THE EXPERIENCE FACTORY

Victor R. Basili - Gianluigi Caldiera
University of Maryland

H. Dieter Rombach
Universitat Kaiserlautern

Reprinted from
Encyclopedia of Software Engineering - 2 Volume Set
ISBN #1-54004-8

EXPERIENCE FACTORY

INTRODUCTION

Reuse of products, processes and experience originating from the system life cycle is seen today as a feasible solution to the problem of developing higher quality systems at a lower cost. In fact, quality improvement is very often achieved by reusing, and modifying over and over the same elements, learning about them by direct experience.

This article presents an infrastructure, called the *experience factory*, aimed at capitalization and reuse of life cycle experience and products. The experience factory is a logical and physical organization, and its activities are independent from the ones of the development organization. The activities of the development organization and of the experience factory can be outlined in the following way:

- The *development organization,* whose mission is to develop and deliver systems, provides the experience factory with product development and environment characteristics, data, and a diversity of models (resources, quality, product, process) currently used by the projects in order to deliver their capabilities.
- The *experience factory,* through processing this information and other state-of-the-practice notions, will return direct feedback to each project activity, together with goals and models tailored from previous project increments. It will also produce, store and provide upon request baselines, tools, lessons learned, data, all presented from a more generalized perspective.

MOTIVATION

Any successful business requires a combination of technical and managerial solutions. These include:

- A well-defined set of product needs to satisfy the customer, assist the developer in accomplishing those needs and create competencies for future business.
- A well-defined set of processes to accomplish what needs to be accomplished, control development, and improve the overall business.
- A closed-loop process that supports learning and feedback.

The key technologies for supporting these requirements include: modeling, measurement, and the reuse of processes, products, and other forms of knowledge relevant to the business.

In order to be successful in the software business there are some basic requirements (Basili, 1985, 1989; E. Deming, 1986). First, we must understand the software process and product. Second, we must define our business needs if we are to achieve them, i.e., we must define process and product qualities. Third, we must evaluate every aspect of the business, i.e., we need to evaluate our various successes and failures. Fourth, we must have a closed-loop process, i.e., we must feed back information for project control. Fifth, each project should provide information that allows us to do business better, i.e., we must learn from our experiences. Sixth, we must build competencies in our areas of business, i.e., we must package and reuse units of experience relevant to our business to be able to achieve more in the future.

Almost any business today involves the development or use of software. It is either the main aspect of the business, the value added to the product, or on the critical path to project success. It permeates every aspect of life. If an organization is not investing heavily in the basic aspects of the software business then it will not be competitive and may not be in the business in the future.

Part of the problem with the software business is the lack of understanding of the nature of software and software development. To some extent, software is different from most products. First of all, software is developed in the creative, intellectual sense, rather than produced in the manufacturing sense. Each software system is developed rather than manufactured. Second, there is a nonvisible nature to software. Unlike an automobile or a television set, it is hard to see the structure or the function of software.

The software business requires understanding, continuous improvement, and the packaging of experience for reuse. There are certain concepts that have become understood with regard to software:

- There are factors that create similarities and differences among projects; this means that one model for software development does not work in all situations.
- There is a direct relationship between process and product; this means one must choose the right processes to create the desired product characteristics.
- Measurement is necessary and must be based on the appropriate goals and models; that is, appropriate measurement provides visibility.
- Evaluation and feedback are necessary for project control: this means a closed loop process for project control is needed.
- Software development follows an experimental paradigm, thus, learning and feedback are natural activities for software development and maintenance.
- Process, product, knowledge, and quality models need to be better defined and tailored; the components of the software business have an evolutionary nature and must be defined according to it.
- Evaluation and feedback are necessary for learning; a closed loop for long range improvement, as well as for individual project control, is needed.
- New technologies must be continually introduced; organizations and researchers need to experiment with technologies.
- Reusing experience in the form of processes, products, and other forms of knowledge is essential for improvement, that is, reuse of knowledge is the basis of improvement.
- Experience needs to be packaged; organizations must build competencies in software.
- Experiences must be evaluated for reuse potential; an analysis process is required.
- Software development and maintenance processes must support reuse of experience, where reuse must be defined in terms of what, how and when to reuse.
- A variety of experiences can be packaged: process, product, resource, defect and quality models can be developed and updated based on experience.
- Experiences can be packaged in a variety of ways; we can use equations, histograms, algorithms, etc. as mechanisms for packaging experience.
- Packaged experiences need to be integrated; an experience base, is a repository of integrated information, relating similar projects, products, characteristics, phenomena, etc.

