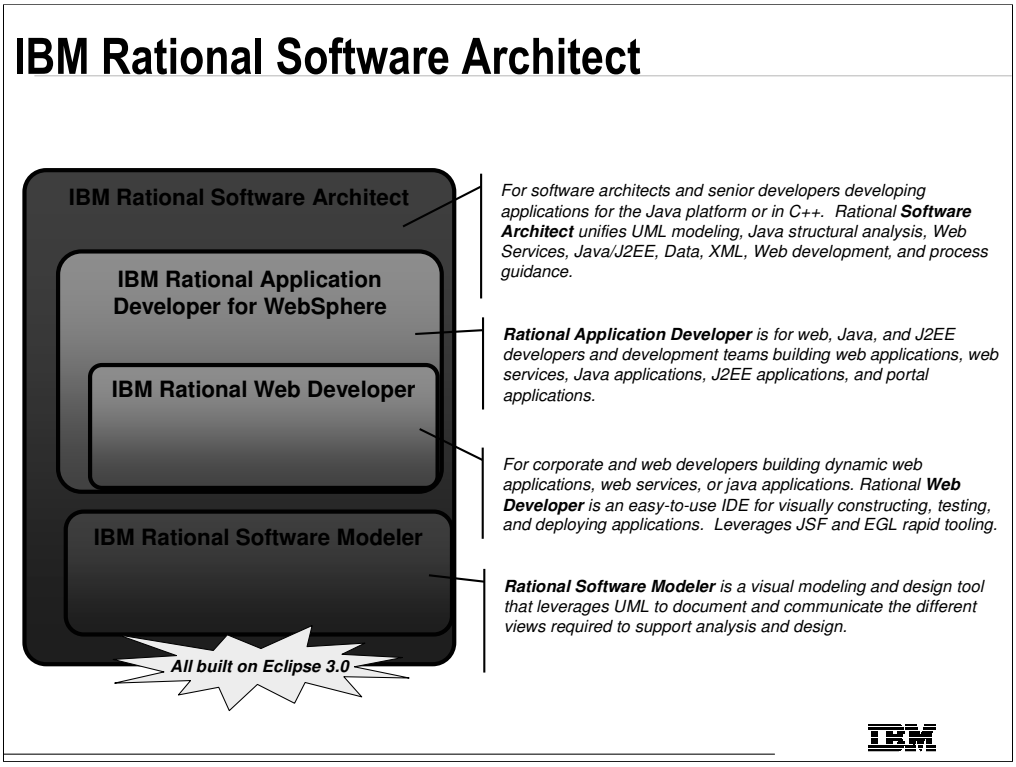
IBM

To increase comprehension, use a common language like the Unified Modeling Language to express models. The UML has been adopted as an industry standard. It is platform-independent. It defines a graphical language for presenting models and defines the semantics for each graphical element.



Visual modeling with the UML makes an application's architecture tangible, permitting us to assess it in multiple dimensions. How portable is it? Can it exploit expected advances in parallel processing? How might we modify it to support a family of applications? We've discussed the importance of architectural resilience and quality. The UML enables us to evaluate these key characteristics during early iterations—at a point when design defects can be corrected before threatening project success.

Advances in forward and reverse engineering techniques permit changes to an application's model to be automatically reflected in its source code, and changes to its source code to be automatically reflected in its model. This is critical when using an iterative process, where we expect such changes with each iteration.



RSA - *For software architects and senior developers developing applications for the Java platform or in C++, **Software Architect** is a design and construction tool for developing well architected applications, including applications on a Service Oriented Architecture. **Software Architect** unifies UML modeling, Java structural analysis, Web Services, Java/J2EE, Data, XML, Web development, and process guidance.*

RAD – *For web, Java, and J2EE developers and development teams building web applications, web services, Java applications, J2EE applications, and portal applications. **Application Developer** is a comprehensive IDE for visually designing, constructing, testing, profiling, and deploying applications. **Application Developer** improves application design and performance while increasing individual and team productivity.*

RWD – *For corporate and web developers building dynamic web applications, web services, or java applications. **Web Developer** is an easy-to-use IDE for visually constructing, testing, and deploying applications. Leverages JSF and EGL rapid tooling.*

RSM - *For architects, system analysts, and designers that need to ensure that their specifications, architecture, and designs are clearly defined and communicated with their stakeholders. **Software Modeler** is a visual modeling and design tool that leverages UML to document and communicate*

IBM Rational Software Architect Overview

"Application Analyzer"

- Automatic anti-pattern and pattern detection
- Architectural discovery, analysis, metrics, and stability reporting
- Implementation level architectural rules

"Modeler"

- UML 2.0 Diagrams for Class, Communication, Component, Composite Structure, Deployment, Activity, Sequence, State, and Use Case
- OCL Support
- Automatic diagram generation
- Pattern content
- Pattern/Transform authoring framework and services
- Extensive open API
- Java-based "scripting" for extensibility
- HTML and XML based data extraction and reporting
- Extensive printing
- RAS tools

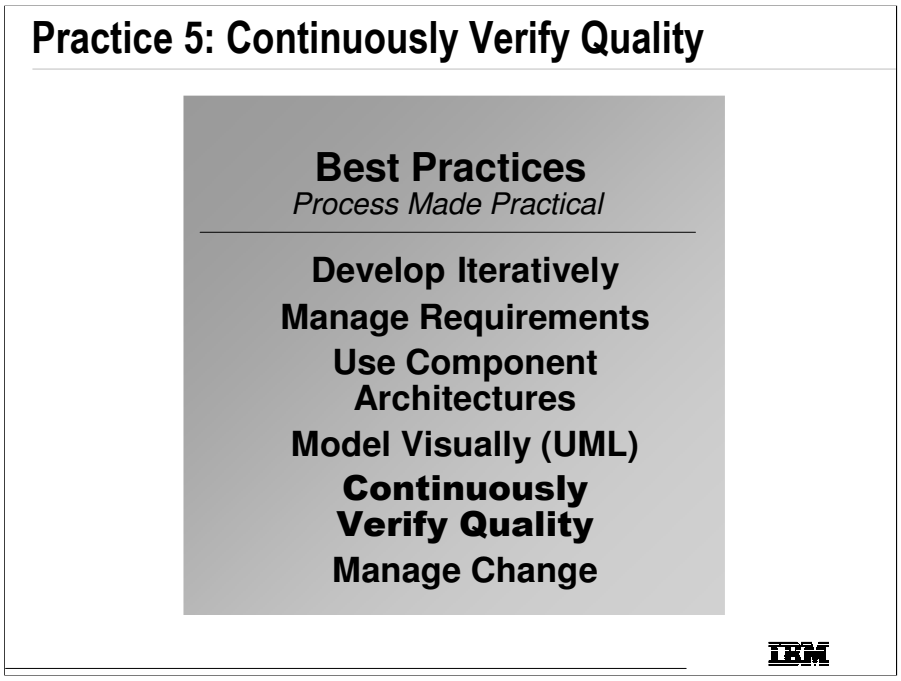
- Sample UML-to-code transforms for EJB, Java, and C++
- Selective language to UML harvesting

- C/C++ editors and build management
- Compiler and debugger integration
- UML code editors

"WSAD v6"

- JSF, SDO, Struts
- Java GUI editor
- Web diagram editor
- Site designer
- Web Services development tools
- Database editing tools
- EGL
- EJB development tools
- UML code editors for EJB, Java, and Data
- Static Analysis
- Runtime Analysis
- Component test automation
- Portal/Portlet development tools

IBM

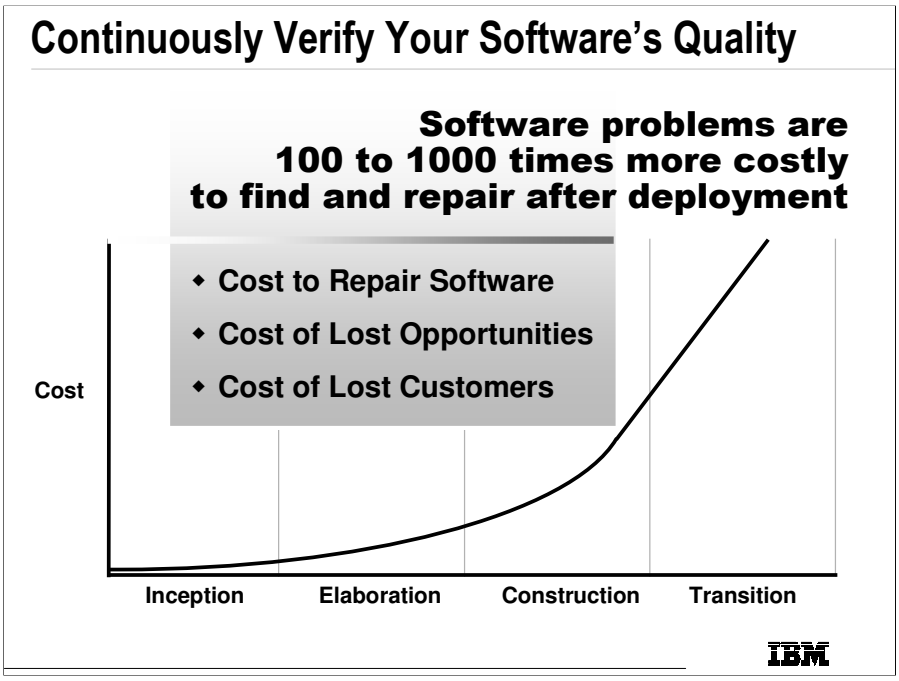


Quality, as used within RUP, is defined as "The characteristic of having demonstrated the achievement of producing a product which meets or exceeds agreed-upon requirements, as measured by agreed-upon measures and criteria, and is produced by an agreed upon process." Given this definition, achieving quality is not simply "meeting requirements" or producing a product that meets user needs and expectations. Quality also includes identifying the measures and criteria (to demonstrate the achievement of quality), and the implementation of a process to ensure that the resulting product has achieved the desired degree of quality (and can be repeated and managed).

