# The Experience Factory and Its Relationship to Other Quality Approaches

VICTOR R. BASILI

*Institute for Advanced Computer Studies*
*and*
*Department of Computer Science*
*University of Maryland*
*College Park, Maryland*

## Abstract

This chapter describes the principles behind a specific set of integrated software quality improvement approaches which include the Quality Improvement Paradigm, an evolutionary and experimental improvement framework based on the scientific method and tailored for the software business, the Goal/Question/Metric Paradigm, a paradigm for establishing project and corporate goals and a mechanism for measuring against those goals, and the Experience Factory Organization, an organizational approach for building software competencies and supplying them to projects on demand. It then compares these approaches to a set of approaches used in other businesses, such as the Plan–Do–Check–Act, Total Quality Management, Lean Enterprise Systems, and the Capability Maturity Model.

## 1. Introduction

The concepts of quality improvement have permeated many businesses. It is clear that the nineties will be the quality era for software and there is a growing need to develop or adapt quality improvement approaches to the software business. Thus we must understand software as an artifact and software development as a business.

Any successful business requires a combination of technical and managerial solutions. It requires that we understand the processes and products of the business, i.e., that we know the business. It requires that we define our business needs and the means to achieve them, i.e., we must define our process and product qualities. We need to define closed loop processes so that we can feed back information for project control. We need to evaluate every aspect of the business, so we must analyze our successes and failures. We must learn from our experiences, i.e., each project should provide information that allows us to do business better the next time. We must build competencies in our areas of business by packaging our successful experiences for reuse and then we must reuse our successful experiences or our competencies as the way we do business.

Since the business we are dealing with is software, we must understand the nature of software and software development. Some of the most basic premises assumed in this work are that:

The *software discipline is evolutionary and experimental;* it is a laboratory science. Thus we must experiment with techniques to see how and when they really work, to understand their limits, and to understand how to improve them.

*Software is development not production.* We do not produce the same things over and over but rather each product is different from the last. Thus, unlike in production environments, we do not have lots of data points to provide us with reasonably accurate models for statistical quality control.

*The technologies of the discipline are human based.* It does not matter how high we raise the level of discourse or the virtual machine, the development of solutions is still based on individual creativity and human ability will always create variations in the studies.

*There is a lack of models that allow us to reason about the process and the product.* This is an artifact of several of the above observations. Since we have been unable to build reliable, mathematically tractable models, we have tended not to build any. And those that we have, we do not always understand in context.

*All software is not the same; process is a variable, goals are variable, content varies, etc.* We have often made the simplifying assumption that software is software is software. But this is no more true that hardware is hardware is hardware. Building a satellite and a toaster are not the same thing, any more than building a microcode for a toaster and the flight dynamic software for the satellite are the same thing.

Packaged, reusable, experiences require additional resources in the form of organization, processes, people, etc. The requirement that we build packages of

reusable experiences implies that we must learn by analyzing and synthesizing our experiences. These activities are not a byproduct of software development, they require their own set of processes and resources.

## 2. Experience Factory/Quality Improvement Paradigm

The *Experience Factory/Quality Improvement Paradigm* (EF/QIP) (Basili, 1985, 1989; Basili and Rombach, 1987, 1988) aims at addressing the issues of quality improvement in the software business by providing a mechanism for continuous improvement through the experimentation, packaging, and reuse of experiences based on a business's needs. The approach has been evolving since 1976 based on lessons learned in the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) Software Engineering Laboratory (SEL) (Basili *et al.*, 1992).

The basis for the approach is the QIP, which consists of six fundamental steps:

*Characterize* the current project and its environment with respect to models and metrics.

*Set* the quantifiable *goals* for successful project performance and improvement.

*Choose* the appropriate *process* model and supporting methods and tools for this project.

*Execute* the processes, construct the products, collect and validate the prescribed data, and analyze it to provide real-time feedback for corrective action.

*Analyze* the data to evaluate the current practices, determine problems, record findings, and make recommendations for future project improvements.

*Package* the experience in the form of updated and refined models and other forms of structured knowledge gained from this and prior projects and save it in an experience base to be reused on future projects.

Although it is difficult to describe the QIP in great detail here, we will provide a little more insight into the preceding six steps here.

