UMIACS–TR–89–48
CS–TR–2244

May, 1989

Maintenance = Reuse–Oriented Software Development†

Victor R. Basili‡
Institute for Advanced Computer Studies
Department of Computer Science
University of Maryland
College Park, MD 20742

ABSTRACT

In this paper, we view maintenance as a reuse process. In this context, we discuss a set of models that can be used to support the maintenance process. We present a high level reuse framework that characterizes the object of reuse, the process for adapting that object for its target application, and the reused object within its target application. Based upon this framework, we offer a qualitative comparison of the three maintenance process models with regard to their strengths and weaknesses and the circumstances in which they are appropriate. To provide a more systematic, quantitative approach for evaluating the appropriateness of the particular maintenance model, we provide a measurement scheme, based upon the reuse framework, in the form of an organized set of questions that need to be answered. To support the reuse perspective, a set of reuse enablers are discussed.

† This research was supported in part by NASA grant NSG–5123 and ONR grant N00014–87–K–0307 to the University of Maryland.

‡ Keynote address, conference on Software Maintenance, Phoenix, AZ, October 1988.

Maintenance = Reuse-Oriented Software Development

Victor R. Basili
Institute for Advanced Computer Studies
Department of Computer Science
University of Maryland

[Keynote Address, Conference on Software Maintenance,
Phoenix, AZ, October 1988.]

Abstract

In this paper, we view maintenance as a reuse process. In this context, we discuss a set of models that can be used to support the maintenance process. We present a high level reuse framework that characterizes the object of reuse, the process for adapting that object for its target application, and the reused object within its target application. Based upon this framework, we offer a qualitative comparison of the three maintenance process models with regard to their strengths and weaknesses and the circumstances in which they are appropriate. To provide a more systematic, quantitative approach for evaluating the appropriateness of the particular maintenance model, we provide a measurement scheme, based upon the reuse framework, in the form of an organized set of questions that need to be answered. To support the reuse perspective, a set of reuse enablers are discussed.

Introduction

If we take the view that software should be developed with the goal of maximizing the reuse of prior experience in the form of knowledge, processes, products and tools, then the maintenance process is logically ideally suited to a reuse-oriented software development process. There are a variety of reuse models. The key issue here is which process model is best suited to the particular maintenance problem at hand.

In this paper, we present a high level organizational paradigm for software development and maintenance in which an organization can learn from prior and current development and maintenance tasks and then apply that paradigm to several maintenance process models. The paradigm has associated with it a mechanism for setting goals that can be measured so that the organization can evaluate the process and the product and learn from its experience for future projects or enhancements of the current project.

We begin by identifying three process models that can be used for maintenance. We then present a high level reuse framework that characterizes the object of reuse, the process for adapting that object for its target application, and the reuse object within its target application. Based upon this framework, we offer a qualitative comparison of the three maintenance process models with regard to their strenghts and weaknesses and the circumstances in which they are appropriate. To provide a systematic, quantitative approach for evaluating the appropriateness of the particular maintenance model, we provide a measurement scheme, using the Goal/Question/Metric Paradigm. Since reuse requires a supportive environemnt, a set of environmental reuse enablers are discussed.

Maintenance

The nature of software is that it can be modified without the use of physical tools such as screw drivers and soldering irons. This has lead to the false assumption that maintenance is easy and inexpensive. Clearly nothing could be further from the truth.

Most software systems are complex and modification requires a deep understanding of the functional and non-functional requirements, the mapping of functions to system components, and the interaction of the

components. Without good documentation of the requirements, design and code with respect to function, traceability and structure, maintenance becomes a difficult, expensive, error-prone task. As early as 1976, Belady and Lehman reported on the problems with the evolution of OS 360 [7]. The literature is filled with similar experiences and lessons learned [10,12,16,18,20].

Maintenance consists of several different types of activities: correction of faults existing in the system, the adaptation of the system to a changing operating environment, e.g., new terminals, operating system modifications, etc., and changes to the original requirements. The new system is like the old system but different in a specific set of characteristics. One can view the new version of the system as a modification of the old system or a new system which reuses many of the components of the old system. Although these two views have many aspects in common, they are quite different with respect to the process models used and their effects on future environments.

In fact, we can identify at least three process models associated with maintenance depending upon the characteristics of the modification. We will call these (1) the quick fix model, (2) the iterative enhancement model, and (3) the full reuse model. All three models reuse the old system and so are reuse-oriented. Which model should be chosen for any particular modification is a combination of management and technical decisions.

Quick Fix Model. The quick fix model involves taking the existing system, usually just the code, and making the necessary changes to the source code and the accompanying documentation, e.g. requirements, design, and recompiling the system as a new version. This may be as straightforward as a change to some internal component, e.g. an error correction involving a single component or a structural change or even some functional enhancement. Here reuse is implicit.