In many organizations, software testing accounts for 30 percent to 50 percent of software development costs. Yet most people believe that software is not well-tested before it is delivered. This contradiction is rooted in two clear facts. First, testing software is enormously difficult. The different ways a particular program can behave are almost infinite. Second, testing is typically done without a clear methodology and without the required automation or tool support. While the complexity of software makes complete testing an impossible goal, a well-conceived methodology and use of state-of-the-art tools can greatly improve the productivity and effectiveness of the software testing.

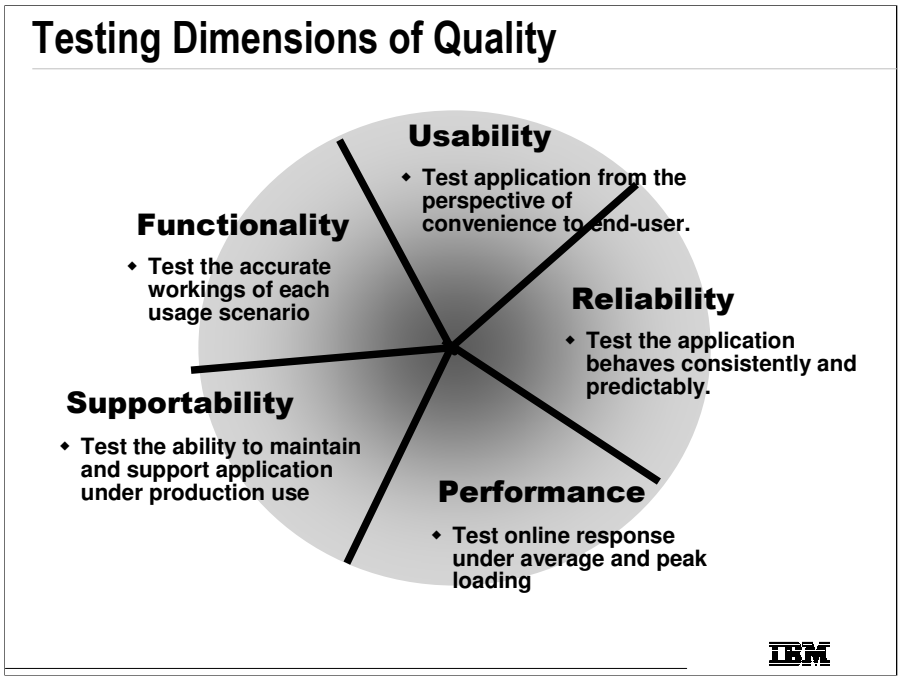
Many people remember Barry Boehm's groundbreaking work in Software Economics where he quantified the relative expense to fix a bug at different times in the development lifecycle. Be cautious, however, since his work was based on the waterfall model, not an iterative development model.

The iterative model fundamentally changes how and when we test.



This principle is driven by a fundamental and well-known property of software development: it's a lot less expensive to correct defects during development than to correct them after deployment.

- Tests for key scenarios ensure that all requirements are properly implemented
- Poor application performance hurts as much as poor reliability
- Verify software reliability—memory leaks, bottlenecks
- Test every iteration—automate test!



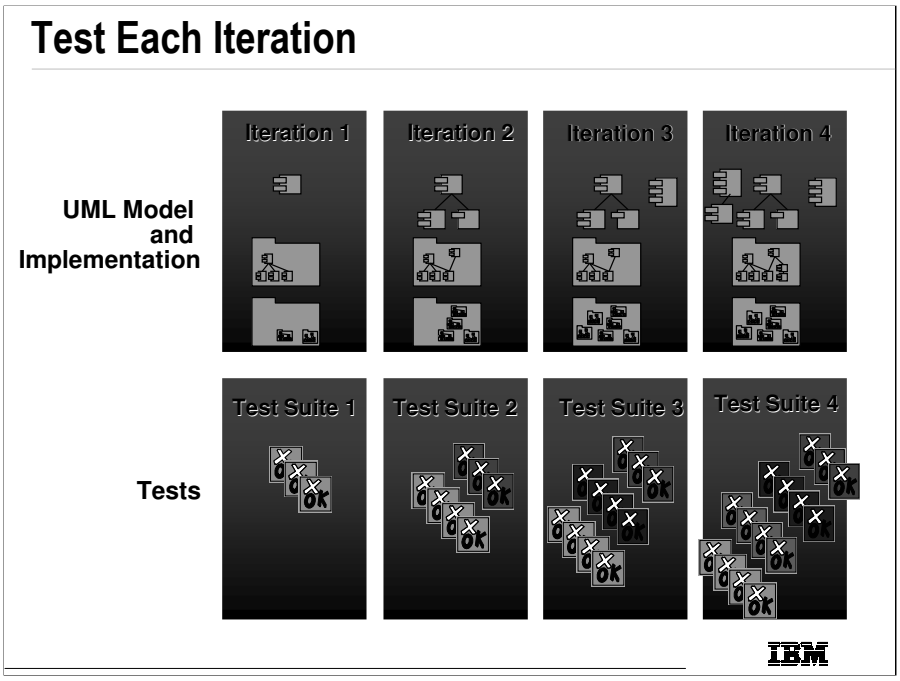
Functional testing verifies that a system executes the required use-case scenarios as intended. Functional tests may include the testing of features, usage scenarios and security.

Usability testing evaluates the application from the user's perspective. Usability tests focus on human factors, aesthetics, consistency in the user interface, online and context-sensitive help, wizards and agents, user documentation, and training materials.

Reliability testing verifies that the application performs reliably and is not prone to failures during execution (crashes, hangs, memory leaks). Effective reliability testing requires specialized tools. Reliability tests include integrity, structure, stress, contention and volume tests.

Performance testing checks that the target system works functionally and reliably under production load. Performance tests include benchmark tests, load tests, and performance profile tests.

Supportability testing verifies that the application can be deployed as intended. Supportability tests include installation and configuration tests.



In each iteration, automated tests are created that test the requirements addressed in that iteration. As new requirements are added in subsequent iterations, new tests are generated and run. At times, a requirement may be changed in a later iteration. In that case, the tests associated with the changed requirement may be modified or simply regenerated by an automated tool.

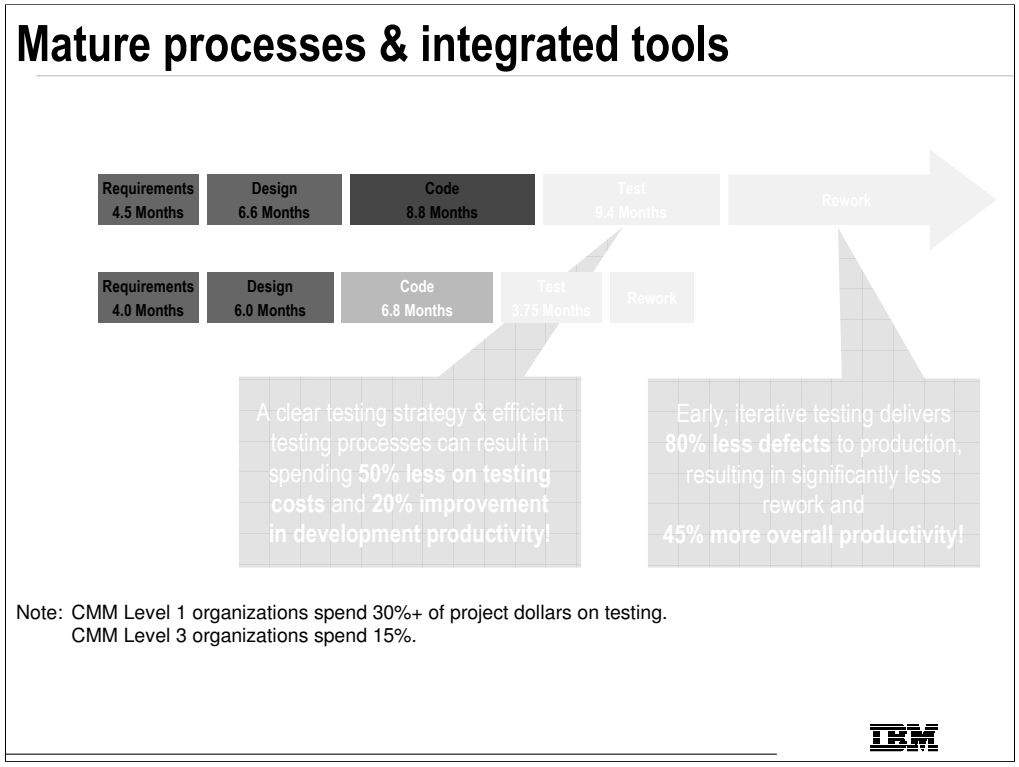
Incrementally Build Test Automation

- Incrementally build test automation to:
  - Detect defects early
  - Minimize up-front investments
- Generating test code directly from the design models:
  - Saves time
  - Provides incentives for early testing
  - Increases the quality of testing by minimizing the number of bugs in the test software

Making the right design decisions can greatly improve our ability to automate testing.