***Characterizing the Project and Environment.*** Based on a set of models of what we know about our business we need to classify the current project with respect to a variety of characteristics, distinguish the relevant project environment for the current project, and find the class of projects with similar characteristics and goals. This provides a context for goal definition, reusable experiences and objects, process selection, evaluation and comparison, and prediction. There are a large variety of project characteristics and environmental factors that need to be modeled and baselined. They include various people factors, such as the number of people, level of expertise, group organization, problem experience, process experience; problem factors, such as the application domain, newness to state of the art, susceptibility to change, problem constraints, etc.;

process factors, such as the life cycle model, methods, techniques, tools, programming language, and other notations; product factors, such as deliverables, system size, required qualities, e.g., reliability, portability, etc.; and resource factors, such as target and development machines, calendar time, budget, existing software, etc.

*Goal Setting and Measurement.* We need to establish goals for the processes and products. These goals should be measurable, driven by models of the business. There are a variety of mechanisms for defining measurable goals: Quality Function Deployment Approach (QFD) (Kogure and Akao, 1983), the Goal/Question/Metric Paradigm (GQM) (Weiss and Basili, 1985), and Software Quality Metrics Approach (SQM) (McCall *et al.*, 1977).

We have used the GQM as the mechanism for defining, tracking, and evaluating the set of operational goals, using measurement. These goals may be defined for any object, for a variety of reasons, with respect to various models of quality, from various points of view, relative to a particular environment. For example, goals should be defined from a variety of points of view: user, customer, project manager, corporation, etc.

A goal is defined by filling in a set of values for the various parameters in the template. Template parameters included purpose (what object and why), perspective (what aspect and who), and the environmental characteristics (where).

### *Purpose:*

Analyze some
(objects: process, products, other experience models)
for the purpose of
(why: characterization, evaluation, prediction, motivation, improvement)

### *Perspective:*

With respect to
(focus: cost, correctness, defect removal, changes, reliability, user friendliness, . . .)
from the point of view of
(who: user, customer, manager, developer, corporation, . . .)

### *Environment:*

In the following context
(problem factors, people factors, resource factors, process factors, . . .)

### *Example:*

Analyze the (system testing method) for the purpose of (evaluation) with respect to a model of (defect removal effectiveness) from the point of view of the (developer) in the following context: the standard NASA/GSFC environment,

i.e., process model (SEL version of the waterfall model, . . .), application (ground support software for satellites), machine (running on a DEC 780 under VMS), etc.

The goals are defined in an operational, tractable way by refining them into a set of quantifiable questions that are used to extract the appropriate information from the models of the object of interest and the focus. The questions and models define the metrics and the metrics, in turn, specify the data that needs to be collected. The models provide a framework for interpretation.

Thus, the GQM is used to (1) specify the goals for the organization and the projects, (2) trace those goals to the data that are intended to define these goals operationally, and (3) provide a framework for interpreting the data to understand and evaluate the achievement of the goals, (4) and support the development of data models based on experience.

***Choosing the Execution Model.*** We need to be able to choose a generic process model appropriate to the specific context, environment, project characteristics, and goals established for the project at hand, as well as any goals established for the organization, e.g., experimentation with various processes or other experience objects. This implies we need to understand under what conditions various processes are effective. All processes must be defined to be measurable and defined in terms of the goals they must satisfy. The concept of defining goals for processes will be made clearer in later chapters.

Once we have chosen a particular process model, we must tailor it to the project and choose the specific integrated set of sub-processes, such as methods and techniques, appropriate for the project. In practice, the selection of processes is iterative with the redefinition of goals and even some environmental and project characteristics. It is important that the execution model resulting from these first three steps be integrated in terms of its context, goals, and processes. The real goal is to have a set of processes that will help the developer satisfy the goals set for the project in the given environment. This may sometimes require that we manipulate all three sets of variables to ensure this consistency.

***Executing the Processes.*** The development process must support the access and reuse packaged experience of all kinds. On the other hand, it needs to be supported by various types of analyses, some done in close to real time for feedback for corrective action. To support this analysis, data needs to be collected from the project. But this data collection must be integrated into the processes—it must not be an add on, e.g., defect classification forms part of configuration control mechanism. Processes must be defined to be measurable to begin with, e.g., design inspections can be defined so that we keep track of the various activities, the effort expended in those activities, such as peer reading, and the effects of those activities, such as the number and types of defects found. This allows us to measure such things as domain understanding (how well the process performer understands the object of study and the application domain) and assures that the processes are well defined and can evolve.