To address the business needs of software, software equipment offers a framework based on an evolutionary quality management paradigm tailored for the software business, the *Quality Improvement Paradigm.* The Paradigm is supported by a tool for establishing project and corporate goals and a mechanism for measuring against those goals, the *Goal Question Metric Paradigm* (qv), and by an organizational approach for building software competencies and supplying them to projects, the *Experience Factory Organization.*

The rest of this article will define and discuss the Experience Factory Organization concept, after a preliminary discussion of its basic methodological device, the Quality Improvement Paradigm.

THE QUALITY IMPROVEMENT PARADIGM

The Quality Improvement Paradigm is the basic methodological device for the Experience Factory, and as such it is a basic component of our discussion. Therefore it is useful to take a closer look at some of the issues associated with it and with its phases.

The Quality Improvement Paradigm developed by Basili and co-workers (1985) is the result of the application of the scientific method to the problem of software quality improvement. As such it is related to the Shewart-Deming Cycle Plan/Do/Check/Act (Deming, 1986) widely used in the industry for the implementation of quality management plans.

The Quality Improvement Paradigm is articulated into the following six steps (Fig. 1):

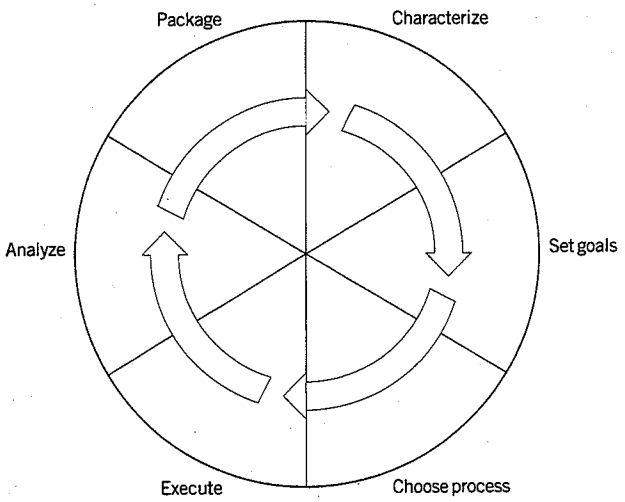

Figure 1. The six steps of the Quality Improvement Paradigm.

1. *Characterize.* Understand the environment based upon available models, data, intuition, etc. Establish baselines with the existing business processes in the organization and characterize their criticality.
2. *Set Goals.* On the basis of the initial characterization and of the capabilities that have a strategic relevance to the organization, set quantifiable goals for successful project and organization performance and improvement. The reasonable expectations are defined based upon the baseline provided by the characterization step.
3. *Choose Process:* On the basis of the characterization of the environment and of the goals that have been set, choose the appropriate processes for improvement, and supporting methods and tools, making sure that they are consistent with the goals that have been set.
4. *Execute:* Perform the processes constructing the products and providing project feedback based upon the data on goal achievement that are being collected.
5. *Analyze:* At the end of each specific project, analyze the data and the information gathered to evaluate the current practices, determine problems, record findings, and make recommendations for future project improvements.
6. *Package:* Consolidate the experience gained in the form of new, or updated and refined, models and other forms of structured knowledge gained from this and prior projects, and store it in an experience base so it is available for future projects.

The Quality Improvement Paradigm implements two feedback cycles:

1. The project feedback cycle (control cycle) is the feedback that is provided to the project during the execution phase: whatever the goals of the organization, the project used as a pilot should use its resources in the best possible way; therefore quantitative indicators at project and task level are useful in order to prevent and solve problems.
2. The corporate feedback cycle (capitalization cycle) is the feedback that is provided to the organization and has the double purpose of providing analytical information about project performance at project completion time by comparing the project data with the nominal range in the organization and analyzing concordance and discrepancy. Accumulating reusable experience in the form of software artifacts that are applicable to other projects and are, in general, improved based on the performed analysis.