IBM

Automated testing has been a key area of focus for, among others, the agile community. The aim is to automate testing of all code and tests are written before the code is written (test-first design).



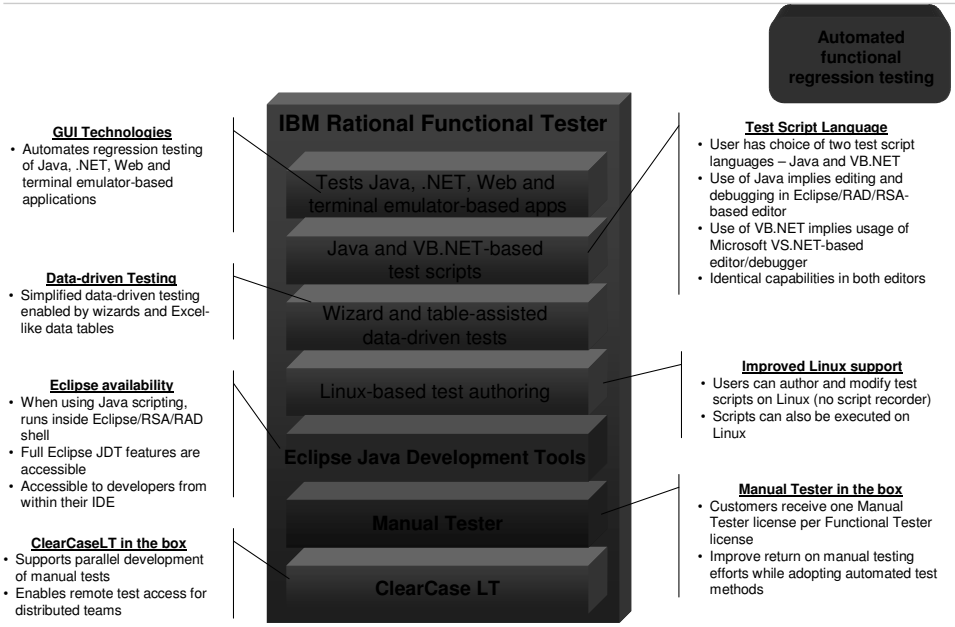
SEI (Software Engineering Institute) claims that CMMi level 3 projects will show around 20% productivity increase. This requires institutionalization of common processes, methods and tools across an organization with a common line of business, plus objective metrics, plus maturity in best practices gets an organization to improving return on investment from project to project.

This benefit is mostly visible during the testing activities : defining a clear testing strategy and implementing an efficient testing process can drive to a 50% decrease of the global testing costs.

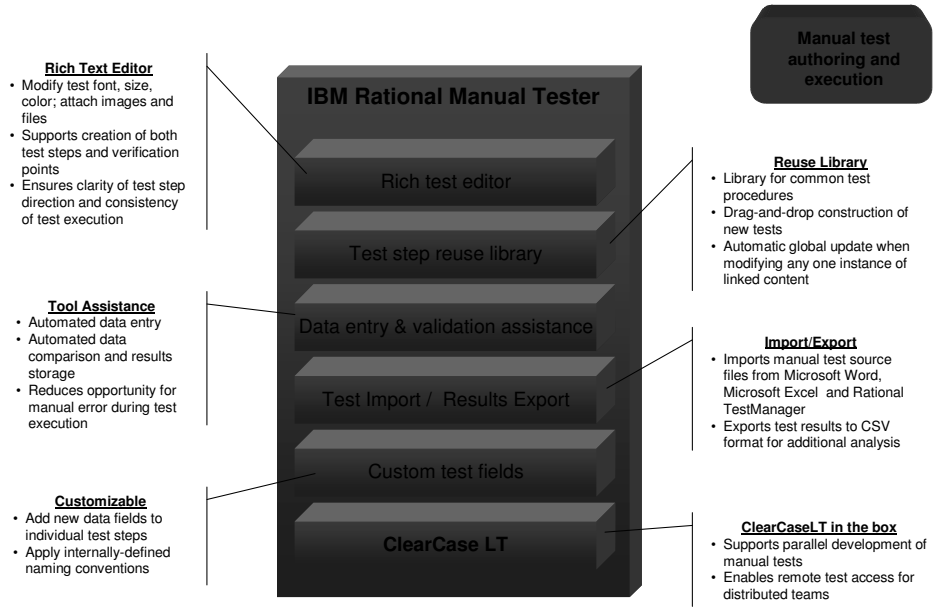
More, with earlier and smarter tests, the applications are delivered with 80% bug less, reducing the rework (working on residual bugs) and the overall productivity is increased by 45%.

That's why the testing strategy must be the concern of all the organization, at all levels.

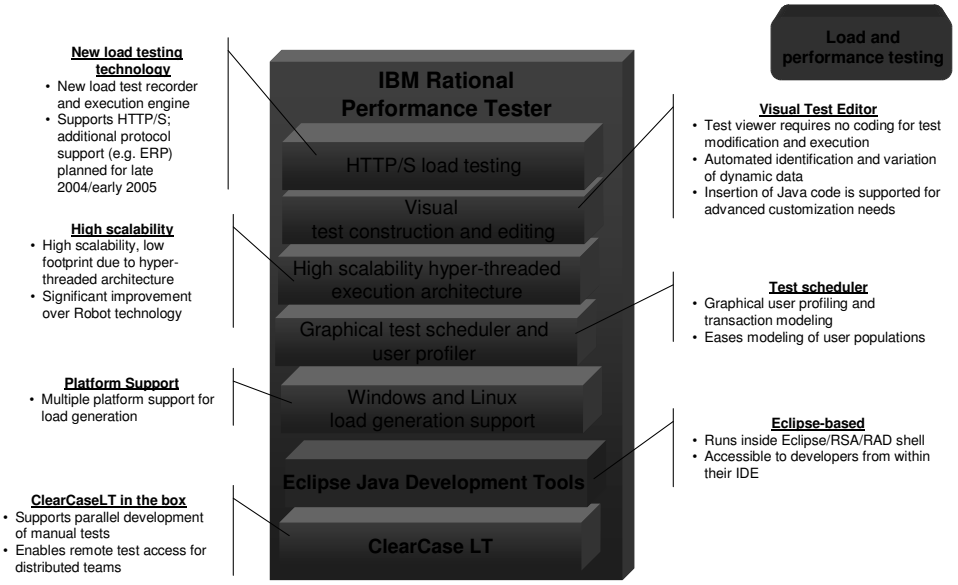
IBM Rational Functional Tester Overview

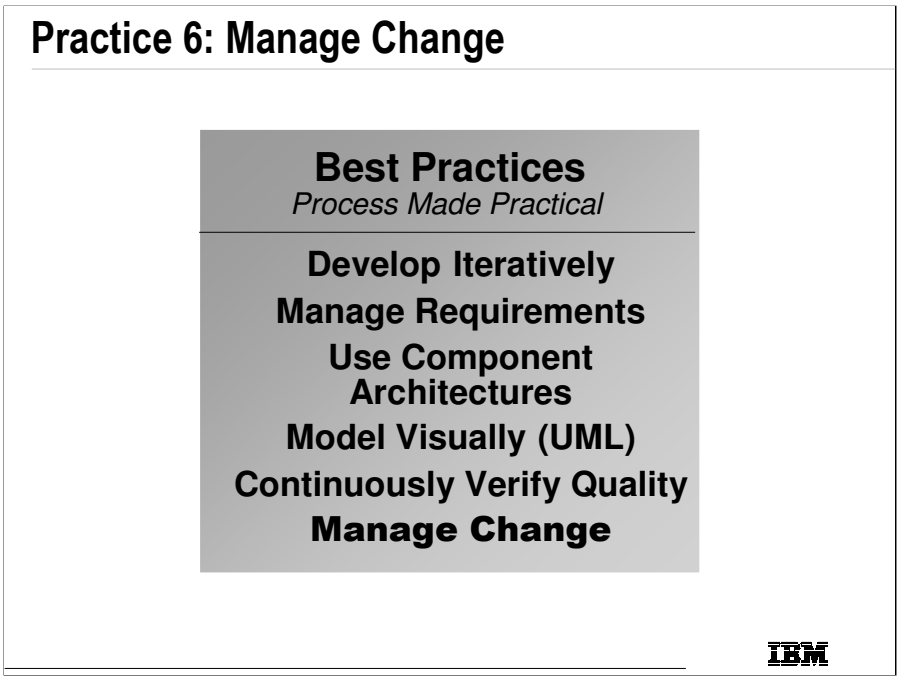


IBM Rational Manual Tester Overview



IBM Rational Performance Tester Overview





As we indicated earlier, we cannot stop change from being introduced into our project. However, we must control how and when changes are introduced into project artifacts, and who introduces the changes. We also must synchronize change across development teams and locations.