Support activities, such as data validation, education and training in the models, and metrics and data forms are also important. Automated support necessary to support mechanical tasks and deal with the large amounts of data and information needed for analysis. It should be noted, however, that most of the data cannot be automatically collected. This is because the more interesting and insightful data tends to require human response.

The kinds of data collected include: resource data such as, effort by activity, phase, type of personnel, computer time, and calendar time; change and defect data, such as changes and defects by various classification schemes, process data such as process definition, process conformance, and domain understanding; product data such as product characteristics, both logical, e.g., application domain, function, and physical, e.g., size, structure, and use and context information, e.g., who will be using the product and how will they be using it so we can build operational profiles.

*Analyzing the Data.* Based on the goals, we interpret the data that has been collected. We can use this data to characterize and understand, so we can answer questions like "What project characteristics effect the choice of processes, methods and techniques?" and "Which phase is typically the greatest source of errors?" We can use the data to evaluate and analyze to answer questions like "What is the statement coverage of the acceptance test plan?" and "Does the Cleanroom Process reduce the rework effort?" We can use the data to predict and control to answer questions like "Given a set of project characteristics, what is the expected cost and reliability, based upon our history?" and "Given the specific characteristics of all the modules in the system, which modules are most likely to have defects so I can concentrate the reading or testing effort on them?" We can use the data to motivate and improve so we can answer questions such as "For what classes of errors is a particular technique most effective?" and "What are the best combination of approaches to use for a project with a continually evolving set of requirements based on our organization's experience?"

*Packaging the Models.* We need to define and refine models of all forms of experiences, e.g., resource models and baselines, change and defect baselines and models, product models and baselines, process definitions and models, method and technique evaluations, products and product parts, quality models, and lessons learned. These can appear in a variety of forms, e.g., we can have mathematical models, informal relationships, histograms, algorithms, and procedures, based on our experience with their application in similar projects, so they may be reused in future projects. Packaging also includes training, deployment, and institutionalization.

The six steps of the QIP can be combined in various ways to provide different views into the activities. First note that there are two feedback loops, a project feedback loop that takes place in the execution phase and an organizational feedback loop that takes place after a project is completed. The organizational

learning loop changes the organization's understanding of the world by the packaging of what was learned from the last project and as part of the characterization and baselining of the environment for the new project. It should be noted that there are numerous other loops visible at lower levels of instantiation, but these high-level loops are the most important from an organizational structure point of view.

One high-level organizational view of the paradigm is that we must *understand* (characterize), *assess* (set goals, choose processes, execute processes, analyze data), and *package* (package experience). Another view is to *plan* for a project (characterize, set goals, choose processes), *develop* it (execute processes), and then *learn* from the experience (execute processes, analyze data).

## 2.1 The Experience Factory Organization

To support the Improvement Paradigm, an organizational structure called the Experience Factory Organization (EFO) was developed. It recognizes the fact that improving the software process and product requires the continual accumulation of evaluated experiences (*learning*), in a form that can be effectively understood and modified (*experience models*), stored in a repository of integrated experience models (*experience base*), that can be accessed or modified to meet the needs of the current project (*reuse*).

Systematic *learning* requires support for recording, *off-line* generalizing, tailoring, formalizing, and synthesizing of experience. The off-line requirement is based on the fact that reuse requires separate resources to create reusable objects. Packaging and *modeling* useful experience requires a variety of models and formal notations that are tailorable, extendible, understandable, flexible, and accessible.

An effective *experience base* must contain accessible and integrated set of models that capture the *local* experiences. Systematic *reuse* requires support for using existing experience and on-line generalizing or tailoring or candidate experience.

This combination of ingredients requires an organizational structure that supports: a software evolution model that supports reuse, processes for learning, packaging, and storing experience, and the integration of these two functions. It requires separate logical or physical organizations with different focuses and priorities, process models, expertise requirements.

We divide the functions into a *Project Organization* whose focus/priority is product delivery, supported by packaged reusable experiences, and an *Experience Factory* whose focus is to support project developments by analyzing and synthesizing all kinds of experience, acting as a repository for such experience, and supplying that experience to various projects on demand.

The *Experience Factory* packages experience by building informal, formal or schematized, and productized models and measures of various software processes, products, and other forms of knowledge via people, documents, and automated support.

