CHAPTER - 5
Advance Topics in Software Engineering

5.1 Software Process Improvement (SPI)

Some software organizations have little more than an ad hoc software process. As they work to improve their software engineering practices, they must address weaknesses in their existing process and try to improve their approach to software work. *Software process improvement* encompasses a set of activities that will lead to a better software process and, as a consequence, higher-quality software delivered in timely manner. The approach to SPI is iterative and continuous, but it can be viewed in five steps:

1. Assessment of the current software process,
2. Education and training of practitioners and managers,
3. Selection and justification of process elements, software engineering methods, and tools,
4. Implementation of the SPI plan, and
5. Evaluation and tuning based on the results of the plan.

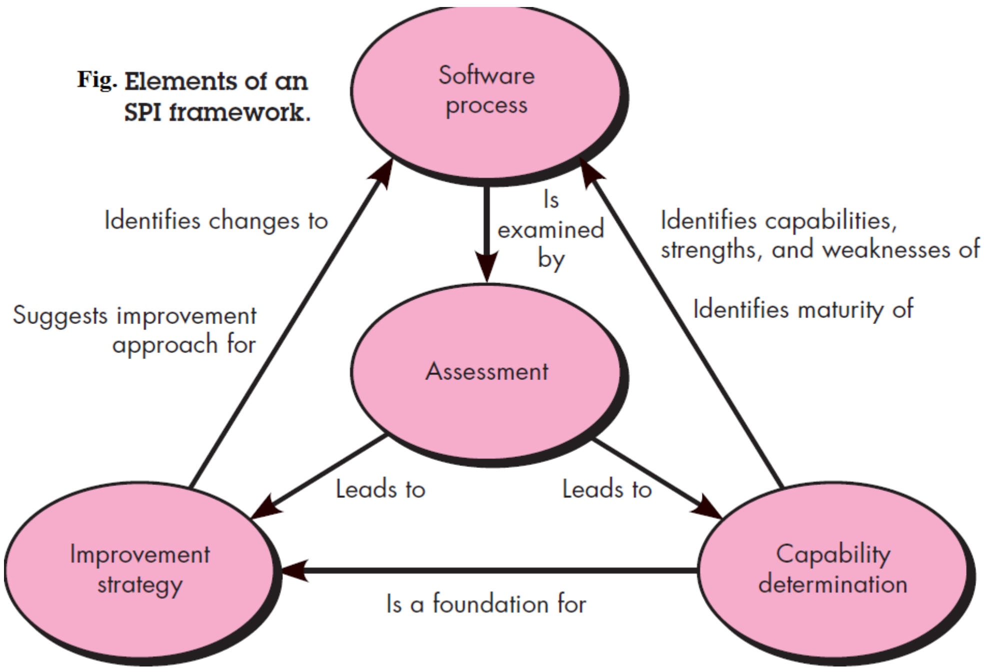
SPI framework

Although an organization can choose a relatively informal approach to SPI, the vast majority choose one of a number of SPI frameworks. An *SPI framework* defines

1. A set of characteristics that must be present if an effective software process is to be achieved,
2. A method for assessing whether those characteristics are present,
3. A mechanism for summarizing the results of any assessment, and
4. A strategy for assisting a software organization in implementing those process characteristics that have been found to be weak or missing.

An SPI framework assesses the "maturity" of an organization's software process and provides a qualitative indication of a maturity level. Figure provides an overview of a typical SPI framework. The key elements of the framework and their relationship to one another are shown.

Fig. Elements of an SPI framework.



SPI Processes

The Software Engineering Institute has developed IDEAL—"an organizational improvement model that serves as a roadmap for initiating, planning, and implementing improvement actions". IDEAL is representative of many process models for SPI, defining five distinct activities—initiating, diagnosing, establishing, acting, and learning—that guide an organization through SPI activities.

The five activities are applied during SPI process in an iterative (cyclical) manner in an effort to foster continuous process improvement.