Unified Change Management (UCM) is Rational Software's approach to managing change in software system development, from requirements to release.

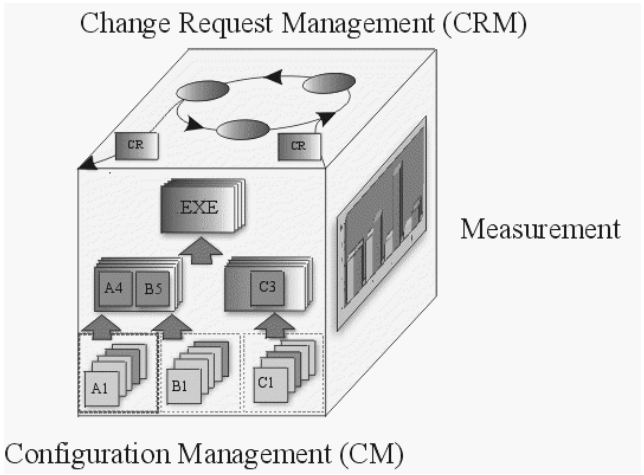
Embrace and Manage Change

- Today's applications are too complex for requirements, design, implementation, and testing to align perfectly the first time through.
- Most effective application development methods embrace the inevitability of change.
- The iterative approach provides us with the opportunity to implement those changes incrementally.

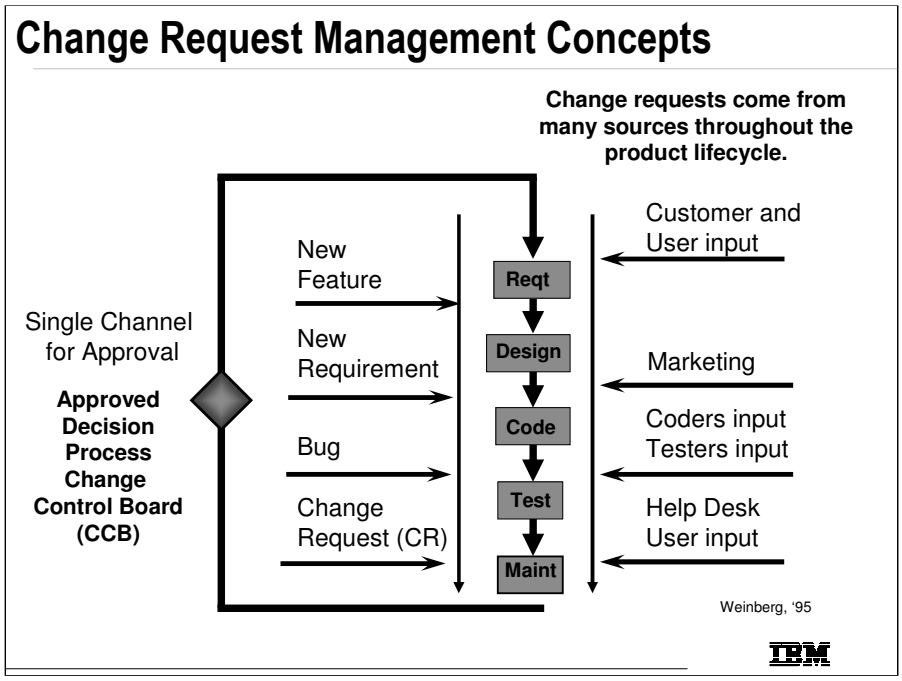
IBM

Change needs to be managed by having the processes and tools in place to avoid hindering creativity.

The Configuration and Change Management (CCM) Cube



The CCM cube summarizes the interdependence of these functions. The three aspects are closely connected.



Change Request Management (CRM) addresses the organizational infrastructure required to assess the cost and schedule impact of a requested change to the existing product. CRM addresses the workings of a Change Review Team or Change Control Board.

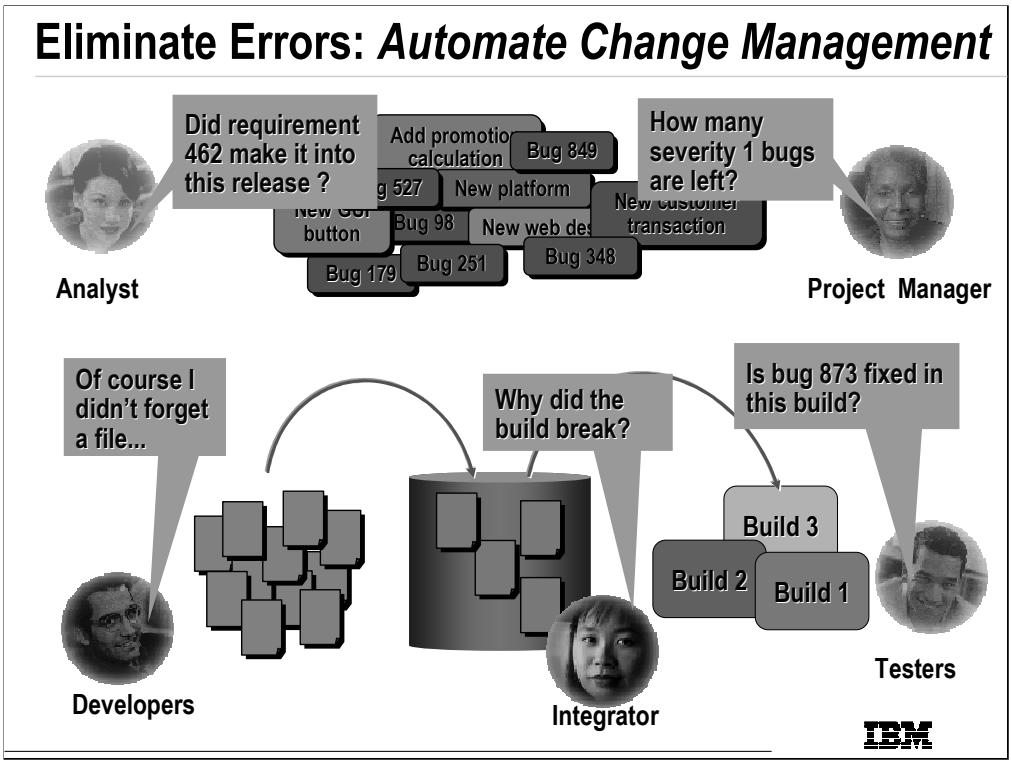
Configuration Status Accounting (Measurement) is used to describe the "state" of the product based on the type, number, rate, and severity of defects found and fixed during the course of product development. Metrics derived under this aspect, either through audits or raw data, are useful in determining the overall completeness of the project.

Configuration Management (CM) describes the product structure and identifies its constituent configuration items that are treated as single versionable entities in the configuration management process. CM deals with defining configurations, building and labeling, and collecting versioned artifacts into constituent sets and maintaining traceability between these versions.

Change Tracking describes what is done to components for what reason and at what time. It serves as the history and rationale of changes. It is quite separate from assessing the impact of proposed changes as described under Change Request Management.

Version Selection is to ensure that the right versions of configuration items are selected for change or implementation. Version selection relies on a solid foundation of "configuration identification."

Software Manufacture covers the need to automate the steps to compile, test, and package software for distribution.



"What's the big deal with change?

The software team usually experiences it as a blizzard of requests -- to make enhancements, fix bugs, you name it.

"If you think about how artifacts move through the software development process, you see that each member of the team experiences all these changes differently.

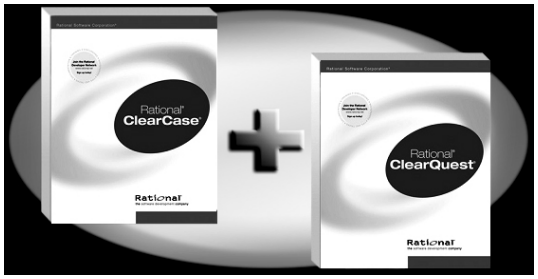
Project managers try to assess the project status and analysts want to know what features are arriving in builds.