The Experience Factory deals with reuse of all kinds of knowledge and experience. But what makes us think we can be successful with reuse this time, when we have not been so successful in the past. Part of the reason is that we are not talking about reuse of only code in isolation but about reuse of all kinds of experience and of the context for that experience. The Experience Factory recognizes and provides support for the fact that experience requires the appropriate context definition for to be reusable and it needs to be identified and analyzed for its reuse potential. It recognizes that experience cannot always be reused as is, that it needs to be tailored and packaged to make it easy to reuse. In the past, reuse of experience has been too informal, and has not been supported by the organization. It has to be fully incorporated into the development or maintenance process models. Another major issue is that a project's focus is delivery, not reuse, i.e., reuse cannot be a by-product of software development. It requires a separate organization to support the packaging and reuse of local experience.

The Experience Factory really represents a paradigm shift from current software development thinking. It separates the types of activities that need to be performed by assigning them to different organizations, recognizing that they truly represent different processes and focuses. Project personnel are primarily responsible for the planning and development activities—the *Project Organization* (Fig. 1) and a separate organization, the *Experience Factory* (Fig. 2) is primarily responsible for the learning and technology transfer activities. In the *Project Organization*, we are problem solving. The processes we perform to solve a problem consist

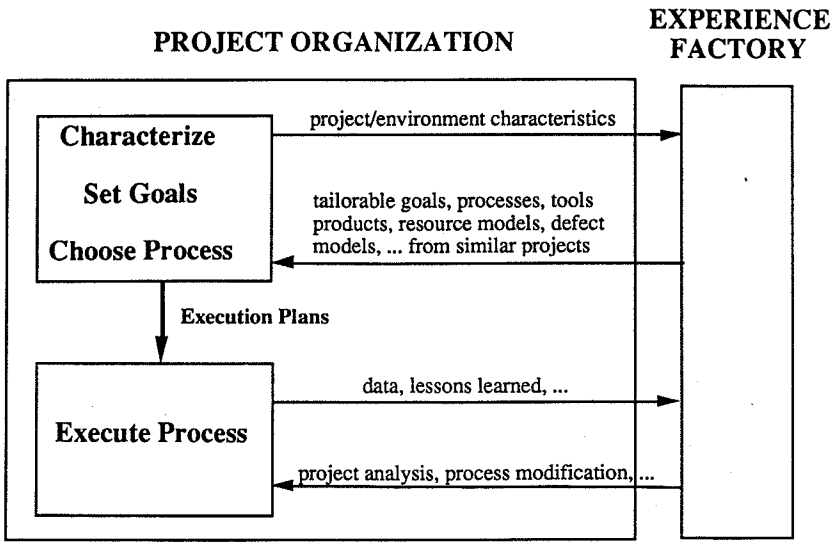



FIG. 1. The Project Organization.

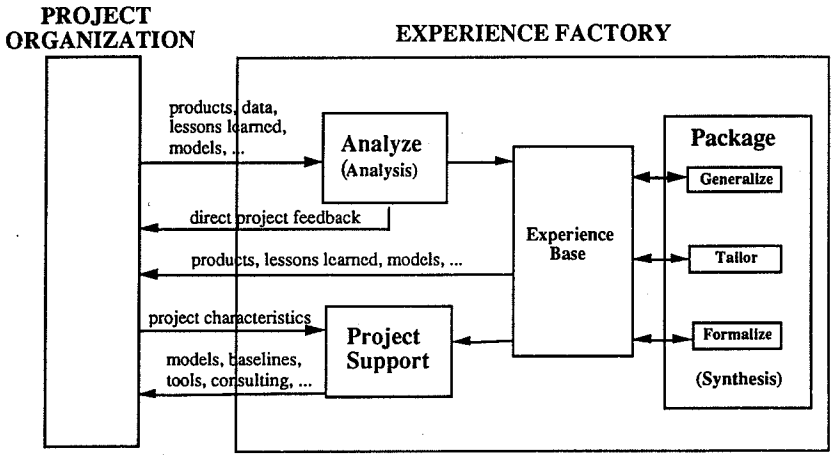



FIG. 2. The Experience Factory.

of the decomposition of a problem into simpler ones, instantiation of higher-level solutions into lower-level detail, the design and implementation of various solution processes, and activities such as validation and verification. In the *Experience Factory,* we are understanding solutions and packaging experience for reuse. The processes we perform are the unification of different solutions and redefinition of the problem, generalization and formalization of solutions in order to abstract them and make them easy to access and modify, an analysis synthesis process enabling us to understand and abstract, and various experimentation activities so we can learn. These sets of activities are totally different.

