International standards, approaches and frameworks relevant to Software Quality Management and Software Process Improvement

To help organizations managing software quality and improving software processes several standards, models, approaches and frameworks have been developed during the last decades. The most widely known and recognized of them are presented in this document.

- Capability Maturity Model (CMM)
- CMM Integration (CMMI)
- Personal Software Process (PSP) and Team Software Process (TSP)
- ISO 9000 standards family
- TickIT
- ISO/IEC TR 15504 Information Technology - Software Process Assessment (SPICE)
- ISO/IEC 12207 Information Technology - Software Life-Cycle Processes
- BOOSTRAP
- Rational Unified Process

CMM

Publication Date: Version 1.1 - February 1993

Description: The Capability Maturity Model for Software (SW-CMM or CMM) is a model used by organizations for appraising the maturity of their software processes and for identifying practices that will increase the maturity of those processes. It was developed by the Software Engineering Institute, in cooperation with industry representatives. The Software CMM has become a de facto standard for assessing and improving software processes. Through the SW-CMM, the SEI and community have put in place an effective means for modeling, defining, and measuring the maturity of the processes used by software professionals.

The Capability Maturity Model for Software describes the principles and practices underlying software process maturity and is intended to help software organizations improve the maturity of their software processes in terms of an evolutionary path from ad hoc, chaotic processes to mature, disciplined software processes. The CMM is organized into five maturity levels that consist of Key Process Areas:

Level	Focus	Key Process Areas
Level 5 Optimizing	Continuous improvement	Process Change Management Technology Change Management Defect Prevention
Level 4 Managed	Product and process quality	Software Quality Management Quantitative Process Management
Level 3 Defined	Engineering process	Organization Process Focus Organization Process Definition Peer Reviews Training Program Intergroup Coordination Software Product Engineering Integrated Software Management
Level 2 Repeatable	Project management	Requirements Management Software Project Planning Software Project Tracking and Oversight Software Subcontract Management Software Quality Assurance Software Configuration Management
Level 1 Initial	Heroes	No KPA at this time

1) Initial. The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics.

2) Repeatable. Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

3) Defined. The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.

4) Managed. Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.

5) Optimizing. Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

Predictability, effectiveness, and control of an organization's software processes are believed to improve as the organization moves up these five levels. While not rigorous, the empirical evidence to date supports this belief.

Except for Level 1, each maturity level is decomposed into several key process areas that indicate the areas an organization should focus on to improve its software process.

The key process areas at Level 2 focus on the software project's concerns related to establishing basic project management controls. They are Requirements Management, Software Project Planning, Software Project Tracking and Oversight, Software Subcontract Management, Software Quality Assurance, and Software Configuration Management.

The key process areas at Level 3 address both project and organizational issues, as the organization establishes an infrastructure that institutionalizes effective software engineering and management processes across all projects. They are Organization Process Focus, Organization Process Definition, Training Program, Integrated Software Management, Software Product Engineering, Intergroup Coordination, and Peer Reviews.

The key process areas at Level 4 focus on establishing a quantitative understanding of both the software process and the software work products being built. They are Quantitative Process Management and Software Quality Management.

The key process areas at Level 5 cover the issues that both the organization and the projects must address to implement continual, measurable software process improvement. They are Defect Prevention, Technology Change Management, and Process Change Management.

Each key process area is described in terms of the key practices that contribute to satisfying its goals. The key practices describe the infrastructure and activities that contribute most to the effective implementation and institutionalization of the key process area.

Relation to Other Frameworks: The Capability Maturity Model for Software (SW-CMM), Systems Engineering Capability Maturity Model (SE-CMM), and Integrated Product Development Capability Maturity Model (IPD-CMM) are being integrated by the CMMI project.

Sources of Framework:

- Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber, "Capability Maturity Model for Software, Version 1.1", Software Engineering Institute, CMU/SEI-93-TR-24, DTIC Number ADA263403, February 1993.
- Mark C. Paulk, Charles V. Weber, Suzanne M. Garcia, Mary Beth Chrissis, and Marilyn W. Bush, "Key Practices of the Capability Maturity Model, Version 1.1", Software Engineering Institute, CMU/SEI-93-TR-25, DTIC Number ADA263432, February 1993.

Related Links:

http://www.sei.cmu.edu/cmm/

CMMI

Publication Date: Version 1.02 - December 2000

Description: The CMM Integration project was formed to address the problem of having to use multiple Capability Maturity Models. The initial mission of the project was to combine three source models:

1. Capability Maturity Model for Software (SW-CMM) v2.0 draft C
2. Electronic Industries Alliance Interim Standard (EIA/IS) 731, Systems Engineering Capability Model (SECM)
3. Integrated Product Development Capability Maturity Model (IPD-CMM) v0.98

into a single model for use by organizations pursuing enterprise-wide process improvement.

The content and characteristics of these three models provide the basis for the initial CMM Integration Product Suite. The direction is to integrate the development characteristics and delivery methods of these and future capability models (CMs), which will enable users to reduce the cost of performing assessments and implementing improvements.

The initial CMM Integration Product Suite includes a framework for generating CMMI products to meet business objectives/mission needs, and a set of CMMI products produced by the framework. The framework includes common elements and best features of the current models as well as rules and methods for generating CMMI products. Discipline-specific elements (e.g., software, systems engineering) of the CMMI Product Suite will provide the user with the ability to select elements applicable to specific situations.

The purpose of Capability Maturity Model (CMM®) Integration is to provide guidance for improving organization's processes and their ability to manage the development, acquisition, and maintenance of products and services. CMM Integration places proven practices into a structure that helps organizations to assess their organizational maturity and process area capability, establish priorities for improvement, and guide the implementation of these improvements.

Sources of Framework:

- CMMI-SE/SW V1.02 (Continuous Representation)
- CMMI-SE/SW V1.02 (Staged Representation)
- CMMI-SE/SW/IPPD V1.02 (Continuous Representation)
- CMMI-SE/SW/IPPD V1.02 (Staged Representation)

Related Links:

http://www.sei.cmu.edu/cmmi/

Capability Maturity Model for Software (SW-CMM)

Systems Engineering Capability Maturity Model (SE-CMM)

Integrated Product Development Capability Maturity Model (IPD-CMM)

Personal Software Process and Team Software Process

Publication Date: December 2000

Description: To have a high-performance software organization you must have high-performance teams, staffed with high-performance software engineers. The Software Engineering Institute has developed the Personal Software Process (PSP) and the Team Software Process (TSP) to provide a roadmap for organizations and individuals to follow on this road to high-performance.

While the Capability Maturity Model (CMM) focuses on what organizations should do, it does not specify how to reach those goals. The PSP provides specific guidance on how individual engineers can continually improve their performance. The TSP provides specific guidance on how PSP-trained engineers can work as effective team members of high-performance team. All of these technologies can work together to allow organizations to produce quality software on schedule.

The **Personal Software Process** provides engineers with a disciplined personal framework for doing software work. The PSP process consists of a set of methods, forms, and scripts that show software engineers how to plan, measure, and manage their work. It is introduced with a textbook and a course that are designed for both industrial and academic use. The PSP is designed for use with any programming language or design methodology and it can be used for most aspects of software work, including writing requirements, running tests, defining processes, and repairing defects.

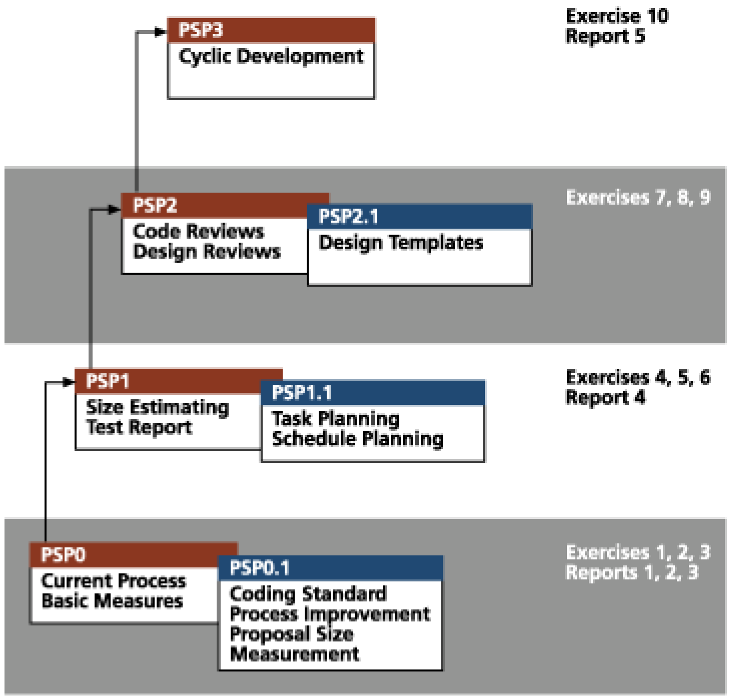


Figure 1: The PSP process evolution

As shown in Figure 1, the engineers follow prescribed methods, represented as levels PSP0 through PSP3, and write a defined set of 10 programming exercises and five reports. With each exercise, they are gradually introduced to various advanced software engineering methods. By measuring their own performance, the engineers can see the effect of these methods on their work.

When engineers use the PSP, the recommended process goal is to produce zero-defect products on schedule and within planned costs. When used with the Team Software Process the PSP has been effective in helping engineers achieve these objectives.

The **Team Software Process** extends and refines the CMM and PSP methods to guide engineers in their work on development and maintenance teams. It shows them how to build a self-directed team and how to perform as an effective team member. It also shows management how to guide and support these teams and how to maintain an environment that fosters high team performance.