1. **Assessment and Gap Analysis:** The intent of assessment is to uncover both strengths and weaknesses in the way our organization applies the existing software process and the software engineering practices that populate the process. Assessment examines a wide range of actions and tasks that will lead to a high quality process.
2. **Education and Training:** To follow new software engineering practices, practitioners and managers must provide the required education and training. Generally, three types of education and training should be conducted: *Generic concepts and methods, Specific technology and tools*, and *Business communication and quality-related topics.* In a modern context, education and training also can be delivered from podcasts, to Internet-based training, to DVDs, to classroom courses can be offered as part of an SPI strategy.
3. **Selection and Justification:** *Selection and justification activity* characteristics and specific software engineering methods and tools are chosen to populate the current software process. During this stage first the best process model is chosen for the organization, its stakeholders, and the software that build. Next, develop a work breakdown for each framework activity (e.g., modeling), defining the task set that would be applied for a typical project.
4. **Installation/Migration:** Installation and migration are actually *software process redesign* (SPR) activities. *Installation* is the first point at which a software organization feels the effects of changes implemented as a consequence of the SPI road map. In some cases, an entirely new process is recommended for an organization. Framework activities, software engineering actions, and individual work tasks must be defined and installed as part of a new software engineering culture. In other cases, changes associated with SPI are relatively minor, representing small, but meaningful modifications to an existing process model. Such changes are often referred to as *process migration.*
5. **Evaluation:** The evaluation activity assesses the degree to which changes have been instantiated and adopted, the degree to which such changes result in better software quality or other tangible process benefits, and the overall status of the process and the organizational culture as SPI activities proceed. Both qualitative factors and quantitative metrics are considered during the evaluation activity

The CMMI

The original CMM was developed and upgraded by the Software Engineering Institute throughout the 1990s as a complete SPI framework. Today, it has evolved into the *Capability Maturity Model Integration* (CMMI), a comprehensive process meta-model that is predicated on a set of system and software engineering capabilities that should be present as organizations reach different levels of process capability and maturity.

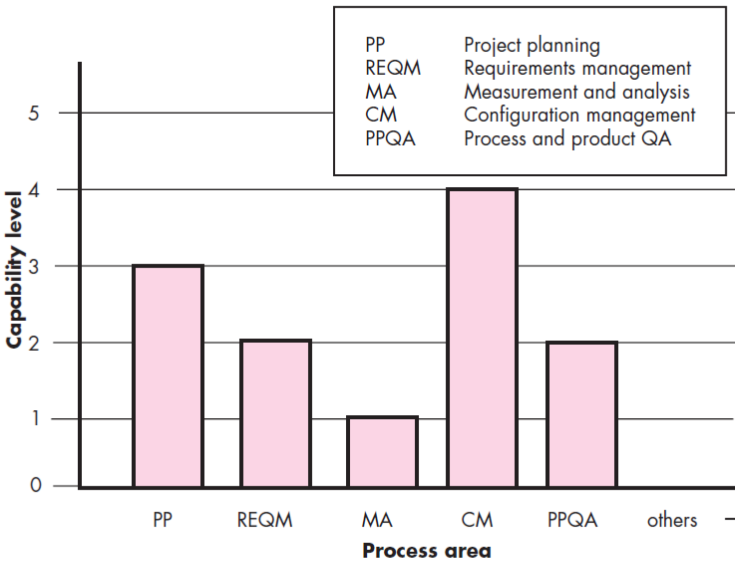


Fig. CMMI process area capability profile.

- **Level 0:** *Incomplete*—the requirements management is either not performed or does not achieve all goals and objectives defined by the CMMI for level 1 capability for the process area.
- **Level 1:** *Performed*—all of the specific goals of the process area (as defined by the CMMI) have been satisfied. Work tasks required to produce defined work products are being conducted.

- **Level 2:** *Managed*—all capability level 1 criteria have been satisfied. In addition, all work associated with the process area conforms to an organizationally defined policy; all people doing the work have access to adequate resources to get the job done; stakeholders are actively involved in the process area as required; all work tasks and work products are "monitored, controlled, and reviewed; and are evaluated for adherence to the process description".
- **Level 3:** *Defined*—all capability level 2 criteria have been achieved. In addition, the process is "tailored from the organization's set of standard processes according to the organization's tailoring guidelines, and contributes work products, measures, and other process-improvement information to the organizational process assets".
- **Level 4:** *Quantitatively managed*—all capability level 3 criteria have been achieved. In addition, the process area is controlled and improved using measurement and quantitative assessment. "Quantitative objectives for quality and process performance are established and used as criteria in managing the process".
- **Level 5:** *Optimized*—all capability level 4 criteria have been achieved. In addition, the process area is adapted and optimized using quantitative (statistical) means to meet changing customer needs and to continually improve the efficacy of the process area under consideration.