## 2.2 Examples of Packaged Experience in the SEL

The SEL has been in existence since 1976 and is a consortium of three organizations: NASA/GSFC, the University of Maryland, and Computer Sciences Corporation (McGarry, 1985; Basili *et al.,* 1992). Its goals have been to (1) understand the software process in a particular environment, (2) determine the impact of available technologies, and (3) infuse identified/refined methods back into the development process. The approach has been to identify technologies with potential, apply and extract detailed data in a production environment (experiments), and measure the impact (cost, reliability, quality, etc.).

Over the years we have learned a great deal and have packaged all kinds of experience. We have built resource models and baselines, e.g., local cost models, resource allocation models; change and defect models and baselines, e.g., defect prediction models; types of defects expected for the application, product models, and baselines, e.g., actual vs. expected product size, library access; over time, pro-

cess definitions and models, e.g., process models for Cleanroom, Ada waterfall model; method and technique models and evaluations, e.g., best method for finding interface faults; products and product models, e.g., Ada generics for simulation of satellite orbits; a variety of quality models, e.g., reliability models, defect slippage models, ease of change models; and a library of lessons learned, e.g., risks associated with an Ada development (Basili *et al.,* 1992; Basili and Green, 1994).

We have used a variety of forms for packaged experience. There are equations defining the relationship between variables, e.g., effort = $1.48*KSLOC^{.98}$, number of runs = 108 + 150*KSLOC†; histograms or pie charts of raw or analyzed data, e.g., classes of faults: 30% data, 24% interface, 16% control, 15% initialization, 15% computation; graphs defining ranges of "normal," e.g., graphs of size growth over time with confidence levels; specific lessons learned associated with project types, phases, activities, e.g., reading by stepwise abstraction is most effective for finding interface faults; or in the form of risks or recommendations, e.g., definition of a unit for unit test in Ada needs to be carefully defined; and models or algorithms specifying the processes, methods, or techniques, e.g., an SADT diagram defining design inspections with the reading technique being a variable on the focus and reader perspective.

Note that these packaged experiences are representative of software development in the Flight Dynamics Division at NASA/GSFC. They take into account the local characteristics and are tailored to that environment. Another organization might have different models or even different variables for their models and therefore could not simply use these models. This inability to just use someone else's models is a result of all software not being the same.

These models are used on new projects to help management control development (Valett, 1987) and provide the organization with a basis for improvement based on experimentation with new methods. It is an example of the EF/QIP in practice.

## 2.3 In Summary

How does the EF/QIP approach work in practice? You begin by getting a commitment. You then define the organizational structure and the associated processes. This means collecting data to establish baselines, e.g., defects and resources, that are process and product independent, and then measuring your strengths and weaknesses to provide a business focus and goals for improvement, and establishing product quality baselines. Using this information about your business, you select and experiment with methods and techniques to improve your processes based on your product quality needs and you then evaluate your improvement based on existing resource and defect baselines. You can define and tailor better and more measurable processes, based on the experience and knowledge gained within your own environment. You must measure for process conformance and domain understanding to make sure that your results are valid.

† KSLOC is thousands of source lines of code.

In this way, you begin to understand the relationship between some process characteristics and product qualities and are able to manipulate some processes to achieve those product characteristics. As you change your processes you will establish new baselines and learn where the next place for improvement might be.

The SEL experience is that the cost of the Experience Factory activities amounts to about 11% of the total software expenditures. The majority of this cost (approximately 7%) has gone into analysis rather than data collection and archiving. However, the overall benefits have been measurable. Defect rates have decreased from an average of about 4.5 per KLOC to about 1 per KLOC. Cost per system has shrunk from an average of about 490 staff months to about 210 staff months and the amount of reuse has jumped from an average of about 20% to about 79%. Thus, the cost of running an Experience Factory has more than paid for itself in the lowering of the cost to develop new systems, meanwhile achieving an improvement in the quality of those systems.