The TSP has five objectives:

- Build self-directed teams that plan and track their work, establish goals, and own their processes and plans. These can be pure software teams or integrated product teams (IPT) of three to about 20 engineers.
- Show managers how to coach and motivate their teams and how to help them sustain peak performance.
- Accelerate software process improvement by making CMM Level 5 behavior normal and expected.
- Provide improvement guidance to high-maturity organizations.
- Facilitate university teaching of industrial-grade team skills.

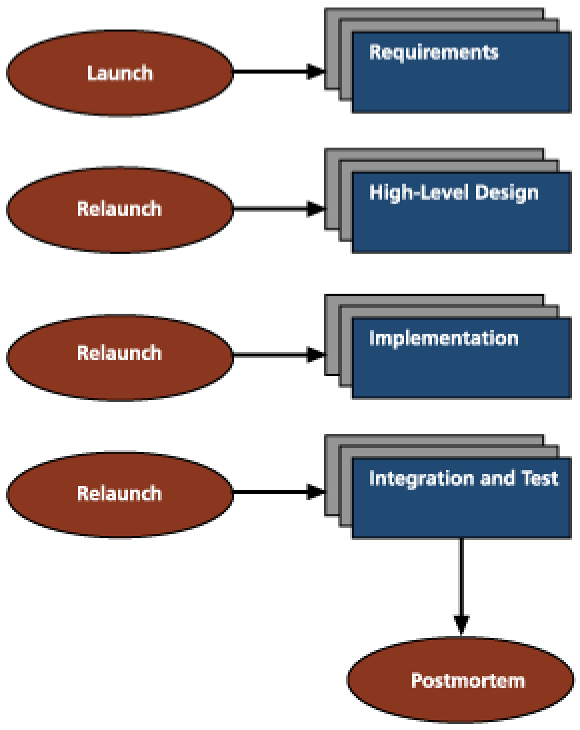


Figure 2: TSP structure

The principal benefit of the TSP is that it shows engineers how to produce quality products for planned costs and on aggressive schedules. It does this by showing engineers how to manage their work and by making them owners of their plans and processes.

The TSP provides team projects with explicit guidance on how to accomplish their objectives. As shown in Figure 2, the TSP guides teams through the four typical phases of a project.

These projects may start or end on any phase, or they can run from beginning to end. Before each phase, the team goes through a complete launch or relaunch, where they plan and organize their work. Generally, once team members are PSP trained, a four-day launch workshop provides enough guidance for the team to complete a full project phase. Teams then need a two-day relaunch workshop to kick off the second and each subsequent phase. These launches are not training; they are part of the project.

To start a TSP project, the launch process script leads teams through the following steps:

- Review project objectives with management.
- Establish team roles.
- Agree on and document the team's goals.
- Produce an overall development strategy.
- Define the team's development process.
- Plan for the needed support facilities.
- Make a development plan for the entire project.
- Make a quality plan and set quality targets.
- Make detailed plans for each engineer for the next phase.
- Merge the individual plans into a team plan.
- Rebalance team workload to achieve a minimum overall schedule.
- Assess project risks and assign tracking responsibility for each key risk.
- Hold a launch postmortem.

In the final launch step, the team reviews its plans and the project's key risks with management. Once the project starts, the team conducts weekly team meetings and periodically reports its status to management and to the customer.

In the four-day launch workshop, TSP teams produce:

- Written team goals
- Defined team roles
- A process development plan
- The team quality plan
- The project's support plan
- An overall development plan and schedule
- Detailed next-phase plans for each engineer

- A project risk assessment
- A project status report

Early experience with the TSP shows that its use improves the quality and productivity of engineering teams while helping them to more precisely meet cost and schedule commitments.

The CMM, PSP, and TSP provide an integrated three-dimensional framework for process improvement. As shown in Table 1, the CMM has 18 key process areas, and the PSP and TSP guide engineers in addressing almost all of them. These methods not only help engineers be more effective but also provide the in-depth understanding needed to accelerate organizational process improvement.

Level	Focus	Key Process Area	PSP	TSP
5 Optimizing	Continuous Process Improvement	Defect Prevention	X	X
		Technology Change Management	X	X
		Process Change Management	X	X
4 Managed	Product and Process Quality	Quantitative Process Management	X	X
		Software Quality Management	X	X
3 Defined	Engineering Process	Organization Process Focus	X	X
		Organization Process Definition	X	X
		Training Program		
		Integrated Software Management	X	X
		Software Product Engineering	X	X
		Intergroup Coordination		X
		Peer Reviews	X	X
2 Repeatable	Project Management	Requirements Management		X
		Software Project Planning	X	X
		Software Project Tracking	X	X
		Software Quality Assurance		X
		Software Configuration Management		X
		Software Subcontract Management		

Table 1: PSP and TSP coverage of CMM key process areas

The CMM provides a useful framework for organizational assessment and a powerful stimulus for process improvement. The experiences of many organizations show that the CMM is effective in helping software organizations improve their performance.

Once groups have started process improvement and are on their way toward CMM Level 2, the PSP shows engineers how to address their tasks in a professional way. Although relatively new, the PSP has already shown its potential to improve engineers' ability to plan and track their work and to produce quality products.

Once engineering teams are PSP trained, they generally need help in applying advanced process methods to their projects. The TSP guides these teams in launching their projects and in planning and managing their work. Perhaps most important, the TSP shows managers how to guide and coach their software teams to consistently perform at their best.

Sources of Framework:

- The Personal Software Process (PSP), Humphrey, Watts S. Technical Report CMU/SEI-2000-TR-022, ESC-TR-2000-022, December 2000
- The Team Software Process (TSP), Humphrey, Watts S. Technical Report CMU/SEI-2000-TR-023, ESC-TR-2000-023, December 2000

Related Links:
http://www.sei.cmu.edu/tsp/
http://www.sei.cmu.edu/tsp/psp.html
http://www.sei.cmu.edu/tsp/tsp.html

ISO 9000 standards family

Publication Date: initial version of ISO 9000 standard series was published in 1987, major updates were done in 1994 and 2000, ISO 9000-3 Guidance for Software was published in 1991, major revision was in 1997. Now the current version is obsolete and will be reworked according to ISO 9000:2000.

Description: The ISO 9000 Standards are Quality System standards. They were developed by the International Organization for Standardization (ISO) with representation from over 90 countries.

The ISO 9000 family consists of the following standards, guidelines and technical reports:

Standards and guidelines	**Purpose**
ISO 9000:2000, *Quality management systems - Fundamentals and vocabulary*	Establishes a starting point for understanding the standards and defines the fundamental terms and definitions used in the ISO 9000 family which you need to avoid misunderstandings in their use.
ISO 9001:2000, *Quality management systems - Requirements*	This is the requirement standard you use to assess your ability to meet customer and applicable regulatory requirements and thereby address customer satisfaction. It is now the only standard in the ISO 9000 family against which third-party certification can be carried.
ISO 9004:2000, *Quality management systems - Guidelines for performance improvements*	This guideline standard provides guidance for continual improvement of your quality management system to benefit all parties through sustained customer satisfaction.
ISO 19011, *Guidelines on Quality and/or Environmental Management Systems Auditing* (currently under development)	Provides you with guidelines for verifying the system's ability to achieve defined quality objectives. You can use this standard internally or for auditing your suppliers.
ISO 10005:1995, *Quality management - Guidelines for quality plans*	Provides guidelines to assist in the preparation, review, acceptance and revision of quality plans.
ISO 10006:1997, *Quality management - Guidelines to quality in project management*	Guidelines to help you ensure the quality of both the project processes and the project products.
ISO 10007:1995, *Quality management - Guidelines for configuration management*	Gives you guidelines to ensure that a complex product continues to function when components are changed individually.
ISO/DIS 10012, *Quality assurance requirements for measuring equipment -* Part 1: *Metrological confirmation system for measuring equipment*	Give you guidelines on the main features of a calibration system to ensure that measurements are made with the intended accuracy.
ISO 10012-2:1997, *Quality assurance for*	Provides supplementary guidance on the application of

Standards and guidelines	**Purpose**
measuring equipment - Part 2: *Guidelines for control of measurement of processes*	statistical process control when this is appropriate for achieving the objectives of Part 1.
ISO 10013:1995, *Guidelines for developing quality manuals*	Provides guidelines for the development, and maintenance of quality manuals, tailored to your specific needs.
ISO/TR 10014:1998, *Guidelines for managing the economics of quality*	Provides guidance on how to achieve economic benefits from the application of quality management.
ISO 10015:1999, *Quality management - Guidelines for training*	Provides guidance on the development, implementation, maintenance and improvement of strategies and systems for training that affects the quality of products.
ISO/TS 16949:1999, *Quality systems - Automotive suppliers - Particular requirements for the application of ISO 9001:1994*	Sector specific guidance to the application of ISO 9001 in the automotive industry.

The new ISO 9001:2000 standard requirements are summarized below. For more detail, please see the associated ISO 9001:2000 clauses (in brackets).

- Communicate with customers (7.2.3).
- Identify customer requirements (5.2, 7.2.1).
- Meet customer requirements (5.2).
- Monitor and measure customer satisfaction (8.2.1).
- Meet regulatory requirements (5.1).
- Meet statutory requirements (5.1).
- Support internal communication (5.5.3).
- Provide quality infrastructure (6.3).
- Provide a quality work environment (6.4).
- Evaluate the effectiveness of training (6.2.2).
- Monitor and measure processes (8.2.3).
- Evaluate the suitability of quality management system (8.4).
- Evaluate the effectiveness of quality management system (8.4).
- Identify quality management system improvements (5.1, 8.4).
- Improve quality management system (5.1, 8.5).