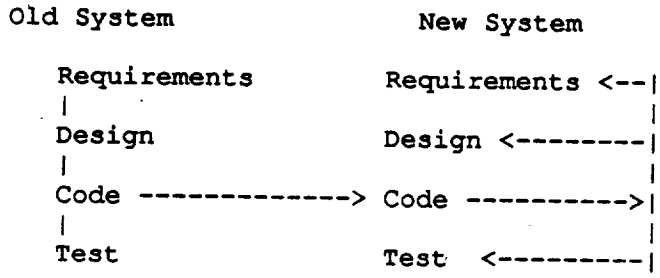

Figure 1. Quick Fix Process Model

Iterative Enhancement Model. Iterative Enhancement [5] is an evolutionary model which was proposed for software development in environments where the complete set of requirements for a system were not fully understood or the author did not know how to build the full system. Although it was proposed as a development model, it is well suited to maintenance. The process model involves:

1. Starting with the existing system requirements, design, code, test and analysis documents
2. Redeveloping starting with the appropriate document based upon analysis of the existing system, propagating the changes through the full set of documents
3. At each step of the evolutionary process, continuing to redesign, based upon analysis.

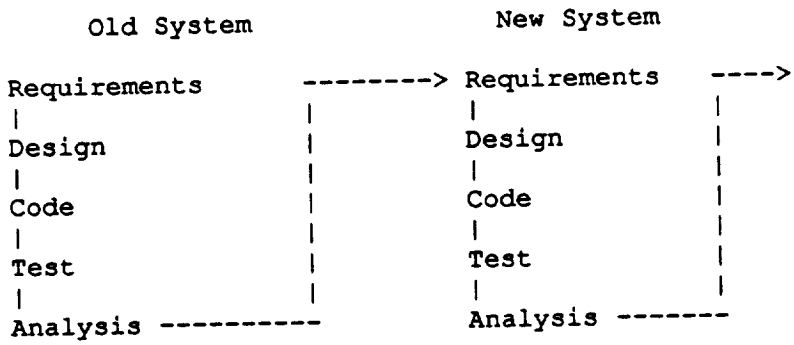

Figure 2. Iterative Enhancement Model

To view this as a maintenance model, assume the initial implementation is the system in its current state in the evolutionary maintenance process. The process assumes that the maintenance organization has the ability to analyze the existing product, characterize the proposed set of modifications, and redesign the current version where necessary for the new capabilities. Again, reuse is implicit.

Full Reuse Process Model. While iterative enhancement starts with evaluating the existing system for redesign and modification, a full reuse process model starts with the requirements analysis and design of the new system, with the concept of reusing whatever requirements, design and code are available from the old system. The reuse process model involves:

1. Starting the requirements for the new system, reusing as much of the old system as feasible
2. Building a new system using components from the old system or other systems available in the repository developing new components where appropriate.

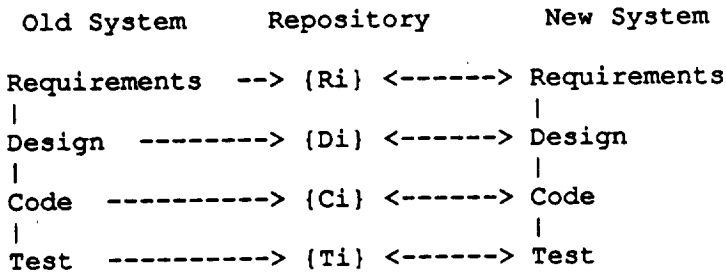

Figure 3. Full Reuse Process Model.

Here reuse is explicit, packaging of prior components is necessary and analysis is required for the selection of the appropriate components.

The difference between the last two approaches is more one of perspective. The full reuse model frees the developer to design the solution relative to the available set of solutions of similar systems. The iterative enhancement model takes the last version of the current system and enhances it. Both approaches encourage redesign, but the full reuse model suggests a wider forum and can lead to the development of more reusable components for future systems where the iterative enhancement model suggests the tailoring of an existing system for the given extensions.

A Reuse Framework

The existence of several models for maintenance raises several questions. Which is the most appropriate model for a particular environment? a particular system? a particular set of changes? the task at hand? How do I improve each step in the process model I have chosen? How do I minimize overall cost and maximize overall quality?