Developers change a huge collection of different files, and they have to assure they have them all checked in for the builds.

The integrator, or build manager has to figure out why builds break and testers who receive new builds on a regular basis need to know what's new and needs testing."

With an automated change management application, the entire team cab become more aware of changes for a given iteration and the team cab

UCM Tools: ClearCase and ClearQuest



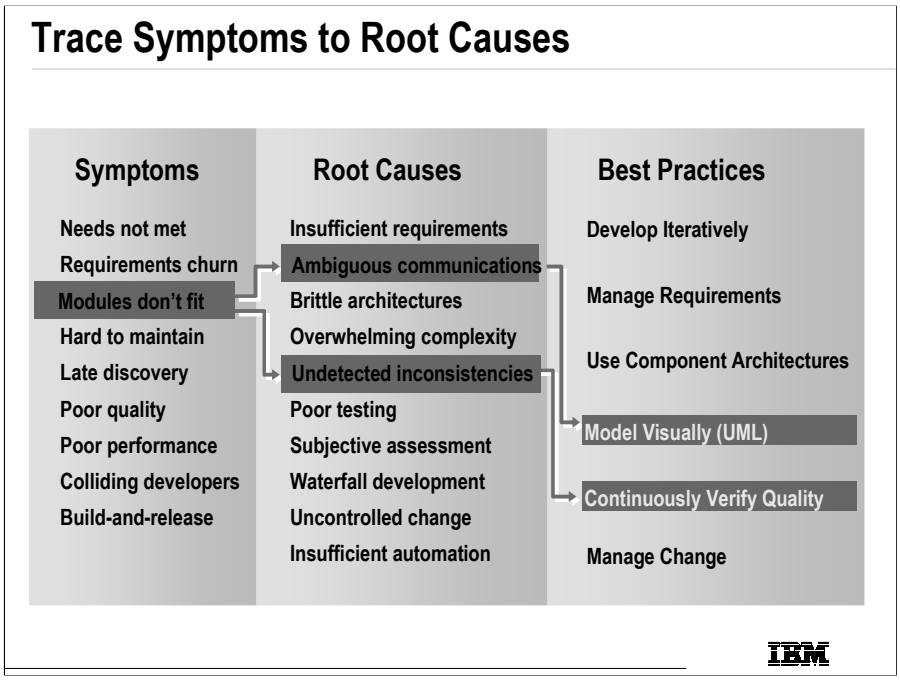
- You can relate the changes you make to artifacts to the change requests
- You can quickly access the right versions of artifacts
- You can work in a parallel or serial development environment
- You work in isolation without being affected by others' changes
- You can track, manage, and report activities

IBM

Rational ClearCase is a robust software artifact management tool that provides automated parallel development.

Rational ClearQuest is a flexible, customizable defect and change-tracking application.

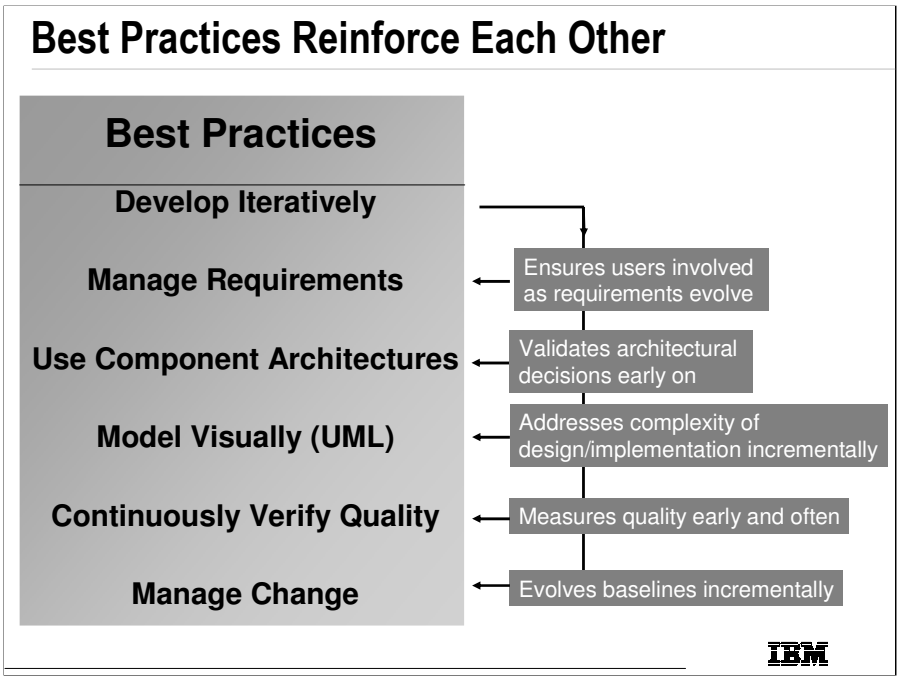
ClearCase UCM can be used independently of ClearQuest. You will have artifact management, but no defect and change-tracking capabilities.



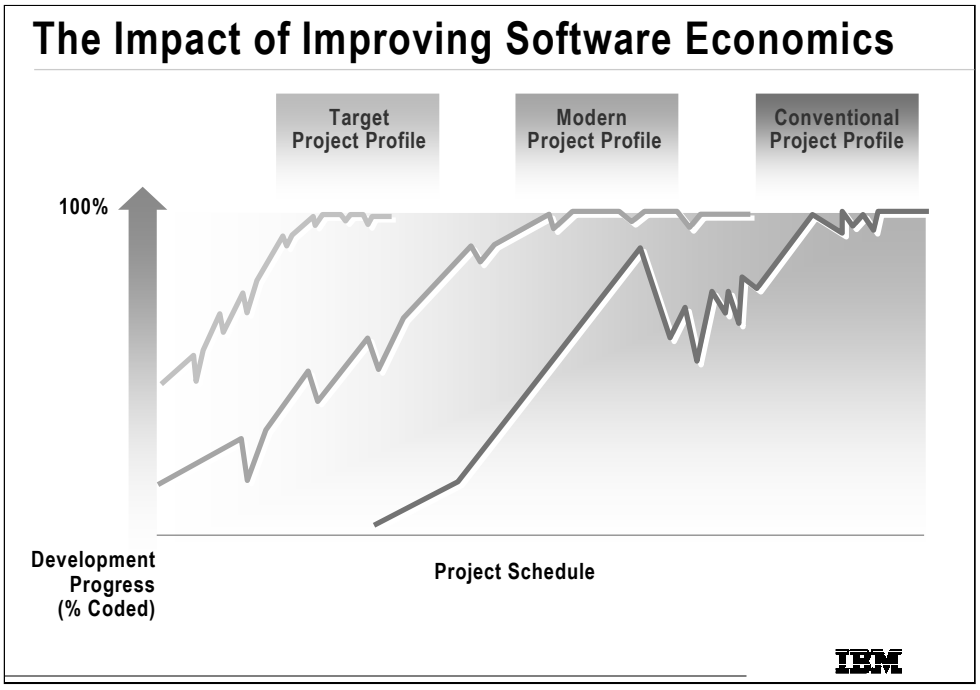
Treat these root causes, and you'll eliminate the symptoms. Eliminate the symptoms, and you'll be in a much better position to develop quality software in a repeatable and predictable fashion.

Best practices are a set of commercially proven approaches to software development, which, when used in combination, strike at the root causes of software development problems. These are so-called "best practices," not so much because we can precisely quantify their value, but rather because they are observed to be commonly used in the industry by successful organizations.

The best practices are harvested from thousands of customers on thousands of projects and from industry experts.



In the case of our six Best Practices, the whole is much greater than the sum of the parts. Each of the Best Practices reinforces and, in some cases, enables the others. The slide shows just one example: how iterative development leverages the other five Best Practices. However, each of the other five practices also enhances iterative development. For example, iterative development done without adequate requirements management can easily fail to converge on a solution. Requirements change at will, users can't agree, and the iterations can go on forever. When requirements are managed, this is less likely to happen. Changes to requirements are visible, and the impact on the development process is assessed before they are accepted. Convergence on a stable set of requirements is assured. Similarly, each pair of Best Practices provides mutual support. Hence, although it is possible to use one Best Practice without the others, it is not recommended, since the resulting benefits will be significantly decreased.



A modern process framework attacks the primary sources of the diseconomy of scale inherent in the conventional software process. This graphic illustrates the next generation of software project performance by depicting the development progress versus time, where progress is defined as percent coded, i.e., demonstrable in its target form.

The goal of this presentation has been to explain how to move onto the upper shaded region, with a modern process supported by an advanced, fully integrated environment and a component-based architecture. Organizations that succeed should be capable of deploying software products that are constructed largely out of existing components in substantially reduced time, with substantially less development resources and substantially improved quality

Today, about 60% of the world's software organization still operate on conventional project profiles. About 30% have transitioned to modern project profiles and perhaps 10% are already achieving improved software economics and experiencing results similar to the target project profiles.