In order to understand ISO 9001:2000 at a deeper level, you need to recognize that ISO uses a *process approach* to quality management. While the process approach is not new, the increased emphasis ISO now gives to it is new. It is now central to the way ISO thinks about quality management systems. According to this approach, a quality management system can be thought of as a single large *process* that uses many *inputs* to generate many *outputs*. This large process is, in turn, made up of many smaller processes. Each of these processes uses inputs from other processes to generate outputs which, in turn, are used by still other processes.

A detailed analysis of the Standard reveals that an ISO 9001:2000 Quality Management System is made up of at least 21 processes (22 if you recognize that the Quality Management System as a whole is also a process). These 21 processes are listed below:

1. Quality Management Process
2. Resource Management Process
3. Regulatory Research Process
4. Market Research Process
5. Product Design Process
6. Purchasing Process
7. Production Process
8. Service Provision Process
9. Product Protection Process
10. Customer Needs Assessment Process
11. Customer Communications Process
12. Internal Communications Process
13. Document Control Process
14. Record Keeping Process
15. Planning Process
16. Training Process
17. Internal Audit Process
18. Management Review Process
19. Monitoring and Measuring Process
20. Nonconformance Management Process
21. Continual Improvement Process

In order to develop a quality management system that meets the new ISO 9001:2000 standard, you must create or modify each of the above processes. You must:

1. Develop each process.
2. Document each process.
3. Implement each process.
4. Monitor each process.
5. Improve each process.

Each process uses *inputs* to generate *outputs*, and all of these processes are interconnected using these *input-output relationships*. The output from one process becomes the input for other processes. Because of this, inputs and outputs are really the same thing. Incomplete list of some general types of inputs/outputs includes:

- Products
- Services
- Information
- Documents
- Reports
- Records
- Results
- Needs
- Data
- Expectations
- Requirements
- Complaints
- Comments

- Feedback
- Resources
- Measurements
- Authorizations
- Decisions
- Plans
- Ideas
- Solutions
- Proposals
- Instructions

In summary, an ISO 9001:2000 Quality Management System is made up of many processes, and these processes are *glued together* by means of many input-output relationships. These input-output relationships turn a simple list of processes into an *integrated system*. Without these input-output relationships, you wouldn't have a Quality Management System.

Relation to Other Frameworks: The ISO/IEC 12207 standard "Information Technology – Software life cycle processes" cites ISO 9001 as a normative reference. The TickIt standard provides guidance in understanding and applying ISO 9001 in the IT industry.

Source of Framework: ISO9000, 9001, 9004-2000.PDF

Related Links:

http://www.iso.ch/iso/en/iso9000-14000/iso9000/iso9000index.html

TickIt

Publication Date: Issue 5.0 was published in 2000

Description: The TickIT scheme, originating in 1991, is an UK accreditation scheme to the ISO 9001 standard specifically for software companies. Organizations achieving certification under the TickIT scheme are permitted to use the TickIT logo in their ISO 9001 certification materials. The TickIT Guide (A guide to Software Quality System Construction and Certification), Issue 5.0 contains updated and enhanced guidelines for the application of BS EN ISO 9001:2000 to software development, detailed cross referencing to ISO/IEC 12207 software life cycle processes helping software companies to adopt a process-based approach to quality systems management, extended, enhanced general guidance for software supplying organizations, revised general guidance for auditors, extended, updated guidance for customers of computerized systems with a focus on the key role customers play in the achievement of quality, general information about the operation of the TickIT scheme and case studies showing how major companies have used CMM,ISO/IEC 15504 and the EFQM Excellence Model for process improvement.

Relation to Other Frameworks: TickIT is the British Standards Institute's implementation of the ISO 9000 series for software development.

Source of Framework:

- TickIT may be ordered from DISC TickIT Office, British Standards House, 389 Chiswick High Road, London W44AL, Tel:0171 602 8536, Fax: 0171 602 8912.

Related Links:

http://www.tickit.org

ISO/IEC TR 15504 Information Technology — Software Process Assessment (SPICE)

Publication Date: No official release yet. ISO/IEC 15504 Technical Report was released in 1998.
Description: In June 1991, the ISO/IEC JTC1/SC7 initiated a study on the need for a software process assessment standard. The results of this study drove the establishment of the Software Process Improvement and Capability dEtermination (SPICE) project in 1993 to standardize and improve on the existing software assessment methodologies. The SPICE project is developing the ISO/IEC 15504 standard to address all processes involved in software acquisition, development, operation, supply, maintenance, and support and has been created to be aligned closely with ISO/IEC 12207:1995 "Software Life Cycle Processes." Twenty-three requirements were addressed in the first working draft published in June 1995. This emerging standard has completed ISO/IEC 15504 DTR ballot in 2Q98. The DTR contained nine parts covering concepts, process reference model and improvement guide, assessment model and guides, qualifications of assessors, and guide for determining supplier process capability.

The proposed ISO/IEC 15504 standard is dual-path architecture composed of domain-specific processes and capability measures common to each process. In the reference model (described in Part 2 of ISO/IEC 15504 DTR), there are 5 process categories with upwards of 40 processes. In determining the capability of an organization processes, there are six levels defined with nine common attributes (generic practices) allocated across the highest five levels. The organization's assessment result will be a summary of all the highest capability levels achieved for the processes assessed.

These are the parts of ISO 15504:

- ISO/IEC DTR 15504 Software Process Assessment.
- ISO/IEC DTR 15504-1 Part 1: Concepts and Introductory Guide.
- ISO/IEC DTR 15504-2 Part 2: A Reference Model for Processes and Process Capability.
- ISO/IEC DTR 15504-3 Part 3: Performing an Assessment.
- ISO/IEC DTR 15504-4 Part 4: Guide to Performing Assessments.
- ISO/IEC DTR 15504-5 Part 5: An Assessment Model and Indicator Guidance.
- ISO/IEC DTR 15504-6 Part 6: Guide to Competency of Assessors.
- ISO/IEC DTR 15504-7 Part 7: Guide for Use in Process Improvement.
- ISO/IEC DTR 15504-8 Part 8: Guide for Use in Determining Supplier Process Capability.
- ISO/IEC DTR 15504-9 Part 9: Vocabulary.

Relation to Other Frameworks:

- ISO/IEC 15504 Part 2, Reference Model for Processes and Process Capability, is strongly related to ISO/IEC 12207 standard, Information technology – Software Life-Cycle Processes.

- ISO/IEC 15504 Part 6, Guide to Qualification of Assessors, is strongly related to ISO 10011- 2, Guidelines for Auditing Quality Systems – Part 2: Qualification Criteria for Quality Systems Auditors.
- ISO/IEC 15504 is intended to be harmonious with ISO 9000.
- SE-CMM used the two-axis architecture of the ISO/IEC 15504 process model.

Source of Framework:

- ISO-IEC TR 15504-1.pdf
- ISO-IEC TR 15504-2.pdf
- ISO-IEC TR 15504-3.pdf
- ISO-IEC TR 15504-4.pdf
- ISO-IEC TR 15504-5.pdf
- ISO-IEC TR 15504-6.pdf
- ISO-IEC TR 15504-7.pdf
- ISO-IEC TR 15504-8.pdf
- ISO-IEC TR 15504-9.pdf

Related Links:

- SEI is focal point for coordinating U.S. input to the development of ISO 15504 and disseminating information on current status, other sources, and conferences, etc.
- The official SPICE Web site provides information for the participants and others who would like to learn more about SPICE.

ISO/IEC 12207 Information Technology — Software Life-Cycle Processes

Publication Date: 1 August 1995

Description: This standard describes the major component processes of a complete software life cycle, their interfaces with one another, and the high-level relations that govern their interactions. This standard covers the life cycle of software from conceptualization of ideas through retirement. ISO/IEC 12207 describes the following life-cycle processes:

- ***Primary Processes***: Acquisition, Supply, Development, Operation, and Maintenance.
- ***Supporting Processes***: Documentation, Configuration Management, Quality Assurance, Verification Validation, Joint Review, Audit, and Problem Resolution.
- ***Organization Processes***: Management, Infrastructure, Improvement, and Training.

ISO/IEC 12207 also describes how to tailor the standard for an organization or project.

Source of Framework:

- Hard copies of ISO/IEC 12207 are available from http://www.iso.ch

Related Links:

- An article International Standard ISO/IEC 12207 Software Lifecycle Processes provides a detailed description of ISO/IEC 12207.
- U. S. Software Life Cycle Process Standards is an article from STSC's CrossTalk journal that discusses the status and relationships of ISO/IEC 12207, IEEE/EIA 12207, and EIA/IEEE J-STD-016. Dated July 1997.
- ISO 12207 and Related Software Life-Cycle Standards is an article published on the ACM's Web site that describes ISO 12207; traces the evolution of life-cycle standards; differentiates the efforts of IEEE, ISO, and other organizations; and discusses the significance of ISO 12207 to international software acquisition.

BOOSTRAP

Framework Name: BOOTSTRAP

Publication Date: Version 3.2 - 2000

Description: BOOTSTRAP was funded by the European Commission within the ESPRIT program. Its objective was to develop a methodology for software process assessment, quantitative measurement, and improvement, and validate the methodology by applying it to a number of companies. After the completion of the ESPRIT project the BOOTSTRAP Institute was established with the main objective of the continuous development and promotion of the BOOTSTRAP methodology.