An appropriate and unambiguous characterization of the environment is a prerequisite to a correct application of the paradigm. This characterization requires that we classify the current project with respect to a variety of characteristics. It allows us to isolate the class of projects with similar characteristics and goals to the project being developed. This way we can distinguish the relevant project environment for the current project. Characterization provides a context for goal definition, reuse of experience and products, process selection, evaluation and comparison, and prediction.

There are a large number of project and environmental characteristics that affect the software development process and product (Basili, 1981; Basili and co-workers, 1986). These include 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; process factors, such as life cycle models, methods, techniques, tools, programming language, other notations; product factors, such as deliverables, system size, required qualities, e.g., reliability, portability; and resource factors, such as target and development machines, calendar time, budget, existing software.

A realistic definition of the goals is an important correlate to the characterization of the environment. We need to establish models and goals for the processes and products. These goals should be measurable, driven by models, and defined from a variety of perspectives, e.g., the user, customer, project, corporation. There are several techniques for defining measurable goals: the Quality Function Deployment Approach (QFD) (Kogure, 1983), the Goal/Question/Metric Paradigm (GQM) Basili and Weiss, 1984, and the Software Quality Metrics Approach (SQM) Boehm and co-workers, 1976.

The Goal/Question/Metric Paradigm is the mechanism used by the Quality Improvement Paradigm for defining and evaluating a set of operational goals using measurement. It represents a systematic approach for tailoring and integrating goals with models of the software processes, products and quality perspectives of interest, based upon the specific needs of the project and the organization.

The choice of the process execution model involves choosing and tailoring an appropriate genetic life cycle model, a set of methods, and techniques. It should be noted that choosing and tailoring any form of process involves providing its goal and procedure definition. Choosing and

tailoring are always performed in the context of the environment, project characteristics, and goals established for products and processes.

For the purpose of discussion, we define and differentiate the terms technique, method, life cycle model:

- *Technique* is a basic algorithm, or set of steps to be followed in constructing or assessing the software. For example, code reading by stepwise abstraction is a technique for assessing the code.
- *Method* is an organized approach based upon applying some technique. A method has associated with it a technique, as well as a set of guidelines about how and when to apply the technique, when to stop applying it, when the technique is appropriate and how we can evaluate it. For example, a method will have associated with it a set of entry and exit criteria and a set of management supports. Code Inspection is a method, based upon some reading technique, which has a well-defined set of entry and exit criteria as well as a set of management functions defined for how to manipulate the technique.
- *Life cycle model* as an integrated set of methods that covers the entire life cycle of a software product.

There are a variety of software life cycle models, each of which is useful under different circumstances (Basili and Turner, 1975; Boehm, 1988; Royce, 1970). The waterfall model (Royce, 1970) is basically a sequential process model where each of the major documents are developed in sequence, starting with the most abstract, i.e. the requirements document. The waterfall model is most efficient when the problem is well defined and the solution is well understood, that is, when there are not a lot of iterations through the cycle. If the problem or the solution are not well-defined, other process models may be more effective. The iterative enhancement model (Basili and Turner, 1975) is an incremental model that builds several versions of the system, each with more functionality. It starts with a simple initial implementation of a subset of the problem and iteratively enhances the existing version until the full system is implemented. At each step of the process, not only extensions but design modifications are made, based upon what we have learned about the problem and the solution. This process model results in several versions of the system. Iterative enhancement is effective when the problem or solution are not well understood, schedule for full function a risk, or the requirements changing over time. It allows the developer to learn through each cycle of development and the user to provide timely essential feedback, improving each version until the final version of the system is produced. The spiral model (Boehm, 1988) is another incremental model that organizes the activities like a spiral with many cycles: the radial dimension of the spiral represent the cost of the system, the angular dimension represents the progress of the project. At each stage of the development, the, model requires the identification of uncertainties and asks involved, and the development of strategies for resolving them. Another effective process model is prototyping. It involves the development of an experimental version of some aspect of the system. The prototype is typically built in a very high level language or using some modeling or simulation tools. It provides a better specification of the problem requirements and is effective when the user is unsure of the system needs, some aspect of the system is unclear, or an experimental version of the system is needed for analysis.