For more information, see Royce, pages 247-253.

Software Economics

$$\text{Project Cost} = (\text{Complexity})^{(\text{Process})} * (\text{Team}) * (\text{Tools})$$

Where:

- Project Cost = Effort or time
  - Complexity = Volume of human-generated artifacts
  - Process = Methods, notations, maturity
  - Team = Skill set, experience, motivation
  - Tools = Software process automation

IBM

COCOMO II formula for estimating project cost

Lower Cost

- Reducing costs is predominantly achieved through improved software economics:
  - Reduce Complexity
  - Improve the Process
  - Increase Team Efficiency
  - More SDLC Automation

IBM

Reduce Complexity (Build less stuff)

- Reduce the size of the problem
  - Manage scope of the requirements
- Reduce complexity by raising the level of abstraction
  - Create component-based architectures
    - Modularity, layers, interfaces
    - Reuse framework
  - Employ visual modelling (UML)
- Reduce the proportion of hand written code
  - Employ more commercial components & reuse other components
  - Automated code generators

IBM

Improve the Process (Build it more efficiently)

- Reduce scrap and rework
  - Controlled iterative & incremental process
- Attack significant risks early
  - Continuously assessing, prioritizing, and attacking top risks
  - Build the architecture first
- Employ best practices
  - Iterative Development, Manage Requirements, Component Architectures, Visual Modelling (UML), Continuous Verification of Quality, Manage Change
  - Right-size the process

IBM

Increase Team Efficiency (Better collaboration)

- Coordination & communication among team members
  - Smaller cross-functional teams
  - Team organized based on architecture
  - Central location for project artifacts
  - UML
- Increase individual proficiency
  - Motivation (ownership of plans, reward based on results)
  - Teamwork (shared objectives, access to expertise)
  - Skills (education & mentoring in technology, best practices & tools)
  - Experience (domain & software development experience)

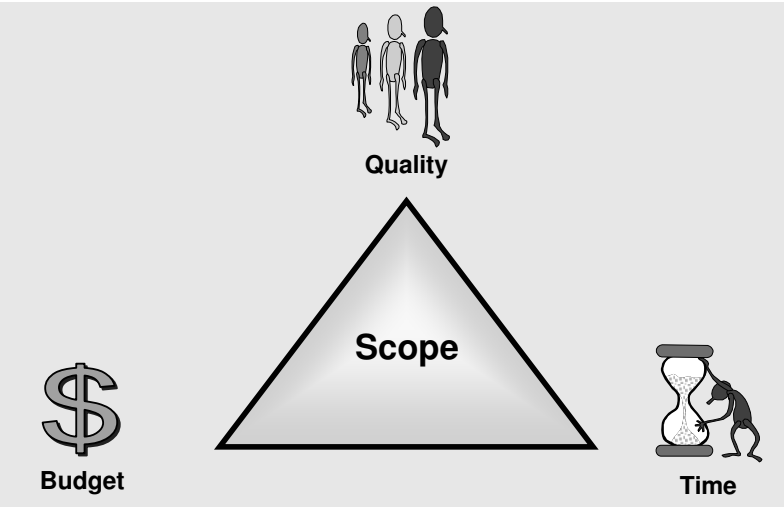
IBM

Increase Automation (with tooling)

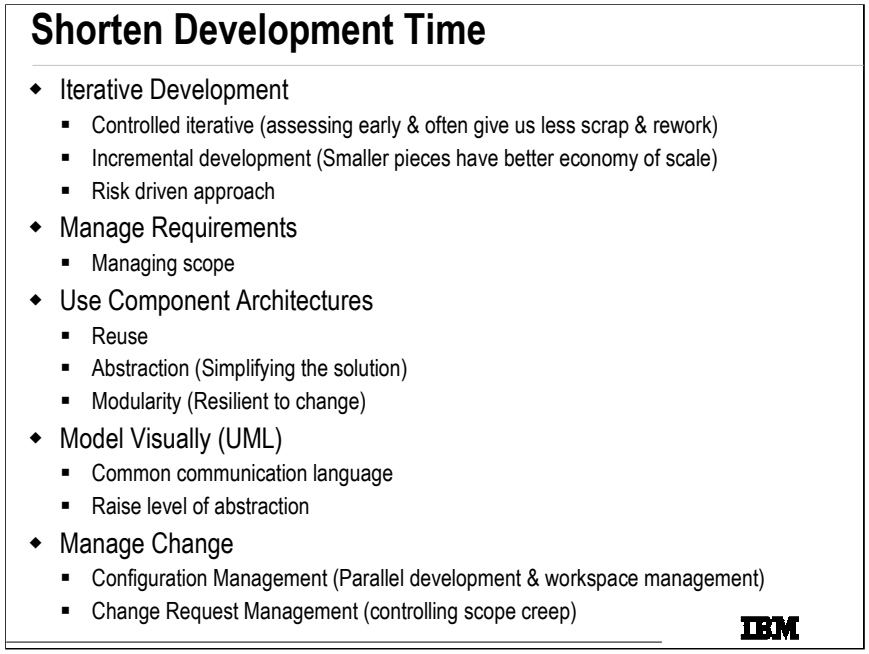
- Reduce error-prone manual tasks
  - Utilize tools for the entire lifecycle, including code generation & round trip engineering
- Increase communication
  - Team unifying platform in a common development environment
- Better project management
  - Tools automate assignment of activities, managing change, monitoring of progress, & measurement of quality
- Automate best practices
  - Tools support modern software development best practices

IBM

Time-Cost-Quality Tradeoff



IBM

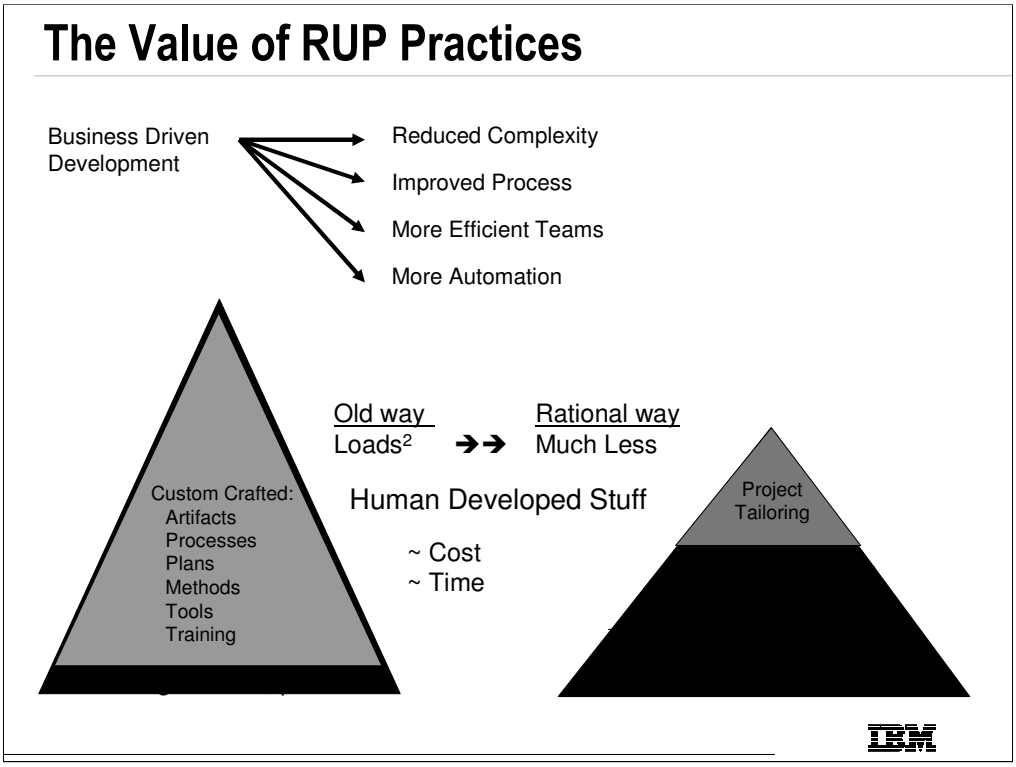


Controlled Iterative and Incremental gives us two things: 1) With Controlled Iterative development we assess earlier and often so we get less scrap and rework. 2) In Incremental Development we end up working on smaller chunks which are less complex (less dependencies between all the pieces) so development goes quicker