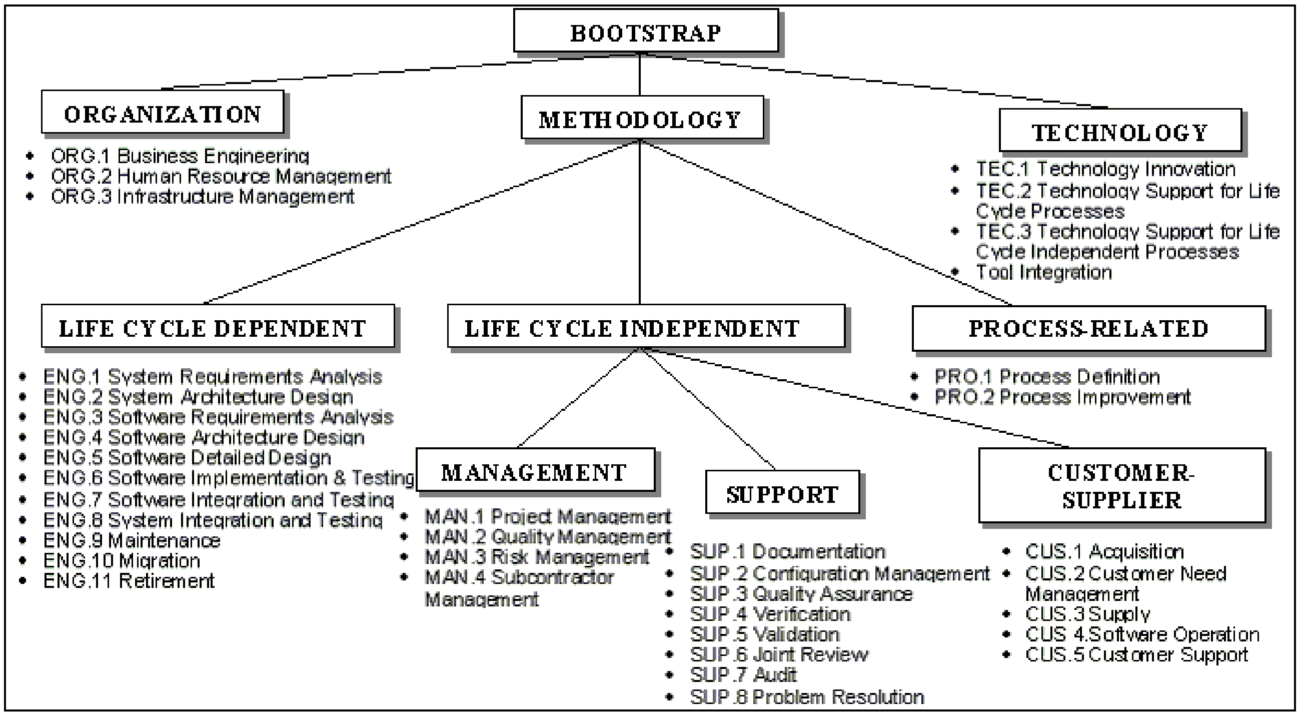
The BOOTSTRAP methodology has the following objectives:

- to provide support for the evaluation of process capability against a set of recognized software engineering best practices;
- to include internationally recognized software engineering standards as sources for identification of best practices;
- to support the evaluation of how the reference standard has been implemented in the assessed organization;
- to assure the reliability and repeatability of the evaluation;
- to identify, in the assessed organization, process strengths and weaknesses;
- to support improvement planning with suitable and reliable results
- to support the achievement of the organization's goals by planning improvement actions;
- to help increase process effectiveness while implementing standard requirements in the organization.

The main features:

- *the assessment process:* the assessment process is part of the improvement. Assessment results provide the main input for the improvement action plan and provide feedback from the improvement activities implemented. During a BOOTSTRAP assessment the organizational processes are evaluated to define each process. The process capability evaluation is based on the BOOTSTRAP process model.
- *the process model:* the BOOTSTRAP process model defines processes and capability levels. Process capability is measured based on the following capability levels:
    - Level 0: Incomplete Process
    - Level 1: Performed Process
    - Level 2: Managed Process
    - Level 3: Established Process
    - Level 4: Predictable Process
    - Level 5: Optimizing Process

  The figure below represents the tree structure of the BOOTSTRAP process model:


BOOTSTRAP
ORGANIZATION
ORG.1 Business Engineering
ORG.2 Human Resource Management
ORG.3 Infrastructure Management
METHODOLOGY
TECHNOLOGY
TEC.1 Technology Innovation
TEC.2 Technology Support for Life Cycle Processes
TEC.3 Technology Support for Life Cycle Independent Processes
Tool Integration
LIFE CYCLE DEPENDENT
ENG.1 System Requirements Analysis
ENG.2 System Architecture Design
ENG.3 Software Requirements Analysis
ENG.4 Software Architecture Design
ENG.5 Software Detailed Design
ENG.6 Software Implementation & Testing
ENG.7 Software Integration and Testing
ENG.8 System Integration and Testing
ENG.9 Maintenance
ENG.10 Migration
ENG.11 Retirement
LIFE CYCLE INDEPENDENT
PROCESS-RELATED
PRO.1 Process Definition
PRO.2 Process Improvement
MANAGEMENT
MAN.1 Project Management
MAN.2 Quality Management
MAN.3 Risk Management
MAN.4 Subcontractor Management
SUPPORT
SUP.1 Documentation
SUP.2 Configuration Management
SUP.3 Quality Assurance
SUP.4 Verification
SUP.5 Validation
SUP.6 Joint Review
SUP.7 Audit
SUP.8 Problem Resolution
CUSTOMER-SUPPLIER
CUS.1 Acquisition
CUS.2 Customer Need Management
CUS.3 Supply
CUS 4.Software Operation
CUS.5 Customer Support

- *questionnaires:* a major part of the assessment is gathering data. The BOOTSTRAP methodology provides two questionnaires, one to gather data about the software development organization and the other to gather data on projects.
- *scoring, rating and result presentation:* assessment results are the basis for process improvement planning, but this role can take place only if assessment data is reliable and provides a good representation of the assessed organisation's capability. Reliability and repeatability are obtained by:
  - assuring that assessors have the same background and use the same approach (this is guaranteed by the BOOTSTRAP assessor accreditation process), and
  - by applying precise scoring and rating rules.

  BOOTSTRAP evaluates each practice based on a four-value scale (not, partially, largely and fully adequate). The capability level is then counted in two ways:

  - on an algorithmic basis showing quartiles within each level;
  - by applying the SPICE rules for deriving the capability level rating.

  Final assessment results are prepared with a tool and presented as capability profiles with quartiles and a SPICE profile. The capability profile is produced at organizational and project level.

- *process improvement guidelines:* these guidelines support the identification of processes that have the greatest impact on the achievement of the organizational goals; then improvement priorities are assigned to processes with low capability and high impact. Improvement targets and priorities are evaluated in light of the potential

risks to the organization in not achieving its goals. This helps provide a rationale for management to undertake the improvement effort.
- *the BOOTSTRAP database:* the assessment data from all BOOTSTRAP assessments are automatically collected into a central database. The BOOTSTRAP database has the following important role in the BOOTSTRAP assessment:
  - to collect data on assessment results performed within Europe, in order to provide a picture of the maturity of the European software industry, and
  - to position each assessed organization within the applicable software industry sector.

Relation to Other Frameworks: BOOTSTRAP incorporates the actual standards for software process definition and improvement. BOOTSTRAP is compliant to ISO/IEC TR 15504 and it allows comparing assessment result to CMM capability levels.

Source of Framework:

- Hard copies of BOOTSTRAP are available from BOOTSTRAP Institute

Rational Unified Process

Publication Date: Version 5.0 - 1998

Description: The Rational Unified Process is a software engineering process, delivered through a web-enabled, searchable knowledge base. It provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end-users, within a predictable schedule and budget.

The Rational Unified Process enhances team productivity, by providing every team member with easy access to a knowledge base with guidelines, templates and tool mentors for all critical development activities. By having all team members accessing the same knowledge base, no matter if you work with requirements, design, test, project management, or configuration management, we ensure that all team members share a common language, process and view of how to develop software.

The Rational Unified Process activities create and maintain models. Rather than focusing on the production of large amount of paper documents, the Unified Process emphasizes the development and maintenance of models--semantically rich representations of the software system under development.

The Rational Unified Process is a guide for how to effectively use the Unified Modeling Language (UML). The UML is an industry-standard language that allows us to clearly communicate requirements, architectures and designs. The UML was originally created by Rational Software, and is now maintained by the standards organization Object Management Group (OMG).

The Rational Unified Process is supported by tools, which automate large parts of the process. They are used to create and maintain the various artifacts-models in particular-of the software engineering process: visual modeling, programming, testing, etc. They are invaluable in supporting all the bookkeeping associated with the change management as well as the configuration management that accompanies each iteration.

The Rational Unified Process is a configurable process. No single process is suitable for all software development. The Unified Process fits small development teams as well as large development organizations. The Unified Process is founded on a simple and clear process architecture that provides commonality across a family of processes. Yet, it can be varied to accommodate different situations. It contains a Development Kit, providing support for configuring the process to suit the needs of a given organization.