The People CMM

A software process will not succeed without talented, motivated software people. The *People Capability Maturity Model* "is a roadmap for implementing workforce practices that continuously improve the capability of an organization's workforce". It was developed in the mid-1990s and refined over the intervening years. The goal of the People CMM is to encourage continuous improvement of generic workforce knowledge, specific software engineering and project management skills, and process-related abilities.

Process areas for the People CMM		
Level	**Focus**	**Process Areas**
Optimized	*Continuous improvement*	Continuous workforce innovation Organizational performance alignment Continuous capability improvement
Predictable	*Quantifies and manages knowledge, skills, and abilities*	Mentoring Organizational capability management Quantitative performance management Competency-based assets Empowered workgroups Competency integration
Defined	*Identifies and develops knowledge, skills, and abilities*	Participatory culture Workgroup development Competency-based practices Career development Competency development Workforce planning Competency analysis
Managed	*Repeatable, basic people management practices*	Compensation Training and development Performance management Work environment Communication and co-ordination Staffing
Initial	*Inconsistent practices*	

Other SPI Frameworks

Although the SEI's CMM and CMMI are the most widely applied SPI frameworks, a number of alternatives have been proposed and are in use. Among the most widely used of these alternatives are:

- **SPICE:** The SPICE (*Software Process Improvement and Capability dEtermination*) model provides an SPI assessment framework that is compliant with ISO 15504:2003 and ISO 12207. The SPICE document suite presents a complete SPI framework including a model for process management, guidelines for conducting an assessment and rating the process under consideration, construction, selection, and use of assessment instruments and tools, and training for assessors.
- **Bootstrap:** The *Bootstrap* SPI framework "has been developed to ensure conformance with the emerging ISO standard for software process assessment and improvement (SPICE) and to align the methodology with ISO 12207". The objective of Bootstrap is to evaluate a software process using a set of software engineering best practices as a basis for assessment. Like the CMMI, Bootstrap provides a process maturity level using the results of questionnaires that gather information about the "as is" software process and software projects. SPI guidelines are based on maturity level and organizational goals.
- **PSP and TSP:** Although SPI is generally characterized as an organizational activity, there is no reason why process improvement cannot be conducted at an individual or team level. Both PSP and TSP emphasize the need to continuously collect data about the work that is being performed and to use that data to develop strategies for improvement.
- **TickIT:** The Ticket auditing method ensures compliance with *ISO 9001:2000 for Software*—a generic standard that applies to any organization that wants to improve the overall quality of the products, systems, or services that it provides. Therefore, the standard is directly applicable to software organizations and companies.

SPI Return on Investment

SPI is hard work and requires substantial investment of dollars and people. Managers who approve the budget and resources for SPI will invariably ask the question: "How do I know that we'll achieve a reasonable return for the money we're spending?"

At a qualitative level, proponents of SPI argue that an improved software process will lead to improved software quality. They contend that improved process will result in the implementation of better quality filters, better control of change, and less technical rework. But can these qualitative benefits be translated into quantitative results? The classic return on investment (ROI) equation is:

$$\text{ROI} = \left[\frac{\Sigma(\textit{benefits}) - \Sigma(\textit{costs})}{\Sigma(\textit{costs})}\right] \times 100\%$$

Where

- *benefits* include the cost savings associated with higher product quality (fewer defects), less rework, reduced effort associated with changes, and the income that accrues from shorter time-to-market.
- *costs* include both direct SPI costs (e.g., training, measurement) and indirect costs associated with greater emphasis on quality control and change management activities and more rigorous application of software engineering methods (e.g., the creation of a design model).

5.2 Emerging Trends in Software Engineering

No one can predict the future with absolute certainty. But it is possible to assess trends in the software engineering area and from those trends to suggest possible directions for the technology. Throughout the relatively brief history of software engineering, practitioners and researchers have developed an array of process models, technical methods, and automated tools in an effort to foster fundamental change in the way we build computer software. Even though past experience indicates otherwise, there is a tacit desire to find the "silver bullet"—the magic process or transcendent technology that will allow us to build large, complex, software based systems easily, without confusion, without mistakes, without delay— without the many problems that continue to plague software work.