In order to execute the processes, experience should be accessible in a packaged form as processes that have been chosen, prior products available for reuse, appropriate resource and data models, and software development models that allow to take advantage of the reusable packages. One needs to analyze the data according to the project specific models and goals in close to real time in order to make the project visible to management and feed back information for corrective action. Data collection must be integrated into the processes, not considered as an add on, e.g., the defect classification form should be part of the configuration control mechanism. Data validation is important to assure we are making decisions based upon valid data. It is clear that automation is necessary to support some mechanical tasks, deal with the large amounts of data and information, and aid in the data analysis.

There is a wide variety, of data that can be collected. Resource data include effort by activity, phase, type of personnel, computer time, and calendar time. Change and defect data include changes and defects by various classification schemes. Process measurement includes process definition, process conformance, and domain understanding data. Product data includes product characteristics, both logical, (e.g., application domain, function) and physical (e.g. size, structure) and use and context information.

THE EXPERIENCE FACTORY

The Quality Improvement Paradigm is based upon the notion 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) into a repository of integrated experience models (experience base) that can be accessed and modified to meet the needs of the current project (reuse). The paradigm implies the logical separation of project development (performed by the Project Organization) from the systematic learning and packaging of reusable experiences (performed by the Experience Factory) (Basili, 1989).

The Experience Factory is a logical and/or physical organization that supports 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 (Fig. 2). It 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 development organization, whose goal is to produce and maintain software, provides the analysis organization with project and environment characteristics, development data, resource usage information, quality records, and process information. It also provides feedback on the actual performance of the models processed by the experience factory and utilized by the project.

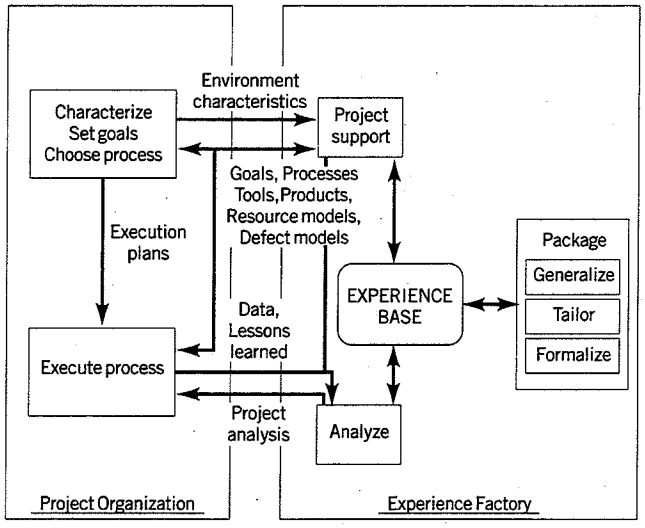

Figure 2. Experience Factory.

The analysis organization, by processing this information received from the development organization, will return direct feedback to each project, together with goals and models tailored from similar projects. It also produces and provides upon request baselines, tools, lessons learned, and data, parametrized in some form in order to be adapted to the specific characteristics of a project.

The support organization sustains and facilitates the interaction between developers and analysts, by saving and maintaining the information, making it efficiently retrievable, and controlling and monitoring the access to it.

The analysis and interpretation of the data collected is based upon the goals. We can use this data to:

- Characterize and understand, (e.g., what project characteristics affect the choice of processes, methods and techniques? which phase is typically the greatest source of errors?).
- Evaluate and analyze, (e.g., what is the statement coverage of the acceptance test plan? does the Inspection Process reduce the rework effort?).
- Predict and control, (e.g., given a set of project characteristics, what is the expected cost and reliability, based upon our history).
- Motivate and improve, (e.g., for what classes of errors is a particular technique most effective?).