The Rational Unified Process captures many of the best practices in modern software development in a form that is suitable for a wide range of projects and organizations. Deploying these best practices using the Rational Unified Process as your guide offers development teams a number of key advantages. In next section, we describe the six fundamental best practices of the Rational Unified Process.

Effective Deployment of 6 Best Practices

The Rational Unified Process describes how to effectively deploy commercially proven approaches to software development for software development teams. These are called "best practices" not so much because you can precisely quantify their value, but rather, because they are observed to be commonly used in industry by successful organizations.

The Rational Unified Process provides each team member with the guidelines, templates and tool mentors necessary for the entire team to take full advantage of among others the following best practices:

1. Develop software iteratively
2. Manage requirements
3. Use component-based architectures
4. Visually model software
5. Verify software quality
6. Control changes to software

Develop Software Iteratively — Given today's sophisticated software systems, it is not possible to sequentially first define the entire problem, design the entire solution, build the software and then test the product at the end. An iterative approach is required that allows an increasing understanding of the problem through successive refinements, and to incrementally grow an effective solution over multiple iterations. The Rational Unified Process supports an iterative approach to development that addresses the highest risk items at every stage in the lifecycle, significantly reducing a project's risk profile. This iterative approach helps you attack risk through demonstrable progress frequent, executable releases that enable continuous end user involvement and feedback. Because each iteration ends with an executable release, the development team stays focused on producing results, and frequent status checks help ensure that the project stays on schedule. An iterative approach also makes it easier to accommodate tactical changes in requirements, features or schedule.

Manage Requirements — The Rational Unified Process describes how to elicit, organize, and document required functionality and constraints; track and document tradeoffs and decisions; and easily capture and communicate business requirements. The notions of use case and scenarios proscribed in the process has proven to be an excellent way to capture functional requirements and to ensure that these drive the design, implementation and testing of software, making it more likely that the final system fulfills the end user needs. They provide coherent and traceable threads through both the development and the delivered system.

Use Component-Based Architectures — The process focuses on early development and baselining of a robust executable architecture, prior to committing resources for full-scale development. It describes how to design a resilient architecture that is flexible, accommodates change, is intuitively understandable, and promotes more effective software reuse. The Rational Unified Process supports component-based software development. Components are non-trivial modules, subsystems that fulfill a clear function. The Rational Unified Process provides a systematic approach to defining an architecture using new and existing components. These are assembled in a well-defined architecture, either ad hoc, or in a component infrastructure such as the Internet, CORBA, and COM, for which an industry of reusable components is emerging.

Visually Model Software — The process shows you how to visually model software to capture the structure and behavior of architectures and components. This allows you to hide the details and write code using "graphical building blocks." Visual abstractions help you communicate different aspects of your software; see how the elements of the system fit together; make sure that the building blocks are consistent with your code; maintain consistency between a design and its implementation; and promote unambiguous communication. The industry-standard Unified Modeling Language (UML), created by Rational Software, is the foundation for successful visual modeling.

Verify Software Quality — Poor application performance and poor reliability are common factors that dramatically inhibit the acceptability of today's software applications. Hence, quality should be reviewed with respect to the requirements based on reliability, functionality, application performance and system performance. The Rational Unified Process assists you in the planning, design, implementation, execution, and evaluation of these test types. Quality assessment is built into the process, in all activities, involving all participants, using objective measurements and criteria, and not treated as an afterthought or a separate activity performed by a separate group.

Control Changes to Software — The ability to manage change--making certain that each change is acceptable, and being able to track changes--is essential in an environment in which change is inevitable. The process describes how to control, track and monitor changes to enable successful iterative development. It also guides you in how to establish secure workspaces for each developer by providing isolation from changes made in other workspaces and by controlling changes of all software artifacts (e.g., models, code, documents, etc.). And it brings a team together to work as a single unit by describing how to automate integration and build management.

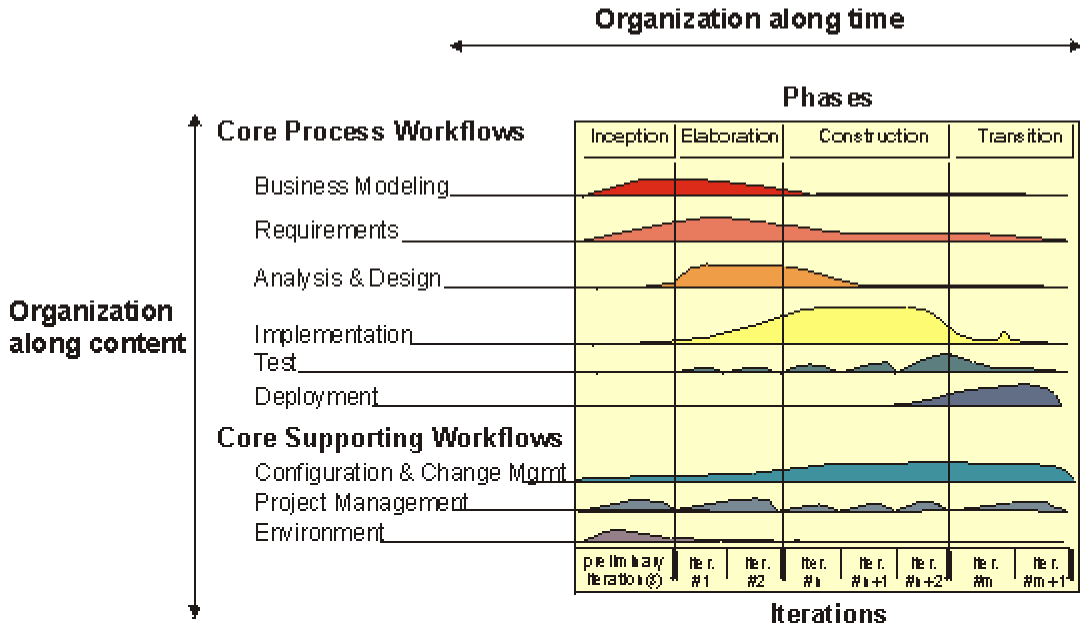
Process Overview

Two Dimensions

The process can be described in two dimensions, or along two axes:

The horizontal axis represents time and shows the dynamic aspect of the process as it is enacted, and it is expressed in terms of cycles, phases, iterations, and milestones.

The vertical axis represents the static aspect of the process: how it is described in terms of activities, artifacts, workers and workflows.



The Iterative Model graph shows how the process is structured along two dimensions.

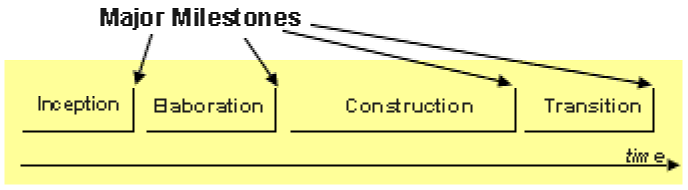
Phases and Iterations — The Time Dimension

This is the dynamic organization of the process along time.

The software lifecycle is broken into cycles, each cycle working on a new generation of the product. The Rational Unified Process divides one development cycle in four consecutive phases.

- Inception phase
- Elaboration phase
- Construction phase
- Transition phase

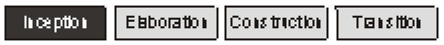
Each phase is concluded with a well-defined *milestone*--a point in time at which certain critical decisions must be made, and therefore key goals must have been achieved.



The phases and major milestones in the process

Each phase has a specific purpose.

Inception Phase



During the inception phase, you establish the business case for the system and delimit the project scope. To accomplish this you must identify all external entities with which the system will interact (actors) and define the nature of this interaction at a high-level. This involves identifying all use cases and describing a few significant ones. The business case includes success criteria, risk assessment, and estimate of the resources needed, and a phase plan showing dates of major milestones.

The outcome of the inception phase is:

- A vision document: a general vision of the core project's requirements, key features, and main constraints.
- An initial use-case model (10%-20% complete).
- An initial project glossary (may optionally be partially expressed as a domain model).
- An initial business case, which includes business context, success criteria (revenue projection, market recognition, and so on), and financial forecast.
- An initial risk assessment.
- A project plan, showing phases and iterations.
- A business model, if necessary.
- One or several prototypes.

Milestone: Lifecycle Objectives

Inception
Elaboration
Construction
Transition
Lifecycle Objective

At the end of the inception phase is the first major project milestone: the Lifecycle Objectives Milestone. The evaluation criteria for the inception phase are:

- Stakeholder concurrence on scope definition and cost/schedule estimates.
- Requirements understanding as evidenced by the fidelity of the primary use cases.
- Credibility of the cost/schedule estimates, priorities, risks, and development process.
- Depth and breadth of any architectural prototype that was developed.
- Actual expenditures versus planned expenditures.

The project may be cancelled or considerably re-thought if it fails to pass this milestone.

Elaboration Phase

The purpose of the elaboration phase is to analyze the problem domain, establish a sound architectural foundation, develop the project plan, and eliminate the highest risk elements of the project. To accomplish these objectives, you must have the "mile wide and inch deep" view of the system. Architectural decisions have to be made with an understanding of the whole system: its scope, major functionality and nonfunctional requirements such as performance requirements.

It is easy to argue that the elaboration phase is the most critical of the four phases. At the end of this phase, the hard "engineering" is considered complete and the project undergoes its most important day of reckoning: the decision on whether or not to commit to the construction and transition phases. For most projects, this also corresponds to the transition from a mobile, light and nimble, low-risk operation to a high-cost, high-risk operation with substantial inertia. While the process must always accommodate changes, the elaboration phase activities ensure that the architecture, requirements and plans are stable enough, and the risks are sufficiently mitigated, so you can predictably determine the cost and schedule for the completion of the development. Conceptually, this level of fidelity would correspond to the level necessary for an organization to commit to a fixed-price construction phase.