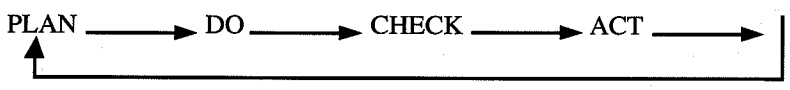
## 3. A Comparison with Other Improvement Paradigms

Aside from the *Experience Factory/Quality Improvement Paradigm,* there have been a variety of organizational frameworks proposed to improve quality for various businesses. The ones discussed here include:

*Plan–Do–Check–Act* is a QIP based on a feedback cycle for optimizing a single process model or production line. *Total Quality Management* represents a management approach to long-term success through customer satisfaction based on the participation of all members of an organization. The *SEI Capability Maturity Model* is a staged process improvement based on assessment with regard to a set of key process areas until you reach level 5 which represents continuous process improvement. *Lean (software) Development* represents a principle supporting the concentration of the production on "value-added" activities and the elimination or reduction of "not-value-added" activities. In what follows, we will try to define these concepts in a little more detail to distinguish and compare them. We will focus only on the major drivers of each approach.

### 3.1 Plan–Do–Check–Act Cycle (PDCA)

The approach is based on work by Shewart (1931) and was made popular by Deming (1986). The goal of this approach is to optimize and improve a single process model/production line. It uses such techniques as feedback loops and statistical quality control to experiment with methods for improvement and build predictive models of the product.



Revise strategy (generate action plans and prioritize KPAs)
For each KPA:
  Establish process action teams
  Implement tactical plan, define processes, plan and execute pilot(s), plan and execute
    Institutionalize
  Document and analyze lessons
  Revise organizational approach

## 3.4 Lean Enterprise Management

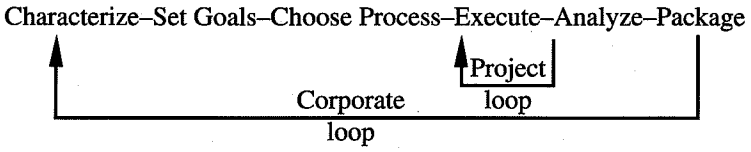
The approach is based on a philosophy that has been used to improve factory output. Womack *et al.* (1990), have written a book on the application of lean enterprises in the automotive industry. The goal is to build software using the minimal set of activities needed, eliminating nonessential steps, i.e., tailoring the process to the product needs. The approach uses such concepts as technology management, human-centered management, decentralized organization, quality management, supplier and customer integration, and internationalization/regionalization.

Given the characteristics for product *V*, select the appropriate mix of subprocesses *pi, qj, rk* . . . to satisfy the goals for *V*, yielding a minimal tailored process *PV* which is composed of *pi, qj, rk* . . .

$$\text{Process } (PV) \rightarrow \text{Product } (V)$$

## 3.5 Comparing the Approaches

As stated above, the Quality Improvement Paradigm has evolved over 17 years based on lessons learned in the SEL (Basili, 1985, 1989; Basili and Rombach, 1987, 1988; Basili *et al.*, 1992). Its goal is to build a continually improving organization based upon its evolving goals and an assessment of its status relative to those goals. The approach uses internal assessment against the organizations own goals and status (rather than process areas) and such techniques as GQM, model building, and qualitative/quantitative analysis to improve the product through the process.




If Processes $(P_X, Q_Y, R_Z, \ldots) \rightarrow$ *Products* $(X, Y, Z, \ldots)$ and we want to build *V*, then based on an understanding of the relationship between $P_X$, $Q_Y$, $R_Z$,

. . . and *X, Y, Z,* . . . and goals for *V* we select the appropriate mix of processes *pi, qj, rk* . . . to satisfy the goals for *V,* yielding a tailored

$$\text{Process } (PV) \rightarrow \text{Product } (V)$$

The EF/QIP is similar to the PDCA in that they are both based on the scientific method. They are both evolutionary paradigms, based on feedback loops from product to process. The process is improved via experiments; process modifications are tried and evaluated and that is how learning takes place.

The major differences are due to the fact that the PDCA paradigm is based on production, i.e., it attempts to optimize a single process model/production line, whereas the QIP is aimed at development. In development, we rarely replicate the same thing twice. In production, we can collect a sufficient set of data based upon continual repetition of the same process to develop quantitative models of the process that will allow us to evaluate and predict quite accurately the effects of the single process model. We can use statistical quality control approaches with small tolerances. This is difficult for development, i.e., we must learn form one process about another, so our models are less rigorous and more abstract. Development processes are also more human based. This again effects the building, use, and accuracy of the types of models we can build. So although development models may be based on experimentation, the building of baselines and statistical sampling, the error estimates are typically high.