Systematic learning requires support for recording experience, off-line generalizing and tailoring of experience, and formalizing experience. Packaging useful experience requires a variety of models and formal notations that are tailorable, extendible, understandable, and accessible.

An effective experience base contains an accessible and integrated set of analyzed, synthesized, and packaged experience models that capture past experiences. Systematic reuse requires support for using existing experience, and necessary generalizing or tailoring of candidate experience. This combination of ingredients requires an organizational structure that supports them. This includes: a software evolution model that supports reuse, a set of processes for learning, packaging, and storing experience, and the integration of these two functions. The experience factory is the organizational unit that performs this integration.

It is important to understand that the term "reuse" is used here to mean more than product reuse. Reuse, in the software domain, has been a long sought after goal with little historical success. Why has reuse been a problem in the software domain? There are several reasons. First we need to reuse more than just code, we need to reuse the context from which the code originates. Second, the reuse of experience is too informal, not fully incorporated into the development or maintenance process models. Third, experience has not yet been analyzed and evaluated for it reuse potential, nor has been appropriately packaged. Fourth, it is often assumed that reuse means reuse as is. On the contrary most experiences need to be tailored in some way to meet the needs of a new context. Lastly, it is also often assumed that reusable packages of experience, be it product, process or any other form of experience, could be developed as a by-product of the project. Clearly this is not the case. The project focus is delivery, not reuse. If we want reuse, the activities that create reusable packages of experience need to be outside of the project.

The packaging of experience is based on tenets and techniques that are different from the problem solving activity used in project development.

In a correct implementation of the experience factory paradigm projects and factory will have different process models: each project will choose its process model based upon the characteristics of the software product that will be delivered, while the experience factory will define (and change) its process model based upon organizational and performance issues.

This creates the need for separate organizations, at least from a logical point of view: the Project Organization for product development and the Experience Factory for packaging experience. Both organizations have different focuses and priorities, use different process models, and have different expertise requirements. Trying to mix them in the same organization is destined to failure (Caldiera and Basili, 1991).

In practice, the Quality Improvement Paradigm/Experience Factory Organization approach is implemented by first putting the organization in place. This means collecting data to establish baselines, e.g., defects and resources, that are process and product independent, and measuring the strengths and weaknesses of the organization in order to provide business focus and goals for the improvement process. The initial data collection is also critical for establishing the product quality baseline that should be improved through the program. Using this information, the organization selects methods and techniques to improve process and products, and experiments with them. Better and measurable processes can be defined and tailored based on the experience and knowledge gained within the project environments. The results are always validated with respect to process conformance and domain understanding.

When the relationship between some process characteristics and product qualities within a specific environment is better understood, the organization is ready to manipulate its processes to achieve those product characteristics.

Changes in processes establish new baselines and provide new goals for improvement.

In this way the organization defines itself as continuously improving, even if its maturity level is not very high, because it learns from its own business, not from an external, ideal process model. Process improvements are based upon the understanding of the relationship between process and product in the specific organization. Technology infusion is motivated by the local problems, and people are more willing to try something new.

EXAMPLES OF PACKAGED EXPERIENCE

The experience factory can package all kinds of experience. It can build resource models and baselines, change and defect models and baselines, product models and baselines, process definitions and models, method and technique models and evaluations, products and product models, a library of lessons learned, and a variety of quality models.
There are a variety of forms for packaged experience:

- Equations defining the relationship between variables, (e.g., Effort=a*Size^b).
- Histograms or pie charts of raw or analyzed data, (e.g. % of each class of fault).
- Graphs defining ranges of "normal" (e.g., graphs of size growth over the 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).
- Models or algorithms specifying the processes, methods, or techniques, (e.g. an SADT diagram defining Design Inspections with the reading technique as variable dependent upon the focus and reader perspective).

Examples of resource models and baselines include cost models, resource allocation models for staffing, schedule, and computer utilization, and the relationship between resources and various factors that affect resources, e.g. specific methods, customer complexity, the application, the environment, and defect classes (Basili and Beane, 1981).