But history indicates that our quest for the silver bullet appears doomed to failure. New technologies are introduced regularly, hyped as a "solution" to many of the problems software engineers face, and incorporated into project large and small. Industry pundits stress the importance of these "new" software technologies, the cognoscenti of the software community adopt them with enthusiasm, and ultimately, they do play a role in the software engineering world. But they tend not to meet their promise, and as a consequence, the quest continues.

Technology Evolution

Evolution occurs as a result of positive feedback—"the more capable methods resulting from one stage of evolutionary progress are used to create the next stage".

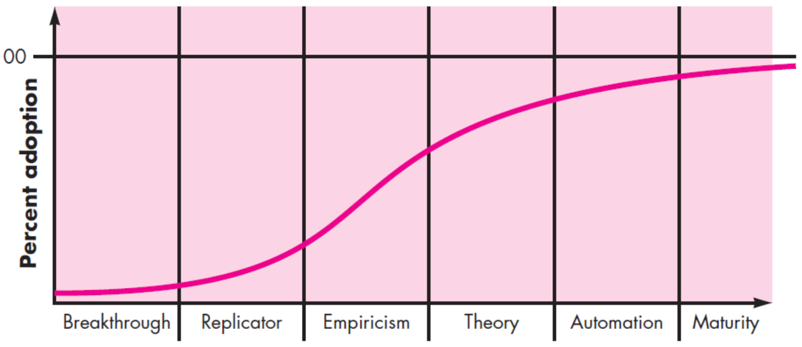


Fig. A technology innovation life cycle

When a successful new technology is introduced, the initial concept moves through a reasonably predictable "innovation life cycle" as shown in figure. In the *breakthrough* phase, a problem is recognized and repeated attempts at a viable solution are attempted. At some point, a solution shows promise. The initial breakthrough work is reproduced in the *replicator* phase and gains wider usage. *Empiricism* leads to the creation of empirical rules that govern the use of the technology, and repeated success leads to a broader *theory* of usage that is followed by the creation of automated tools during the *automation* phase. Finally, the technology matures and is used widely.

Computing technology is evolving at an exponential rate, and its growth may soon become explosive. Kurzweil suggests that within 20 years, technology evolution will accelerate at an increasingly rapid pace, ultimately leading to an era of non-biological intelligence that will merge with and extend human intelligence in ways that are fascinating to contemplate.

Observing Software Engineering Trends

Soft trends have a significant impact on the overall direction of software engineering. In the present context of software engineering, Barry Boehm suggests that "software engineers [will] face the often formidable challenges of dealing with rapid change, uncertainty and emergence, dependability, diversity, and interdependence, but they also have opportunities to make significant contributions that will make a difference for the better."

The Gartner Group—a consultancy that studies technology trends across many industries—has developed a *hype cycle for emerging technologies.* The "hype cycle" presents a realistic view of short-term technology integration. The Gartner Group cycle exhibits five phases:

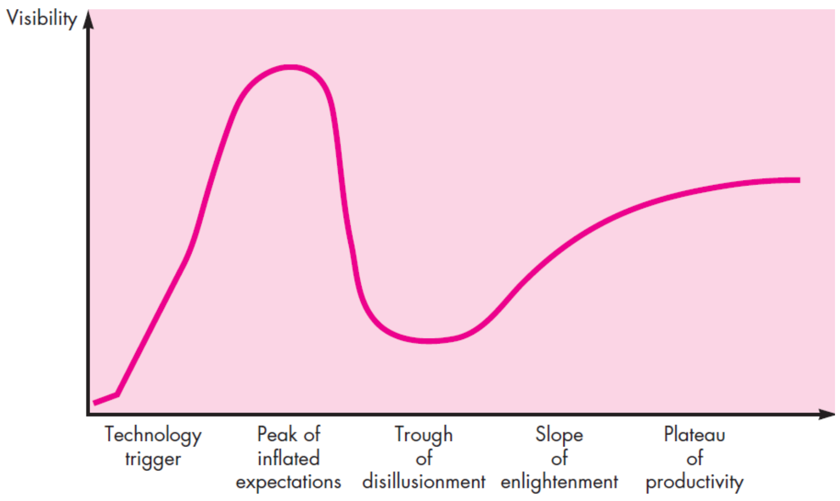


Fig. The Gartner Group's hype cycle for emerging technologies.