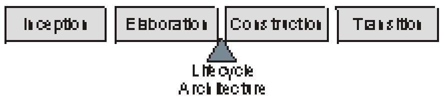
In the elaboration phase, an executable architecture prototype is built in one or more iterations, depending on the scope, size, risk, and novelty of the project. This effort should at least address the critical use cases identified in the inception phase, which typically expose the major technical risks of the project. While an evolutionary prototype of a production-quality component is always the goal, this does not exclude the development of one or more exploratory, throw-away prototypes to mitigate specific risks such as design/requirements trade-offs, component feasibility study, or demonstrations to investors, customers, and end-users.

The outcome of the elaboration phase is:

- A use-case model (at least 80% complete) - all use cases and actors have been identified, and most use-case descriptions have been developed.

- Supplementary requirements capturing the non-functional requirements and any requirements that are not associated with a specific use case.
- A Software Architecture Description.
- An executable architectural prototype.
- A revised risk list and a revised business case.
- A development plan for the overall project, including the coarse-grained project plan, showing iterations" and evaluation criteria for each iteration.
- An updated development case specifying the process to be used.
- A preliminary user manual (optional).

Milestone: Lifecycle Architecture



At the end of the elaboration phase is the second important project milestone, the Lifecycle Architecture Milestone. At this point, you examine the detailed system objectives and scope, the choice of architecture, and the resolution of the major risks.

- The main evaluation criteria for the elaboration phase involves the answers to these questions:
- Is the vision of the product stable?
- Is the architecture stable?
- Does the executable demonstration show that the major risk elements have been addressed and credibly resolved?
- Is the plan for the construction phase sufficiently detailed and accurate? Is it backed up with a credible basis of estimates?
- Do all stakeholders agree that the current vision can be achieved if the current plan is executed to develop the complete system, in the context of the current architecture?
- Is the actual resource expenditure versus planned expenditure acceptable?

The project may be aborted or considerably re-thought if it fails to pass this milestone.

Construction Phase

Inception
Elaboration
Construction
Transition

During the construction phase, all remaining components and application features are developed and integrated into the product, and all features are thoroughly tested. The construction phase is, in one sense, a manufacturing process where emphasis is placed on managing resources and controlling operations to optimize costs, schedules, and quality. In this sense, the management mindset undergoes a transition from the development of intellectual property during inception and elaboration, to the development of deployable products during construction and transition.

Many projects are large enough that parallel construction increments can be spawned. These parallel activities can significantly accelerate the availability of deployable releases; they can also increase the complexity of resource management and workflow synchronization. A robust architecture and an understandable plan are highly correlated. In other words, one of the critical qualities of the architecture is its ease of construction. This is one reason why the balanced development of the architecture and the plan is stressed during the elaboration phase.

The outcome of the construction phase is a product ready to put in hands of its end-users. At minimum, it consists of:

- The software product integrated on the adequate platforms.
- The user manuals.
- A description of the current release.

Milestone: Initial Operational Capability

Inception
Elaboration
Construction
Transition
Initial Operational Capability

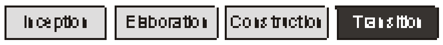
At the end of the construction phase is the third major project milestone (Initial Operational Capability Milestone). At this point, you decide if the software, the sites, and the users are ready to go operational, without exposing the project to high risks. This release is often called a "beta" release.

The evaluation criteria for the construction phase involve answering these questions:

- Is this product release stable and mature enough to be deployed in the user community?
- Are all stakeholders ready for the transition into the user community?
- Are the actual resource expenditures versus planned expenditures still acceptable?

Transition may have to be postponed by one release if the project fails to reach this milestone.

Transition Phase



The purpose of the transition phase is to transition the software product to the user community. Once the product has been given to the end user, issues usually arise that require you to develop new releases, correct some problems, or finish the features that were postponed.

The transition phase is entered when a baseline is mature enough to be deployed in the end-user domain. This typically requires that some usable subset of the system has been completed to an acceptable level of quality and that user documentation is available so that the transition to the user will provide positive results for all parties. This includes:

- "Beta testing" to validate the new system against user expectations
- Parallel operation with a legacy system that it is replacing
- Conversion of operational databases
- Training of users and maintainers
- Roll-out the product to the marketing, distribution, and sales teams

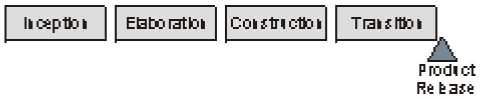
The transition phase focuses on the activities required to place the software into the hands of the users. Typically, this phase includes several iterations, including beta releases, general availability releases, as well as bug fix and enhancement releases. Considerable effort is expended in developing user-oriented documentation, training users, supporting users in their initial product use, and reacting to user feedback. At this point in the lifecycle, however, user feedback should be confined primarily to product tuning, configuring, installation, and usability issues.

The primary objectives of the transition phase include:

- Achieving user self-supportability
- Achieving stakeholder concurrence that deployment baselines are complete and consistent with the evaluation criteria of the vision
- Achieving final product baseline as rapidly and cost effectively as practical

This phase can range from being very simple to extremely complex, depending on the type of product. For example, a new release of an existing desktop product may be very simple, whereas replacing a nation's air-traffic control system would be very complex.

Milestone: Product Release



At the end of the transition phase is the fourth important project milestone, the Product Release Milestone. At this point, you decide if the objectives were met, and if you should start another development cycle. In some cases, this milestone may coincide with the end of the inception phase for the next cycle.

The primary evaluation criteria for the transition phase involve the answers to these questions:

- Is the user satisfied?
- Are the actual resources expenditures versus planned expenditures still acceptable?

Iterations

Each phase in the Rational Unified Process can be further broken down into iterations. Iteration is a complete development loop resulting in a release (internal or external) of an executable product, a subset of the final product under development, which grows incrementally from iteration to iteration to become the final system.

Benefits of an Iterative Approach

Compared to the traditional waterfall process, the iterative process has the following advantages:

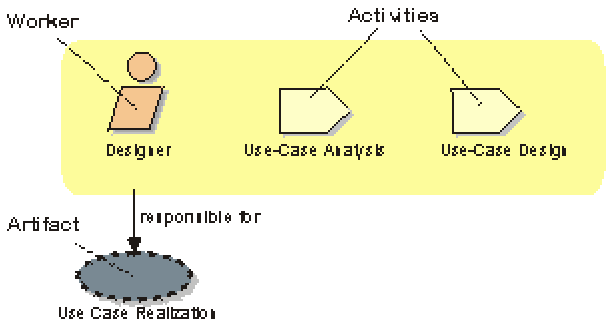
- Risks are mitigated earlier
- Change is more manageable
- Higher level of reuse
- The project team can learn along the way
- Better overall quality

Static Structure of the Process

A process describes who is doing what, how, and when. The Rational Unified Process is represented using four primary modeling elements:

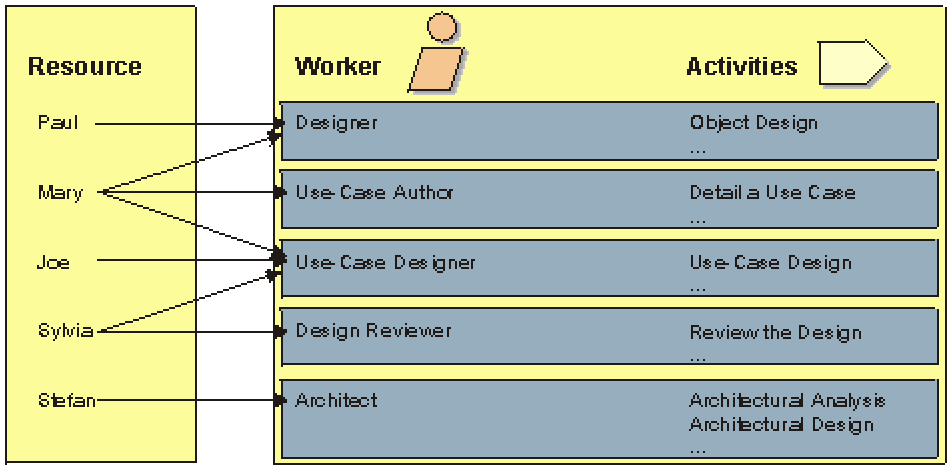
- Workers, the "who"
- Activities, the "how"
- Artifacts, the "what"
- Workflows, the "when"

Activities, Artifacts, and Workers



Worker

A *worker* defines the behavior and responsibilities of an individual, or a group of individuals working together as a team. You could regard a worker as a "hat" an individual can wear in the project. One individual may wear many different hats. This is an important distinction because it is natural to think of a worker as the individual or team itself, but in the Unified Process the worker is more the role defining how the individuals should carry out the work. The responsibilities we assign to a worker include both to perform a certain set of activities as well as being owner of a set of artifacts.



People and workers.

Activity

An *activity* of a specific worker is a unit of work that an individual in that role may be asked to perform. The activity has a clear purpose, usually expressed in terms of creating or updating some artifacts, such as a model, a class, a plan. Every activity is assigned to a specific worker. The granularity of an activity is generally a few hours to a few days, it usually involves one worker, and affects one or only a small number of artifacts. An activity should be usable as an element of planning and progress; if it is too small, it will be neglected, and if it is too large, progress would have to be expressed in terms of an activity's parts.