In building resource models, the experience factory is interested in capturing data associated with a variety of factors associated with prior projects, e.g., size, effort, pages of documentation, calendar time. These relationships can be used to build equations that define the empirical relationships between these factors. Using these data it can either generate separate equations representing characteristically different environments (based upon characterizing factors), or it can use the characterizing factors to adjust the equations to provide better fits to the range of data. These relationships package the organization's experience with respect to various resources. The characterizing factors also provide insight into those factors that effect resources. The equations can be used for prediction, project monitoring, and evaluation.

Examples of change and defect models and baselines developed by an experience factory include: change baselines by various classifications, defect baselines by various classifications, defect prediction models, reliability models (Weiss and Basili, 1985).

An appropriate analysis can capture the number of errors, faults and failures associated with various phases, e.g., in total, by various classes (error domain, time of detection, omission/commission, software aspect, failure by resolution date opened/date closed). Histograms and Pareto charts are built to analyze the various defect classes in order to identify overall patterns as well as common patterns representing characteristically different environments. These defect distributions package the organization's experience with respect to defects associated with various project characteristics. They can be used for prediction, project monitoring, evaluation, and provide specific focuses for improvement.

Project characteristic models and baselines developed by an experience factory include: growth and change histories for size, staffing, computer use, number of test cases, test coverage, etc. These can be compared with the norm for the environment or for a set of characteristically similar projects to make predictions and estimates for the current project and provide guidance for the project manager based upon variation from expectation (McGarry, 1985).

For a variety of data the experience factory can define graphs of various variables over time. The accumulation of such data over a variety of projects provides baselines for planning and monitoring projects. For example, it can plot: growth in lines of code vs. schedule, CPU hours vs. calendar time, the amount of code covered vs. number of tests run, the amount of reuse in each project over time.

The experience factory packages experiences with techniques, methods, and life cycle models by defining and refining models of their definitions and goals, understanding where they are appropriate and how they need to be tailored to a particular set of environmental characteristics. To focus on improvement, the organization needs to introduce new technology. It needs to experiment and record the findings in terms of lessons learned and eventually adjustments to the current processes. When the technology is substantially different from what we are currently using, the experimentation may be off-line. It may take the form of a controlled experiment (for detailed evaluation in the small) or of a case study (to study the scale effects). In both cases, the Goal/Question/Metric paradigm provides an important framework (Basili and co-workers, 1986).

Based upon experimentation, a set of lessons learned can be written that can be made available in the experience base for future uses of the technology. New methods and techniques can be defined or old ones refined.

The main product of the experience factory is the *Experience Package.* The content and the structure of an experience package vary based upon the kind of experience clustered in the package. There is a central element that determines what the package is: a software life cycle product or process, a mathematical relationship, an empirical or theoretical model, a data base, etc. The experience packages are defined by the life cycle product. Examples of experience packages are:

1. *Product Packages* have as their central element a lifecycle product, clustered with the information needed to reuse it and the lessons learned in reusing it. Examples: Programs, Architectures, Designs.
2. *Process Packages* have as their central element a lifecycle process, clustered with the information needed to execute it and the lessons learned in executing it. Examples: Process models, Methods.
3. *Relationship Packages* have as their central element a relationship or a system of relationships among observable characteristics of a software project. There are time based relationships and time independent relationships. In any case, these packages are used for analysis and/or forecast of relevant phenomena. Examples: Cost and Defect Models, Resource Models.
4. *Tool Packages* have as their central element a specific tool, either constructive (Examples: Code Generator, Configuration Management Tool) or analytic (Examples: Static Analyzer, Regression Tester)
5. *Management Packages* have as their central element any container of reference information for project management. Examples: Management Handbooks, Decision Support Models.
6. *Data Packages* have as their central element a collection of defined and validated data relevant for a software project or for activities within it. Examples: Project databases, Quality records.

EXAMPLES OF EXPERIENCE FACTORIES

A software organization that has for a long time recognized the value of accumulation and reuse of experience is the NASA Goddard Space Flight Center which has developed since 1977 the Software Engineering Laboratory, in conjunction with the Department of Computer Science of the University of Maryland and with the Computer Sciences Corporation.