- *Technology trigger*—a research breakthrough or launch of an innovative new product that leads to media coverage and public enthusiasm.
- *Peak of inflated expectations*—overenthusiasm and overly optimistic projections of impact based on limited, but well-publicized successes.
- *Disillusionment*—overly optimistic projections of impact are not met and critics begin the drumbeat; the technology becomes unfashionable among the cognoscenti.
- *Slope of enlightenment*—growing usage by a wide variety of companies leads to a better understanding of the technology's true potential; off-the-shelf methods and tools emerge to support the technology.
- *Plateau of productivity*—real-world benefits are now obvious, and usage penetrates a significant percentage of the potential market.

Identifying "Soft Trends"

Each nation with a substantial IT industry has a set of unique characteristics that define the manner in which business is conducted, the organizational dynamics that arise within a company, the distinct marketing issues that apply to local customers, and the overriding culture that dictates all human interaction. *Globalization* leads to a diverse workforce. This, in turn, demands a flexible organizational structure.

It is estimated that over one billion new consumers will enter the worldwide marketplace over the next decade. Consumer spending in "emerging economies will double to well over $9 trillion". In some world regions, the population of software construction is aging and need to transfer the knowledge to new generation. Now the question is that, "Can new software engineering technologies be developed to meet this worldwide demand?"

- **Managing Complexity:** Modern computer-based systems has increasing the LOC from million to billion and that challenge to manage the huge complexity.
- **Open-World Software:** The "open-world software" is the software that is designed to adapt to a continually changing environment "by self-organizing its structure and self-adapting its behavior".
- **Emergent Requirements:** Requirements will emerge as everyone involved in the engineering and construction of a complex system learns more about it, the environment in which it is to reside, and the users who will interact with it. This reality implies a number of software engineering trends. First, process models must be designed to embrace change and adopt the basic tenets of the agile philosophy. Next, methods that yield engineering models must be used judiciously because those models will change repeatedly as more knowledge about the system is acquired. Finally, tools that support both process and methods must make adaptation and change easy.

- **The Talent Mix:** As software-based systems become more complex, as communication and collaboration among global teams becomes commonplace. Each software team must bring a variety of creative talent and technical skills to its part of a complex system, and the overall process must allow the output of these islands of talent to merge effectively.
- **Software Building Blocks:** software engineering philosophy have emphasized the need for reuse—of source code, object-oriented classes, components, patterns, and COTS software. The important issues is that the components must be proven or well tested.
- **Changing Perceptions of "Value":** Modern customers are interested on the software system having speed of delivery, richness of functionality, and overall product quality.
- **Open Source:** The term *open source* when applied to computer software, implies that software engineering work products (models, source code, test suites) are open to the public and can be reviewed and extended (with controls) by anyone with interest and permission. The promise of open source is better quality, higher reliability, more flexibility, lower cost, and an end to predatory vendor lock-in."

Technology Directions

Software engineering will change more rapidly. a few trends in process, methods, and tools that are likely to have some influence on software engineering over the next decade.