Example of activities:

- Plan an iteration, for the Worker: Project Manager
- Find use cases and actors, for the Worker: System Analyst
- Review the design, for the Worker: Design Reviewer
- Execute performance test, for the Worker: Performance Tester

Artifact

An *artifact* is a piece of information that is produced, modified, or used by a process. Artifacts are the tangible products of the project, the things the project produces or uses while working towards the final product. Artifacts are used as input by workers to perform an activity, and are the result or output of such activities. In object-oriented design terms, as activities are operations on an active object (the worker), artifacts are the parameters of these activities.

Artifacts may take various shapes or forms:

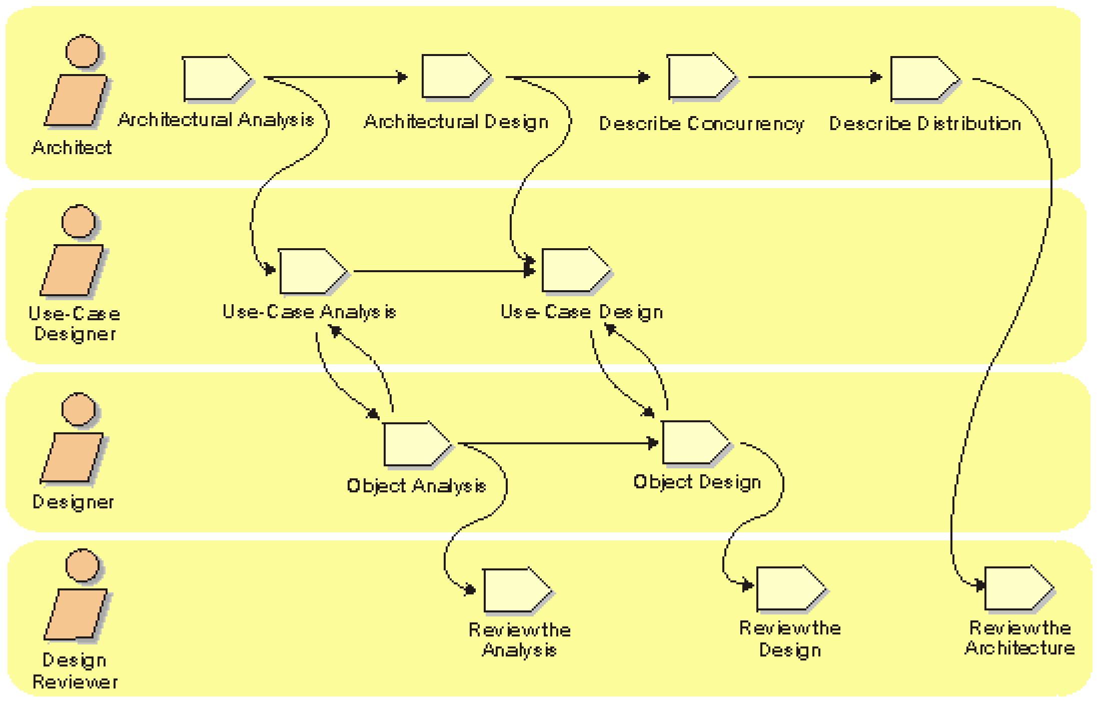
- A model, such as the Use-Case Model or the Design Model
- A model element, i.e. an element within a model, such as a class, a use case or a subsystem
- A document, such as Business Case or Software Architecture Document
- Source code
- Executables

Workflows

A mere enumeration of all workers, activities and artifacts does not quite constitute a process. We need a way to describe meaningful sequences of activities that produce some valuable result, and to show interactions between workers.

A *workflow* is a sequence of activities that produces a result of observable value.

In UML terms, a workflow can be expressed as a sequence diagram, a collaboration diagram, or an activity diagram. We use a form of activity diagrams in this white paper.

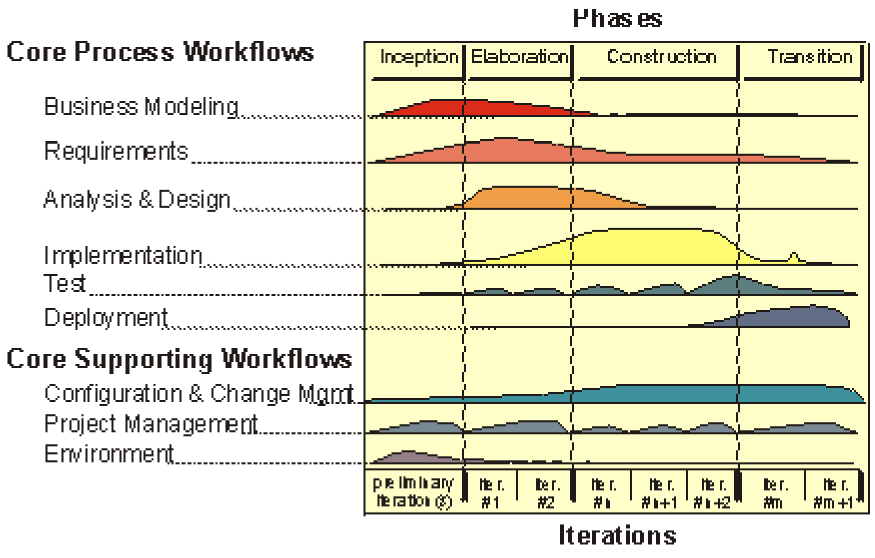


Example of workflow.

Note that it is not always possible or practical to represent all of the dependencies between activities. Often two activities are more tightly interwoven than shown, especially when they involve the same worker or the same individual. People are not machines, and the workflow cannot be interpreted literally as a program for people, to be followed exactly and mechanically. In next section we will discuss the most essential type of workflows in the process, called Core Workflows.

Core Workflows

There are nine core process workflows in the Rational Unified Process, which represent a partitioning of all workers and activities into logical groupings.



The nine core process workflows.

The core process workflows are divided into six core "engineering" workflows:

- Business modeling workflow
- Requirements workflow
- Analysis & Design workflow
- Implementation workflow
- Test workflow
- Deployment workflow

And three core "supporting" workflows:

- Project Management workflow
- Configuration and Change Management workflow
- Environment workflow

Although the names of the six core engineering workflows may evoke the sequential phases in a traditional waterfall process, we should keep in mind that the phases of an iterative process are different and that these workflows are revisited again and again throughout the lifecycle. The actual complete workflow of a project interleaves these nine core workflows, and repeats them with various emphasis and intensity at each iteration.

Business Modeling

One of the major problems with most business engineering efforts is that the software engineering and the business engineering community do not communicate properly with each other. This leads to that the output from business engineering is not used properly as input to the software development effort, and vice-versa. The Rational Unified Process addresses this by providing a common language and process for both communities, as well as showing how to create and maintain direct traceability between business and software models.

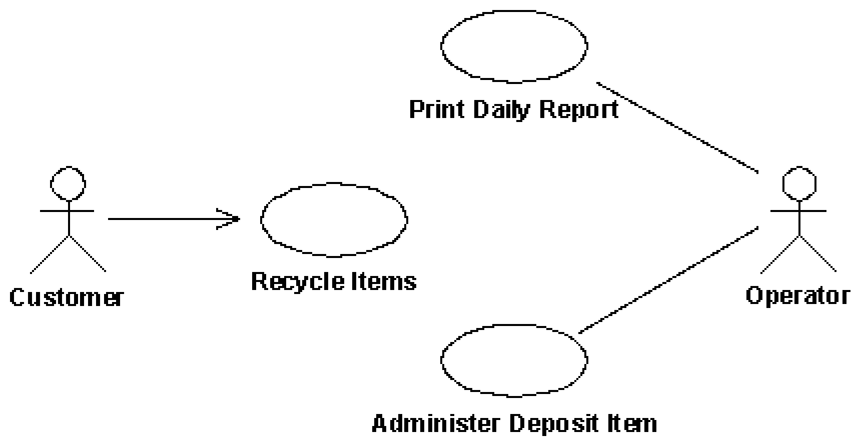
In Business Modeling we document business processes using so-called business use cases. This assures a common understanding among all stakeholders of what business process needs to be supported in the organization. The business use cases are analyzed to understand how the business should support the business processes. This is documented in a business object-model.

Many projects may choose not to do business modeling.

Requirements

The goal of the Requirements workflow is to describe what the system should do and allows the developers and the customer to agree on that description. To achieve this, we elicit, organize, and document required functionality and constraints; track and document tradeoffs and decisions.

A Vision document is created, and stakeholder needs are elicited. Actors are identified, representing the users, and any other system that may interact with the system being developed. Use cases are identified, representing the behavior of the system. Because use cases are developed according to the actor's needs, the system is more likely to be relevant to the users. The following figure shows an example of a use-case model for a recycling-machine system.



An example of a use-case model with actors and use cases.

Each use case is described in detail. The use-case description shows how the system interacts step by step with the actors and what the system does. Non-functional requirements are described in Supplementary Specifications.

The use cases function as a unifying thread throughout the system's development cycle. The same use-case model is used during requirements capture, analysis & design, and test.

Analysis and Design

The goal of the Analysis and Design workflow is to show how the system will be realized in the implementation phase. You want to build a system that:

- Performs-in a specific implementation environment-the tasks and functions specified in the use-case descriptions.
- Fulfills all its requirements.
- Is structured to be robust (easy to change if and when its functional requirements change).

Analysis and Design results in a design model and optionally an analysis model. The design model serves as an abstraction of the source code; that is, the design model acts as a 'blueprint' of how the source code is structured and written.