The Software Engineering Laboratory (SEL) is today a very advanced example of the concept of experience factory. The experience packages developed by the SEL have mainly focused on project management and control, acquisition and tailoring of new technologies for software development and maintenance. The SEL has produced several types of experience packages, specific to its application domain, flight dynamics applications (Druffel and co-workers, 1983).

One of the most interesting experience packages developed by the SEL is the Software Management Environment (SME): a set of data, tools, manuals and analysis techniques supplied to the project management in order to control the execution of a project, compare it with similar ones, detect and analyze problems, identify solutions. An essential part of the SME package is the SEL database, an on-line information base for storage and retrieval of software engineering data.

Another interesting experience package developed by the SEL is built around the Cleanroom process model proposed by Mills. Here the analysts produce the requirements and the developers produce and verify designs and code that are then released to the testers who run statistical tests and release a system with certified reliability. Systematic design, implementation without testing, and statistical testing are the cornerstones of the Cleanroom model. The relevant existing models in the experience base include: the standard SEL models, the IBM/FSD Cleanroom Model (Mills, 1987), and a Cleanroom model used for a controlled experiment at the University of Maryland (Selby and co-workers, 1987).

The U.S. Department of Defense has supported several efforts in the direction of the accumulation of software experience, especially in the field of code reuse. The original goal of the STARS (Software Technology for Adaptable and Reliable Systems, 1983–1995) Program (Druffel and Redwine, 1983) is the expansion of the technological basis available to a software project. Several products of the STARS program can be seen as experience packages. The RAPID Center, developed by SofTech for the US Army Information Systems Engineering Command (ISEC), operates as an experience factory in the area of composition technologies for software reuse Guerrieri, 1988. The RAPID Center supports the activities of other ISEC development centers by

- Developing and maintaining a repository of reusable software components.
- Assisting the development centers in the selection and use of the needed reusable components.
- Assisting the development centers in the design, implementation and identification of reusable components.
- Providing methods, tools and strategies for maximizing reuse and associated benefits.
- Collecting data and measuring the program progress.

The major software industries, Japanese (Toshiba, Hitachi, NEC, Fujitsu) and American (GTE) have identified experience accumulation and reuse as a strategic objective (Cusumano, 1991). The Japanese software factories have chosen a strategy based on small improvements, slowly but constantly introducing new technologies, tools and organizational solutions that in many cases can be seen as partial implementations of the software factory. The scenarios targeted by specific quality improvement and experience packaging programs in those software factories have been: code reuse, reliability modeling, productivity improvement, service quality enhancement.

EXPERIENCE FACTORY IMPLICATIONS

The Experience Factory offers an organizational structure that separates the product development focus from the learning and reuse focus. It supports learning and reuse and generates a tangible corporate asset in the form of packaged experiences. It aids in the formalization of management and development processes. It links focused research with development.

The Experience Factory can be used to consolidate and integrate activities, such as packaging experience, consulting, quality assurance, education and training, process and tool support, and measurement and evaluation.

The Experience Factory makes existing technologies more relevant, e.g., verification techniques for product

packaging. It forces research to focus on corporate needs and technology transfer. Areas of research for supporting the activities include defining and tailoring models, the integration of technologies, scaling-up techniques and methods, building and accessing the experience base, and automation.

It makes existing education in formalism, models and notations more relevant. It requires education in verification technologies, formal requirements and specification notations, formal models of measurement and management, and assessment technologies.

How the experience factory is funded depends upon the organizational structure of the corporation. Clearly the project organization and the experience factory should be separate cost centers, initially funded out of corporate overhead. However, eventually one would like to have projects billed for packages, so that the factory can be self-supporting and focused toward project support.

There are costs involved in instituting such a program. The level of funding clearly depends upon the size of the program. However, some relative data is available. Based upon the SEL experience where a full measurement program has been in progress for over 14 years, project data collection overhead is estimated to be about 5% of the total project cost. Although our experience shows that this typically does not affect total project cost, since the data collection activity pays for itself on the first project in terms of improvement, it must be established as an up-front cost. With regard to the Experience Factory, the costs depend upon the number of projects supported, level of effort and set of activities performed, e.g., quality assurance, process definition, tool building, education and training, etc. One might consider that it takes a minimum of two people, however to create the critical mass necessary to develop such and activity at the minimal level.