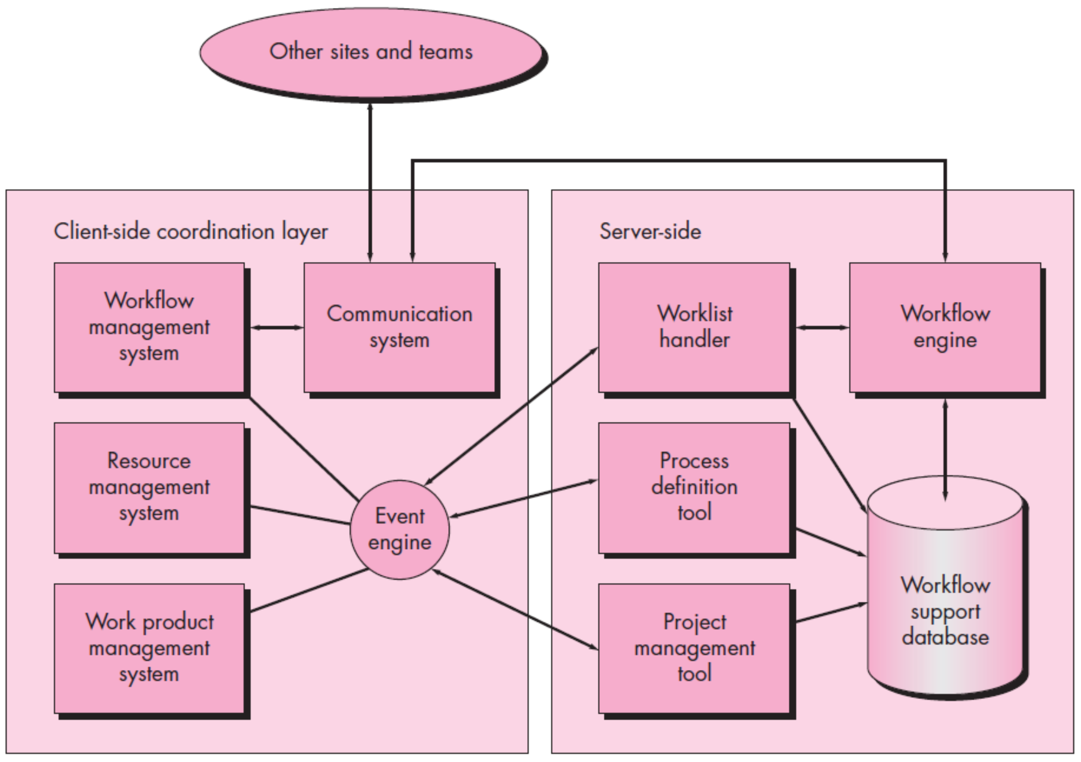
- **Process Trends:** A software procurer's goal is to select the best contractor objectively and rationally. A software company's goal is to survive and grow in a competitive market. An end user's goal is to acquire the software product that can solve the right problem, at the right time, at an acceptable price.
- **The Grand Challenge:** software engineers can meet the daunting challenge of complex software systems development by creating new approaches to understanding system models and using those models as a basis for the construction of high-quality next generation software. The major challenges are: *Multi-functionality, Reactivity and timeliness*, *New modes of user interaction*, *Complex architectures*, *Heterogeneous, distributed systems*, *Criticality or Safety*, and *Maintenance variability*.
- **Collaborative Development:** Collaboration involves the timely dissemination of information and an effective process for communication and decision making. Today, software engineers collaborate across time zones and international boundaries, and every one of them must share information. The same holds for open-source projects in which hundreds or thousands of software developers work to build an open-source app. Again, information must be disseminated so that open collaboration can occur.
- **Requirements Engineering:** Basic requirements engineering actions are elicitation, elaboration, negotiation, specification, and validation. The success or failure of these actions has a very strong influence on the success or failure of the entire software engineering process. Today, most "informal" requirements engineering approaches begin with the creation of user scenarios (e.g., use cases).
- **Model-Driven Software Development:** Model-driven approaches address a continuing challenge for all software developers—how to represent software at a higher level of abstraction than code.
- **Postmodern Design:** Postmodern design will continue to emphasize the importance of software architecture. A designer must state an architectural issue, make a decision that addresses the issue, and then clearly define the assumptions, constraints, and implications that the decision places on the software as a whole.
- **Test-Driven Development:** In *test-driven development* (TDD), requirements for a software component serve as the basis for the creation of a series of test cases that exercise the interface and attempt to find errors in the data structures and functionality delivered by the component. TDD is not really a new technology but rather a trend that emphasizes the design of test cases *before* the creation of source code.

Tools Related Trends

Hundreds of software engineering tools are introduced each year. These claim to improve project management, or requirements analysis, or design modeling, or code generation, or testing, or change management, or any of the many software engineering activities, actions, and tasks. Other tools have been developed as open-source offerings. Future trends in software tools will follow two distinct paths—a *human-focused path* that responds to some of the "soft trends" and a technology-centered path that addresses new technologies.

a. **Tools That Respond to Soft Trends:** The soft trends need to manage complexity, accommodate emergent requirements, establish process models that embrace change, coordinate global teams with a changing talent mix, among others—suggest a new era in which tools support for stakeholder collaboration will become as important as tools support for technology.

   A collaborative software engineering environment (SEE) "supports co-operation and communication among software engineers belonging to distributed development teams involved in modeling, controlling, and measuring software development and maintenance processes. Moreover, it includes an artifact management function that stores and manages software artifacts (work products) produced by different teams in the course of their work". Figure illustrates an architecture for a collaborative SEE.



b. **Tools That Address Technology Trends:** Tools environments will respond to a growing need for communication and collaboration and at the same time integrate domain-specific point solutions that may change the nature of current software engineering tasks. In some instances, modeling and testing tools targeted at a specific application domain will provide enhanced benefit when compared to their generic equivalents.