In order to answer these questions we need a model of the object of reuse, the process of adapting that object for its target application, and the reused object within its target application. A simple model for reuse is given in figure 4. In that model, an object is any software process or product and a transformation is the set of activities that are performed in reusing that object. Given that the scope of the new application are understood, the steps are:

1. identifying the candidate reusable pieces of the old object
2. understanding them
3. modifying them to our needs
4. integrating them into the process

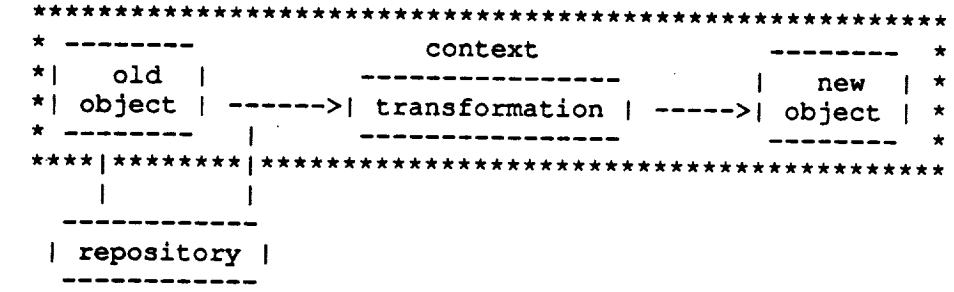

Figure 4. A Simple Reuse Model

To flesh out the model, we need a framework for categorizing objects, transformations, and context. The framework should cover various categories e.g. reuse object: process, product. Within each category there are various classification schemes for product: e.g. requirements document, code module, test plan, and process: e.g. cost estimation, risk analysis, design.

There are a variety of approaches to reuse and schemes that classify the object of reuse [8,9,11,13,15,17,19]. We use here a variation of a reuse framework [4] that captures several aspects of the reuse process, product and context.

Object dimensions include:

(1) **Reuse Object Type**: What is a characterization of the candidate reuse object? Sample categories and classifications are: process (e.g. design, test) and product (e.g. application, tool).
(2) **Self-Containedness**: How independent and understandable is the candidate reuse object? Sample categories and classifications are: syntactic independence (e.g. tightly coupled), semantic independence (e.g. similar functionality), and precision of specification (e.g. formal, informal).
(3) **Reuse Object Quality**: How good is the candidate reuse object? Sample categories and classifications are: maturity (e.g. newly developed, used in one application) complexity (e.g. low cyclomatic complexity) reliability (e.g. no failures during prior use).

Context dimensions include:

(1) **Requirements Domain**: How similar are the requirements domains of the candidate reuse objects and current or future projects? Some example categories and classifications are: application (e.g. ground support software for satellites), distance (e.g. same application, similar algorithms/different problem focus).
(2) **Solution Domain**: How similar are the evoloution process which resulted in the candidate reuse objects and the ones used in the current and future projects? Some example categories and classifications are: process model (e.g. waterfall model) design method (e.g. function decomposition) programming language (e.g. FORTRAN).
(3) **Knowledge Transfer Mechanism**: How is information about the candidate reuse objects and their context passed to current and future projects? An example classification is: humans (e.g. subset of the development team doing maintenance, separate team doing maintenance).

Transformation dimensions include:

(1) **Type of Transoformations**: How do we characterize the transformation activities to be performed? Some sample categories and classifications are: Percent of Change (e.g. 0%, 5%), Direction (e.g. general to domain specific, project specific to domain specific), mechanism of modification (e.g. verbatim, parameterization, template-based, unconstrained) and mechanism for identifying (e.g. by name, by functional requirements).
(2) **Activity Integration**: How do we integrate the transformation activities into the new system development? Some sample categories and classifications are: phase activity performed in the new development planning (e.g. cost estimation, risk analysis), construction (e.g. requirements development), analysis (e.g. testing).
(3) **Transformed Quality**: What is the contribution of the reuse object in the context of the new system with respect to the objectives set for it? Sample category and classifications are: reliability (e.g. no failures associated with that component) and performance (e.g. satisfies the timing requirement).

Comparing the Models Using the Reuse Framework

When applying the reuse framework to the maintenance process, we are focusing on a set of reuse objects that are product documents. Let us compare the various models according to the dimensions given.

Consider the reuse object dimensions:

With regard to reuse object type, the object of the quick fix and iterative enhancement models is the set of documents representing the old system. The object of the full reuse model is the repository including the old system.

With regard to self-containedness, all the models depend upon the unit of change. The quick fix model depends upon how much evolution has taken place since entropy may have unstructured the system. In iterative enhancement, the evolved system should be improving for the specific application and for the appropriate set of changes, the unit of change should be more visible. In the full reuse model, the evolved system should be improving with respect to reuse object independence for the general application, depending upon the quality and maturity of the repository.

With regard to reuse object quality, the quick fix model offers little knowledge of the quality of the old object. In iterative enhancement, the analysis phase provides a fair assessment of quality with respect to the particular application. In full reuse, we have an assessment of quality of the reuse object across several systems.

Consider the context dimensions:

With regard to the requirements domain, the quick fix and iterative enhancement model assume the same application, in fact the same project. The full reuse model allows for manageable variation in the application domain, depending upon what is available in the repository.