The EF/QIP approach is compatible with TQM in that it can cover goals that are customer satisfaction driven and it is based on the philosophy that quality is everyone's job. That is, everyone is part of the technology infusion process. Someone can be on the project team on one project and on the experimenting team on another. All the project personnel play the major role in the feedback mechanism. If they are not using the technology right it can be because they don't understand it, e.g., it wasn't taught right, it doesn't fit or interface with other project activities, it needs to be tailored, or it simply doesn't work. You need the user to tell you how to change it. The EF/QIP philosophy is that no method is "packaged" that hasn't been tried (applied, analyzed, tailored). The fact that it is based upon evolution, measurement, and experimentation is consistent with TQM.

The differences between EF/QIP and TQM are based on the fact that the QIP offers specific steps and model types and is defined specifically for the software domain.

The EF/QIP approach is most similar to the concepts of Lean Enterprise Management in that they are both based upon the scientific method/PDCA philosophy. They both use feedback loops from product to process and learn from experiments. More specifically, they are both based upon the ideas of tailoring a set of processes to meet particular problem/product under development. The goal is to generate an optimum set of processes, based upon models of the

business and our experience about the relationship between process characteristics and product characteristics.

The major differences are once again based upon the fact that LEM was developed for production rather than development and so model building is based on continual repetition of the same process. Thus, one can gather sufficient data to develop accurate models for statistical quality control. Since the EF/QIP is based on development and the processes are human based, we must learn from the application of one set of processes in a particular environment about another set of processes in different environment. So the model building is more difficult, the models are less accurate, and we have to be cautious in the application of the models. This learning across projects or products also requires two major feedback loops, rather than one. In production, one is sufficient because the process being changed on the product line is the same one that is being packaged for all other products. In the EF/QIP, the project feedback loop is used to help fix the process for the particular project under development and it is with the corporate feedback loop that we must learn by analysis and syntheses across different product developments.

The EF/QIP organization is different from the SEI CMM approach, in that the latter is really more an assessment approach rather than an improvement approach.

In the EF/QIP approach, you pull yourself up from the top rather than pushing up from the bottom. At step 1 you start with a level 5 style organization even though you do not yet have level 5 process capabilities. That is, you are driven by an understanding of your business, your product and process problems, your business goals, your experience with methods, etc. You learn from your business, not from an external model of process. You make process improvements based upon an understanding of the relationship between process and product in your organization. Technology infusion is motivated by the local problems, so people are more willing to try something new.

But what does a level 5 organization really mean? It is an organization that can manipulate process to achieve various product characteristics. This requires that we have a process and an organizational structure to help us: understand our processes and products, measure and model the project and the organization, define and tailor process and product qualities explicitly, understand the relationship between process and product qualities, feed back information for project control, experiment with methods and techniques, evaluate our successes and failures, learn from our experiences, package successful experiences, and reuse successful experiences. This is compatible with the EF/QIP organization.

QIP is not incompatible with the SEI CMM model in that you can still use key process assessments to evaluate where you stand (along with your internal goals, needs, etc.). However, using the EF/QIP, the chances are that you will move up the maturity scale faster. You will have more experience early on

operating within an improvement organization structure, and you can demonstrate product improvement benefits early.

## 4. Conclusion

Important characteristics of the EF/QIP process indicate the fact that it is iterative; you should converge over time so don't be overly concerned with perfecting any step on the first pass. However, the better your initial guess at the baselines the quicker you will converge.

No method is "packaged" that hasn't been tried (applied, analyzed, tailored). Everyone is part of the technology infusion process. Someone can be on the project team on one project and on the experimenting team on another. Project personnel play the major role in the feedback mechanism. We need to learn from them about the effective use of technology. If they are not using the technology right it can be because they don't understand it or it wasn't taught right, it doesn't fit/interface with other project activities, it needs to be tailored, or it doesn't work and you need the user to tell you how to change it. Technology infusion is motivated by the local problems, so people are more willing to try something new. In addition, it is important to evaluate process conformance and domain understanding or you have very little basis for understanding and assessment.

The integration of the Improvement Paradigm, the Goal/Question/Metric Paradigm, and the EFO provides a framework for software engineering development, maintenance, and research. It takes advantage of the experimental nature of software engineering. Based upon our experience in the SEL and other organizations, it helps us understand how software is built and where the problems are, define and formalize effective models of process and product, evaluate the process and the product in the right context, predict and control process and product qualities, package and reuse successful experiences, and feed back experience to current and future projects. It can be applied today and evolve with technology.