BIBLIOGRAPHY

V. R. Basili, "Data Collection, Validation, and Analysis," in *Tutorial on models and Metrics for Software Management and Engineering,* IEEE Catalog No. EHO-167-7, 1981, pp. 310–313.

R. Basili, "Quantitative Evaluation of Software Engineering Methodology," *Proceedings of the First Pan Pacific Computer Conference,* Melbourne, Australia, Sept. 1985.

R. Basili, "Software Development: A Paradigm for the Future," *Proceedings of the 13th Annual International Computer Software & Applications Conference (COMPSAC),* Keynote Address, Orlando, Fla., Sept. 1989.

R. Basili and J. Beane, "Can the Parr Curve help with the Manpower Distribution and Resource Estimation Problems," *Journal of Systems and Software,* **2** (1), 47–57 (1981).

R. Basili, R. W. Selby, and D. H. Hutchens, "Experimentation in Software Engineering," *IEEE Transactions on Software Engineering,* **SE-12**(7), 733–743 (July 1986).

R. Basili and A. J. Turner, "Iterative Enhancement: A Practical Technique for Software Development," *IEEE Transactions on Software Engineering* **SE-1**(4) (Dec. 1975).

R. Basili and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Transactions on Software Engineering,* **SE-10**(6), 728–738 (Nov. 1984).

W. Boehm, "A Spiral Model of Software Development and Enhancement," *IEEE Computer,* 61–72 (May 1988).

W. Boehm, J. R. Brown, and M. Lipow, "Quantitative Evaluation of Software Quality," *Proceedings of the Second International Conference on Software Engineering,* 1976, pp. 592–605.

Brophy, W. Agresti, and V. R. Basili, "Lessons Learned in Use of Ada Oriented Design Methods," *Proceedings of the Joint Ada Conference,* Arlington, Va., March 16–19, 1987.

G. Caldiera and V. R. Basili, "Identifying and Qualifying Reusable Components," *IEEE Software,* 61–70 (Feb. 1991).

A. Cusumano, *Japan's Software Factories,* Oxford University Press, New York, Feb. 1991.

E. Deming, *Out of the Crisis,* MIT Center for Advanced Engineering Study, MIT Press, Cambridge, Mass., 1986

E. Druffel, S. T. Redwine, and W. E. Riddle, "The STARS Program: Overview and Rationale," *IEEE Computer,* 21–29 (Nov. 1983).

Guerrieri, "Searching for Reusable Software Components with the RAPID Center Library System," in *Proceedings of the Sixth National Conference on Ada Technology,* March 14–18, 1988, pp. 395–406.

Kogure and Y. Akao, "Quality Function Deployment and CWQC in Japan," *Quality Progress,* 25–29 (Oct. 1983).

E. McGarry, "Recent SEL Studies," *Proceedings of the 10th Annual Software Engineering Workshop,* NASA Goddard Space Flight Center, Dec. 1985.

McGarry and R. Pajerski, "Towards Understanding Software - 15 Years in the SEL," *Proceedings of the 15th Annual Software Engineering Workshop,* NASA Goddard Space Flight Center, Greenbelt, MD, Software Engineering Laboratory Series, SEL-90-006, Nov. 1990.

D. Mills, M. Dyer, and R. C. Linger, "Cleanroom Software Engineering," *IEEE Software,* 19–25 (Sept. 1987).

W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques," *Proceedings of the WESCON,* Aug. 1970.

W. Selby, Jr., V. R. Basili, and T. Baker, "CLEANROOM Software Development: An Empirical Evaluation," *IEEE Transactions on Software Engineering,* **13**(9), 1027–1037 (Sept. 1987).

M. Weiss and V. R. Basili, "Evaluating Software Development by Analysis of Changes: Some Data from the Software Engineering Laboratory," *IEEE Transactions on Software Engineering,* **SE-11**(2) 157–168 (Feb. 1985).

VICTOR R. BASILI
GIANLUIGI CALDIERA
University of Maryland
H. DIETER ROMBACH
Universität Kaiserlautern