The design model consists of design classes structured into design packages and design subsystems with well-defined interfaces, representing what will become components in the implementation. It also contains descriptions of how objects of these design classes collaborate to perform use cases. The next figure shows part of a sample design model for the recycling-machine system in the use-case model shown in the previous figure.

Part of a design model with communicating design classes and package group design classes.

The design activities are centered around the notion of architecture. The production and validation of this architecture is the main focus of early design iterations. Architecture is represented by a number of architectural views. These views capture the major structural design decisions. In essence, architectural views are abstractions or simplifications of the entire design, in which important characteristics are made more visible by leaving details aside. The architecture is an important vehicle not only for developing a good design model, but also for increasing the quality of any model built during system development.

Implementation

The purposes of implementation are:

- To define the organization of the code, in terms of implementation subsystems organized in layers.
- To implement classes and objects in terms of components (source files, binaries, executables, and others).
- To test the developed components as units
- To integrate the results produced by individual implementers (or teams), into an executable system.

The system is realized through implementation of components. The Rational Unified Process describes how you reuse existing components, or implement new components with well-defined responsibility, making the system easier to maintain, and increasing the possibilities to reuse.

Components are structured into Implementation Subsystems. Subsystems take the form of directories, with additional structural or management information. For example, a subsystem can be created as a directory or a folder in a file system, or a subsystem in Rational/Apex for C++ or Ada, or packages using Java.

Test

The purposes of testing are:

- To verify the interaction between objects.
- To verify the proper integration of all components of the software.
- To verify that all requirements have been correctly implemented.
- To identify and ensure defects are addressed prior to the deployment of the software.

The Rational Unified Process proposes an iterative approach, which means that you test throughout the project. This allows you to find defects as early as possible, which radically reduces the cost of fixing the defect. Test are carried out along three quality dimensions reliability, functionality, application performance and system performance. For each of these quality dimensions, the process describes how you go through the test lifecycle of planning, design, implementation, execution and evaluation.

Strategies for when and how to automate test are described. Test automation is especially important using an iterative approach, to allow regression testing at the end of each iteration, as well as for each new version of the product.

Deployment

The purpose of the deployment workflow is to successfully produce product releases, and deliver the software to its end users. It covers a wide range of activities including:

- Producing external releases of the software
- Packaging the software
- Distributing the software
- Installing the software
- Providing help and assistance to users

- In many cases, this also includes activities such as:
- Planning and conduct of beta tests
- Migration of existing software or data
- Formal acceptance

Although deployment activities are mostly centered around the transition phase, many of the activities need to be included in earlier phases to prepare for deployment at the end of the construction phase.

The Deployment and Environment workflows of the Rational Unified Process contain less detail than other workflows.

Project Management

Software Project Management is the art of balancing competing objectives, managing risk, and overcoming constraints to deliver, successfully, a product which meets the needs of both customers (the payers of bills) and the users. The fact that so few projects are unarguably successful is comment enough on the difficulty of the task.

This workflow focuses mainly on the specific aspect of an iterative development process. Our goal with this section is to make the task easier by providing:

- A framework for managing software-intensive projects
- Practical guidelines for planning, staffing, executing, and monitoring projects
- A framework for managing risk

It is not a recipe for success, but it presents an approach to managing the project that will markedly improve the odds of delivering successful software.

Configuration and Change Management

In this workflow we describe how to control the numerous artifacts produced by the many people who work on a common project. Control helps avoid costly confusion, and ensures that resultant artifacts are not in conflict due to some of the following kinds of problems:

- Simultaneous Update--When two or more workers work separately on the same artifact, the last one to make changes destroys the work of the former.
- Limited Notification--When a problem is fixed in artifacts shared by several developers, and some of them are not notified of the change.
- Multiple Versions--Most large programs are developed in evolutionary releases. One release could be in customer use, while another is in test, and the third is still in development. If problems are found in any one of the versions, fixes need to be propagated between them. Confusion can arise leading to costly fixes and re-work unless changes are carefully controlled and monitored.

This workflow provides guidelines for managing multiple variants of evolving software systems, tracking which versions are used in given software builds, performing builds of individual programs or entire releases according to user-defined version specifications, and enforcing site-specific development policies.

We describe how you can manage parallel development, development done at multiple sites, and how to automate the build process. This is especially important in an iterative process where you may want to be able to do builds as often as daily, something that would become impossible without powerful automation. We also describe how you can keep an audit trail on why, when and by whom any artifact was changed.

This workflow also covers change request management, i.e. how to report defects, manage them through their lifecycle, and how to use defect data to track progress and trends.

Environment

The purpose of the environment workflow is to provide the software development organization with the software development environment--both processes and tools--that are needed to support the development team.

This workflow focuses on the activities to configure the process in the context of a project. It also focus on activities to develop the guidelines needed to support a project. A step-by-step procedure is provided describing how you implement a process in an organization.

The environment workflow also contains a Development Kit providing you with the guidelines, templates and tools necessary to customize the process.

Certain aspects of the Environment workflow are not covered in the process such as selecting, acquiring, and making the tools work, and maintaining the development environment.

Rational Unified Process — The Product

The Rational Unified Process product consists of:

- A web-enabled searchable knowledge base providing all team members with guidelines, templates, and tool mentors for all critical development activities. The knowledge base can further be broken down to:
    - Extensive guidelines for all team members, and all portions of the software lifecycle. Guidance is provided for both the high-level thought process, as well as for the more tedious day-to-day activities. The guidance is published in HTML form for easy platform-independent access on your desktop.
    - Tool mentors providing hands-on guidance for tools covering the full lifecycle. The tool mentors are published in HTML form for easy platform-independent access on your desktop. See section "Integration with Tools" for more details.
    - Rational Rose® examples and templates providing guidance for how to structure the information in Rational Rose when following the Rational Unified Process (Rational Rose is Rational's tool for visual modeling)
    - SoDA® templates--more than 10 SoDA templates that helps automate software documentation (SoDA is Rational's Document Automation Tool)
    - Microsoft® Word templates--more than 30 Word templates assisting documentation in all workflows and all portions of the lifecycle
    - Microsoft Project Plans--Many managers find it difficult to create project plans that reflect an iterative development approach. Our templates jump start the creation of project plans for iterative development, according to the Rational Unified Process.
    - Development Kit--describes how to customize and extend the Rational Unified Process to the specific needs of the adopting organization or project, as well as provides tools and templates to assist the effort. This development kit is described in more detail later in this section.

    - Access to Resource Center containing the latest white papers, updates, hints, and techniques, as well as references to add-on products and services.
- A book *Rational Unified Process--An Introduction,* by Philippe Kruchten, published by Addison-Wesley. The book is on 277 pages and provides a good introduction and overview to the process and the knowledge base.

Integration with Tools

A software-engineering process requires tools to support all activities in a system's lifecycle, especially to support the development, maintenance and bookkeeping of various artifacts-models in particular. An iterative development process puts special requirements on the tool set you use, such as better integration among tools and round-trip engineering between models and code. You also need tools to keep track of changes, to support requirements traceability, to automate documentation, as well as tools to automate tests to facilitate regression test. The Rational Unified Process can be used with a variety of tools, either from Rational or other vendors. However, Rational provides many well-integrated tools that efficiently support the Rational Unified Process.

Below you find a list of some of Rational's tools that support the Rational Unified Process.

The Rational Unified Process contains Tool Mentors for almost all of these products. A *tool mentor* is a step-by-step guide describing in detail how to operate a tool, (i.e. what menus to launch, what information to enter into dialog boxes, and how to navigate a tool) to carry out an activity within the process. The Tool Mentors allow us to link the tool-independent process to the actual manipulation of the tools in your daily work.

- **Rational Requisite®Pro** — Keeps the entire development team updated and on track throughout the application development process by making requirements easy to write, communicate and change.
- **Rational ClearQuest™** — A Windows and Web-based change-request management product that enables project teams to track and manage all change activities that occur throughout the development lifecycle.
- **Rational Rose 98** — The world's leading visual modeling tool for business process modeling, requirements analysis, and component architecture design.
- **Rational SoDA** — Automates the production of documentation for the entire software development process, dramatically reducing documentation time and costs.
- **Rational Purify®** — A run-time error checking tool for application and component software developers programming in C/C++; helps detect memory errors.
- **Rational Visual Quantify™** — An advanced performance profiling tool for application and component software developers programming in C++, Visual Basic, and Java; helps eliminate performance bottlenecks.
- **Rational Visual PureCoverage™** — Automatically pinpoints areas of code not exercised in testing so developers can thoroughly, efficiently and effectively test their applications.
- **Rational TeamTest** — Creates, maintains and executes automated functional tests, allowing you to thoroughly test your code and determine if your software meets requirements and performs as expected.
- **Rational PerformanceStudio™** — An easy-to-use, accurate and scalable tool that measures and predicts the performance of client/server and Web systems.

- **Rational ClearCase**® — Market-leading software configuration management tool, giving project managers the power to track the evolution of every software development project.

Sources of Framework:

- Trial version is available at http://www.rational.com/tryit/rup/index.jsp

Related Links:

The Ten Essentials of RUP.pdf
Reaching CMM Levels 2 and 3 with the RUP.pdf
Assessing the Rational Unified Process against ISO 15504.pdf
Incorporating the PSP into the RUP.doc
Using Rational Unified Process for Small Projects.pdf
Developing Large-Scale Systems with the Rational Unified Process.pdf
The Rational Unified Process - An Enabler for Higher Process Maturity.pdf