The approach provides a framework for defining quality operationally relative to the project and the organization, justification for selecting and tailoring the appropriate methods and tools for the project and the organization, a mechanism for evaluating the quality of the process and the product relative to the specific project goals, and a mechanism for improving the organization's ability to develop quality systems productively. The approach is being adopted by several organizations to varying degrees, such as Motorola and HP, but it is not a simple solution and it requires long-term commitment by top-level management.

In summary, the QIP approach provides for a separation of concerns and focus in differentiating between problem solving and experience modeling and packaging. It offers a support for learning and reuse and a means of formalizing and integrating management and development technologies. It allows for the generation of a tangible corporate asset: an experience base of software competencies. It offers a Lean Enterprise Management approach compatible with TQM

while providing a level 5 CMM organizational structure. It links focused research with development. Best of all you can start small, evolve and expand, e.g., focus on a homogeneous set of projects or a particular set of packages and build from there. So any company can begin new and evolve.

## References


Basili, V. R. (1985). Quantitative evaluation of software engineering methodology. *In* "Proceedings of the 1st Pan Pacific Computer Conference, Melbourne, Australia" (also available as Technical Report TR-1519, Department of Computer Science, University of Maryland, College Park, 1985).

Basili, V. R. (1989). Software development: A paradigm for the future. *In* "Proceedings of the 13th Annual International Computer Software and Applications Conference (COMPSAC), Keynote Address, Orlando, FL."

Basili, V. R., and Green, S. (1994). Software process evolution at the SEL. *IEEE Software Mag.*, July, pp. 58–66.

Basili, V. R., and Rombach, H. D. (1987). Tailoring the software process to project goals and environments. *In* "Proceedings of the 9th International Conference on Software Engineering, Monterey, CA," pp. 345–357.

Basili, V. R., and Rombach, H. D. (1988). The TAME Project: Towards improvement-oriented software environments. *IEEE Trans. Software Eng.* **SE-14**(6), 758–773.

Basili, V. R., Caldiera, G., McGarry, F., Pajerski, R., Page, G., and Waligora, S. (1992). The software engineering laboratory—an operational software experience factory. *In* "Proceedings of the International Conference on Software Engineering," pp. 370–381.

Crosby, P. B. (1980). Quality is Free: The art of making quality certain. New American Library, New York.

Deming, W. E. (1986). "Out of the Crisis." MIT Center for Advanced Engineering Study, MIT Press, Cambridge MA.

Feigenbaum, A. V. (1991). "Total Quality Control," 40th Anniv. Ed. McGraw-Hill, New York.

Humphrey, W. S. (1989). "Managing the Software Process," SEI Ser. Software Eng. Addison-Wesley, Reading, MA.

Kogure M., and Akao, Y. (1983). Quality function deployment and CWQC in Japan. *Qual. Prog.*, October, pp. 25–29.

Likert, R. (1967). "The Human Organization: Its Management and Value." McGraw-Hill, New York.

McCall, J. A., Richards, P. K., and Walters, G. F. (1977). "Factors in Software Quality," RADC TR-77-369.

McGarry, F. E. (1985). "Recent SEL Studies," Proc. 10th Annu. Software Eng. Workshop. NASA Goddard Space Flight Center, Greenbelt, MD.

Paulk, M. C., Curtis, B., Chrissis, M. B., and Weber, C. V. Capability Maturity Model for Software, Version 1.1, Technical Report SEI-93-TR-024.

Radice, R., Harding, A. J., Munnis, P. E., and Phillips, R. W. (1985). A programming process study. *IBM Syst. J.* **24**(2).

Shewhart, W. A. (1931). "Economic Control of Quality of Manufactured Product." Van Nostrand, New York.

Software Engineering Institute (1933). "Capability Maturity Model," CMU/SEI-93-TR-25 Version 1.1. Carnegie-Mellon University, Pittsburgh, PA.

Valett, J. D. (1987). The dynamic management information tool (DYNAMITE): Analysis of the prototype, requirements and operational scenarios. M.Sc. Thesis, University of Maryland, College Park.

Weiss, D. M., and Basili, V. R. (1985). Evaluating software development by analysis of changes: Some data from the software engineering laboratory. *IEEE Trans. Software Eng.* **SE-11**(2), 157–168.

Womack, J. P., Jones, D. T., and Roos, D. (1990). "The Machine that Changed the World: Based on the Massachusetts Institue of Technology 5-Million Dollar 5-Year Study on the Future of the Automobile." Rawson Associates, New York.