Increase Quality

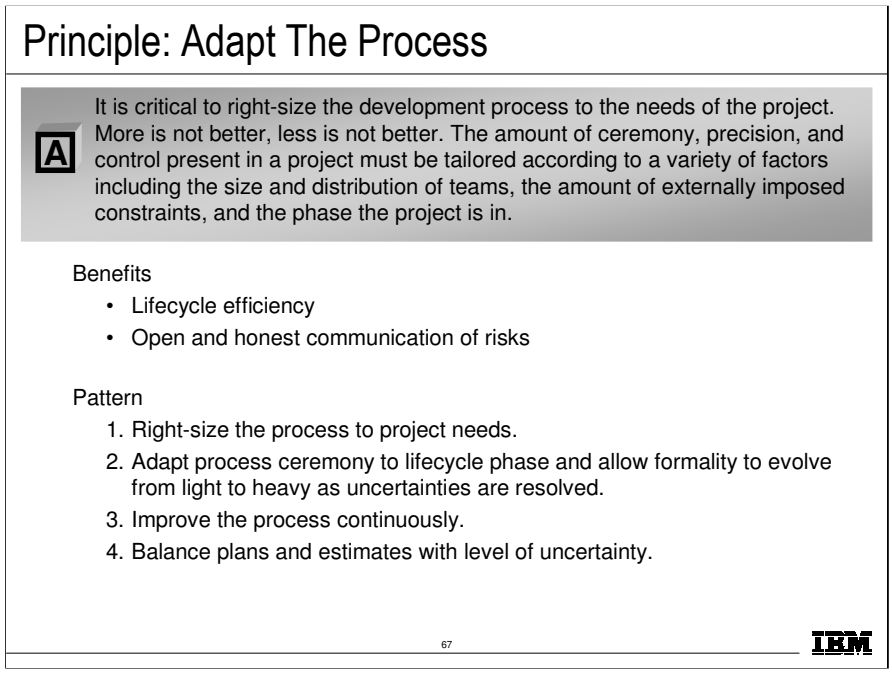
- Iterative Development
  - Controlled iterative (Based on prioritized requirements)
  - Demonstration based approach
  - Continuous integration
- Manage Requirements
  - Requirements best practices
  - Traceability (find holes)
  - Manage change (ensuing the right changes are made)
- Use Component Architectures
  - Modularity (Resilient to change)
- Continuously Verify Quality
  - Different types of testing (regression, performance, usability, etc.)
  - Testing early, often, and base on priority (Iteratively)
- Manage Change
  - Unified Change Management including Configuration Management (build management) & Change Request Management
  - Metrics

IBM


The Value of RUP Practices
Business Driven Development
Reduced Complexity
Improved Process
More Efficient Teams
More Automation
Old way
Loads²
Rational way
Much Less
Human Developed Stuff
~ Cost
~ Time
Custom Crafted:
Artifacts
Processes
Plans
Methods
Tools
Training
Project Tailoring
IBM

Why is Process Important?

- Consistency
  - Enable clear, consistent communication for all team members
  - Help team members understand their responsibilities and their relationship with other team members
- Predictability
  - Help identify what resources are needed and when
  - “Compare to” baseline for addressing bottlenecks and failure points
  - Enables metric development to support future planning and estimation
  - Defines decision points to reduce surprises
- Quality
  - Focus on risk reduction
  - Roadmap of how value will be delivered to the customer



The most recognizable "anti-patterns" or behaviors contrary to this principle that can harm software development projects:

Always see more process and more detailed upfront planning as better:

- Force early estimates and stick to those estimates.
- Develop precise plans and manage project by tracking against a static plan.
- Always use the same degree of process throughout the lifecycle.

Right-Size the Process to Project Needs

- More process is not necessarily better:
  - Artifacts and detailed documentation
  - Models that need to be synchronized
  - Formal reviews
- For smaller projects with co-located teams and known technology, the process should be lightweight.
- As a project grows in size, the process needs to become more disciplined.

68

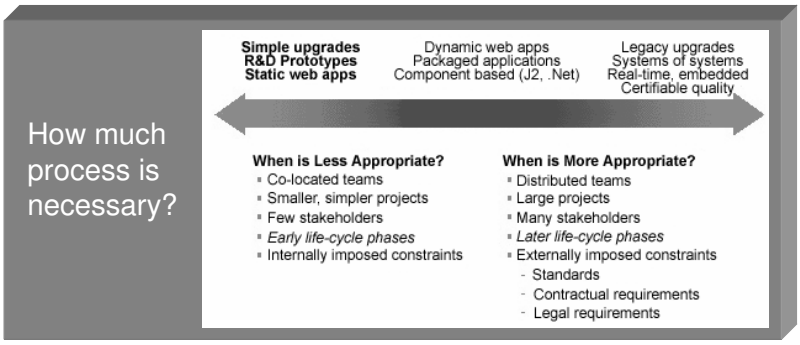
IBM

More process, such as usage of more artifacts, production of more detailed documentation, development and maintenance of more models that need to be synchronized, and more formal reviews, is not necessarily better.

Rather, we need to right-size the process to project needs. As a project grows in size, becomes more distributed, uses more complex technology, has larger number of stakeholders, and needs to adhere to more stringent compliance standards, the process needs to become more disciplined. But, for smaller projects with co-located teams and known technology, the process should be more lightweight.

Factors that affect process right-sizing

- Many factors steer how disciplined a process you need, including:
  - project size
  - team distributions
  - complexity of technology
  - number of stakeholders
  - compliance requirements
  - the phase of the project lifecycle.



69

IBM

Ceremony, Plans and Estimates, and Improvements

- Early in a project
  - Minimize ceremony in the process to enable more creativity
  - Focus on the big-picture in planning and estimating - precision is not attainable
  - Aim at driving out uncertainty
- Later in a project
  - Increase ceremony in the process to enable more control
  - Increase precision in planning
- Continuously improve the process
  - Leverage process experiences after each iteration and at project end to improve the process
  - Encourage all team members to look for opportunities to improve

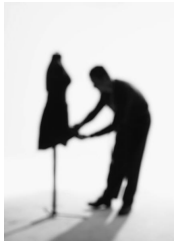
70

IBM

Why Tailor RUP?

- All projects are not the same
- Projects evolve
- Home-grown processes take time to develop and maintain

- You might want tailor your Web site by:
  - Reducing it
  - Enhancing it with additions
  - Modifying its base content



71

IBM® Rational® Method Composer 7.0 Product

- IBM Rational Method Composer 7.0 ← Tailoring functionality
- Method Library with the following plug-ins
  - Rational Unified Process
  - Base Concepts
  - RUP Formal Resources
  - RUP Informal Resources
  - Business Modeling
  - Service-Oriented Architecture
  - RUP for J2EE
  - Rational Software Architect
  - Legacy Evolution
  - Rational Application Development
- Published RUP Configurations
  - RUP for Large Projects
  - RUP for Small Projects

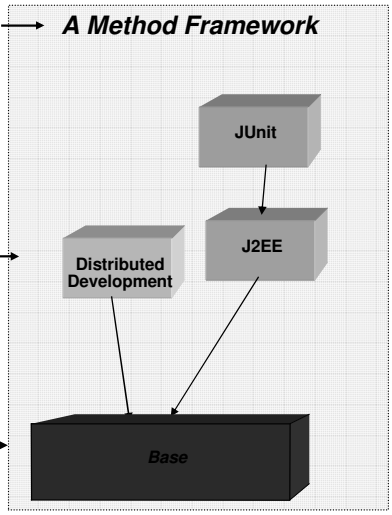
72

What Is a Method Framework?

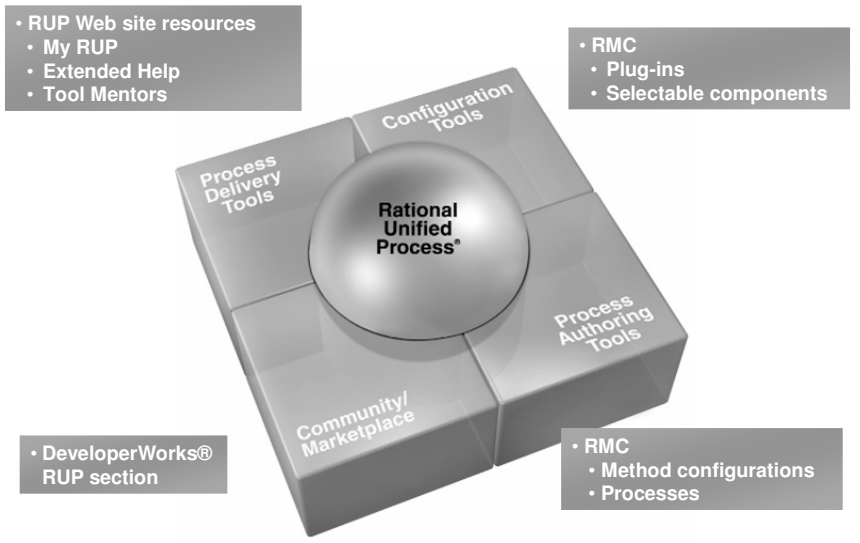
A framework is based on a common set of principles
- All pieces within a framework fits together
- Enables rapid assembly of a diverse set of processes
- A process framework can contain sub frameworks

Contains any number of extension plug-ins
- Adds additional processes or modifies existing processes
- Adds additional method content

Contains a base of standard content
- Base content (principles, practices, roles, work products, tasks, …)
- Optionally a base process (lifecycle)



The Rational Method Composer (RMC) Platform



74

Two levels of Tailoring

- Process tailoring can happen at two levels:
  - At the organizational level, where a common process is configured, to be used organization-wide. Organization-level tailoring takes into consideration application domain, reuse practices, and core technologies mastered by the company.
  - At the project level, where a process is configured for a specific project. Project-level tailoring level takes into consideration the size of the project, the reuse of company assets, the type of development lifecycle.

75

Tailoring Approach

- No matter what level in the organization the process is being tailored for, the overall approach to tailoring the RUP is the same:
  - Identify the scope of the tailoring effort
    - Identify method content to be used
    - Identify existing method assets that could be used
    - Select the content to be tailored (processes, roles, tasks, work products, guidance, and so on)
  - Select the level of tailoring to be performed
    - There are different levels at which RUP can be tailored, each with its own costs and benefits.
  - Tailor the process
    - Tailor the identified parts of RUP using the selected tailoring level

76

Key Steps for Tailoring RUP

- No matter what level of tailoring you choose, tailoring the RUP generally involves these key steps:
  - Develop the method elements
    - Developing new content and refining existing content
  - Configure the method content
    - Decide what content to include and what content to exclude
  - Develop the process for the configuration
    - Select a type of development lifecycle (for example, waterfall versus iterative) and define a process that fits the exact needs of the organization or project.
  - Make the process available
    - Publish the configuration as a process website, and export the process to a project planning tool.

77

IBM

Tailoring the process is just one part of implementing a process for a project. When the process has been tailored, the project manager instantiates and executes it for the given project.

An "Instantiated" process is an enactable project, iteration, or activity plan (it includes actual activities and work products for an actual project). Such instantiation is done as part of project planning.

Levels of Tailoring Available for RUP

- Level 1
  - Document the tailored process in an external document that refers to an underlying process, as well as customized method assets.
- Level 2
  - Personalize an existing process Web site using My RUP and external documents.
- Level 3
  - Configure a process Web site from existing method content using Rational Method Composer.
- Level 4
  - Add Guidance to the existing method framework using Rational Method Composer
- Level 5
  - Develop a new Delivery Process using Rational Method Composer
- Level 6
  - Extend the existing method framework with new method content using Rational Method Composer

78

IBM

Here are some considerations for the levels of tailoring:

Level 1 - This level is beneficial if the process you need to tailor cannot be modified for some reason (for example, it is tightly controlled for auditing purposes) or you do not have access to Rational Method Composer. Tailoring at this level only affects the presentation of the Web site, not its underlying content.

Level 2 - This level is intended for use by individuals on a personal copy of the process Web site, and is generally not the recommended approach for tailoring the process for an entire project or organization. However, personalization might be a good compromise in those cases where you want to do some minor refinement of the presentation of the Web site and you do not have access to Rational Method Composer. Tailoring at this level only affects the presentation of the Web site, not its underlying content.

Level 3 - This level is sometimes referred to as Method Configuration development.

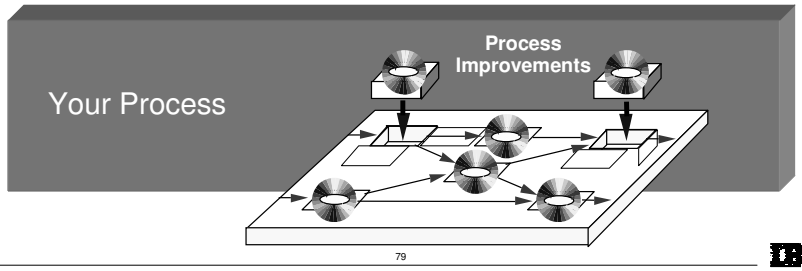
Level 4 - This level is sometimes referred to as "thin" Method Plug-ins development. Thin plug-ins only add Guidance (for example, Concepts, Guidelines, Templates, Examples, Tool Mentors). Thin plug-ins are a mechanism that organizations can use to package their organizational assets, such as work product templates, guidelines, examples and other reusable assets for consumption in the individual project. The creation of thin plug-ins is done at very low cost and, as such, is highly applicable to any sized organization and can usually be justified within the budget of one single project. In addition, the creation of thin plug-ins does not affect the processes (Capability Patterns and Delivery Processes) included in a configuration, because no roles, tasks, or work products are added or refined.

Level 5 - This level is sometimes referred to as Delivery Process development.

Level 6 - This level is sometimes referred to as "structural" plug-in development. A structural plug-in is a plug-in that extends the RUP by adding or refining Roles, Tasks, and/or Work Products.

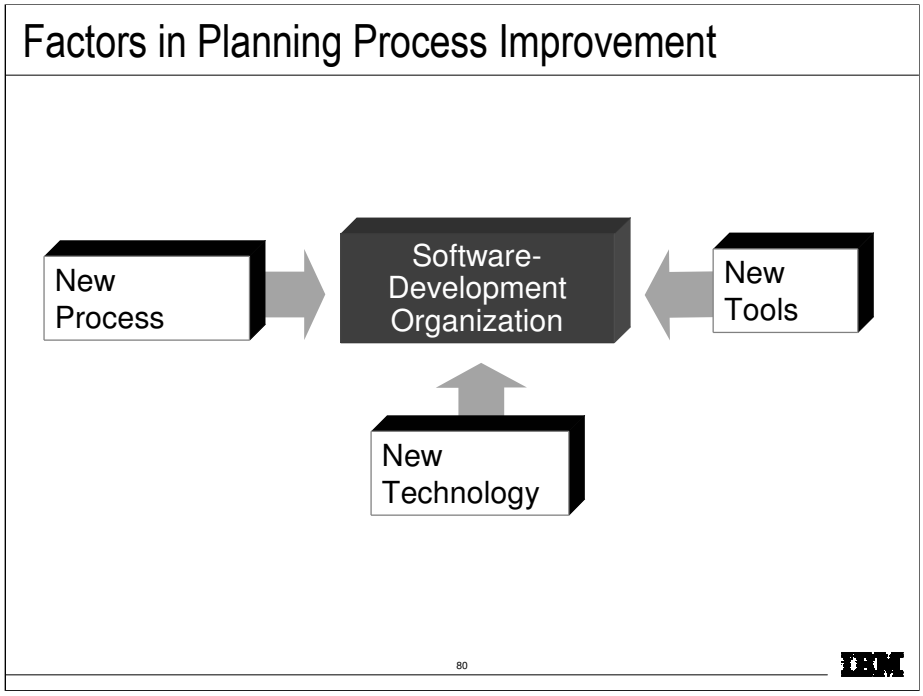
What Does It Mean To Implement RUP?

- Implementing RUP means improving selected parts of your existing process.
- Process improvements are selected by considering (for example):
  - Current pains and problems, for example, delayed projects
  - Mandates, for example, Reach CMMI Level 2
  - Certification requirements, for example, ISO 9000
  - Efficiency improvements necessary to compete, for example, incomplete testing due to schedule compression



79

IBM



You rarely introduce a new process such as RUP without facing several other new factors:

New Technology

- Programming language (Java, C++, VisualBasic)
- Component-based development
- Internet
- Client-Server Architectures
- Graphical User Interfaces

New Tools

- Requirements Management
- Modeling
- Programming
- Configuration Management
- Test

New Process and Best Practices

- Iterative Development
- Component-Based Architecture
- Manage Requirements
- Guidelines, and so on

Process, tools, and technology are not separate issues. An organization needs to implement process, technology, and tools together.

Practices for Process Improvement

- RUP guidance for process improvement:
  - See the content of the Environment discipline
- Practices recommended in the *Concept: Environment Practices*
  - Assess the Project and the Organization
  - Implement Process and Tools Incrementally
  - Manage and Plan
  - Use Mentors
  - Distribute Process Ownership
  - Think Return-On-Investment
  - Keep People Informed and Involved
  - Educate People

81

- **Assess the Project and the Organization:** Assess the current state of the project and the organization to better understand what parts of the environment you should improve.
- **Implement Process and Tools Incrementally:** By implementing the environment incrementally, it is possible to focus on a subset of the environment, which increases the probability of success.
- **Manage and Plan:** Manage and plan the environment tasks just as you would with all other tasks in the software development project.
- **Use Mentors:** Use mentors to introduce a new process in a project.
- **Distribute Process Ownership:** Distribute the ownership for the process among the people on the project because they are more likely to adopt and learn the new process faster.
- **Think Return-On-Investment:** Focus on those things that will pay back more than the investment.
- **Keep People Informed and Involved:** The greatest threat to any change in an organization is peoples' attitudes towards the change.
- **Educate People:** They need to understand both the new process and how to use the new tools.


Thank
You
IBM