With regard to the solution domain, the quick fix model assumes the same solution structure exists during maintenance as during development. There is no change in the basic design or structure of the new system. In the iterative enhancement model, because redesign is a part of the model, there is some modification to the solution structure allowed. The full reuse model allows major differences in the solution structure, i.e. a complete redesign is possible going from functional decomposition to object oriented design.

With regard to knowledge transfer mechanism, the quick fix model and iterative enhancement work best with the same people. The full reuse model can compensate for having a different team, assuming we have application specialists and a well documented reuse object repository.

Consider the transformation dimensions:

With regard to type of activities, the quick fix model typically uses a source code look-up, reading for understanding, unconstrained modification and re-compilation approach. Iterative enhancement typically begins with a search through the highest relevant document, changing it and continuing through the subsequent documents using a variety of modification mechanisms. The full reuse is uses a library search, and a variety of modification mechanisms depending upon the type of change. Here modification is done off-line.

With regard to activity integration, in the quick fix model, all activities are performed at same time. Iterative enhancement associates the activities with all the normal development phases. In the full reuse model, identification of the candidate reusable pieces is done during project planning and the other activities are done during development.

With regard to transformed quality, the quick fix model usually works best on small well-contained modifications since their affect on the system can be understood and verified in context. Iterative enhancement is more appropriate for larger changes where the analysis phase can provide better assessment of the full effect of changes. Full reuse is appropriate for large changes and major redesigns. Here, analysis and prior history of the performance of the reuse objects support quality.

Given these differences, we can provide some analysis of the various maintenance process models and recommend where they might be most applicable. But first, let's discuss the relationship between the development and maintenance process models. In some sense development can be considered a subset of maintenance. Maintenance environments differ from development environments with regard to the constraints on the solution, customer demand, timeliness of response, and organization.

Most maintenance organizations are set up for the quick fix model but not for the iterative enhancement or reuse process models. This is because they are responding to timeliness, e.g. a system failure needs to be fixed immediately, or a customer demand, e.g. a modification of the functionality of the system. Clearly these are strengths for the quick fix model. But the weaknesses of the model are that the modification is usually a patch, not well documented, the structure of the system has been partly destroyed which makes future evolution of the system difficult and error-ridden, and it is not compatible with development processes. This model is best used when timeliness and customer need are dominant and there is little chance the system will be modified again.

The iterative enhancement model allows for redesign so the structure of the system evolves and future modification is easier. It focuses on the particular system, making it as good as possible. It is compatible with development process models. The drawbacks are that it is a more costly and possibly less timely approach (in the short run) than the quick fix model and it provides little support for generics or future similar systems. It is a good approach to use when the product will have a long life and evolve over time. In this case, if timeliness is also a constraint, the quick fix model can be used as a patch and the iterative enhancement model can be used for the long term change, replacing the patch.

The full reuse process model provides the maintainer with a broader perspective, focuses on long range development for a set of products and has the side effect of creating reusable components of all kinds for future developments. It is compatible with development process models, and in fact, it is the way we would like such models to evolve. The drawback is that it is more costly in the short run, is not appropriate for small modifications but can be used in conjunction with other models. It is best used when we are living in multi-

product environments or generic development where the product line has a long life.

The assessment given above is informal and intuitive. This is due to the fact that it is a qualitative analysis. To do a quantitative analysis we need quantitative models of the reuse objects, transformations, and context. We need a measurement framework for characterizing via categorization and classification, evaluation, prediction, and motivation to support management and technical decisions. To do this we apply the goal/question/metric paradigm to the models.

The Goal Question Metric Paradigm

The goal/question/metric (GQM) paradigm [1,2,6] represents a systematic approach for setting the project goals (tailored to the specific needs of an organization), defining them in an operational, tractable way by refining them into a set of quantifiable questions that in turn imply a specific set of metrics and data for collection. The tractability of this software engineering process allows the analysis of the collected data and computed metrics in the appropriate context of the questions and the original goal. This context supports feedback (by integrating analytic and constructive aspects) and learning (by defining the appropriate synthesis procedure for lower-level into higher-level pieces of experience).

The process of setting goals and refining them into quantifiable questions is complex and requires experience. In order to support this process, a set of templates for setting goals, and a set of guidelines for deriving questions and metrics has been developed [2].

Goals are defined in terms of purpose, perspective and environment. Different sets of guidelines exist for defining product-related and process-related questions. Product-related questions are formulated for the purpose of defining the product (e.g., physical attributes, cost, changes and defects, context), defining the quality perspective of interest (e.g., reliability, user friendliness), and providing feedback from the particular quality perspective. Process-related questions are formulated for the purpose of defining the process (quality of use, domain of use), defining the quality perspective of interest (e.g., reduction of defects, cost effectiveness of use), and providing feedback from the particular quality perspective.

Application of the Goal Question Metric Paradigm

In applying the goal/question/metric paradigm, we define the goals of the maintenance process and articulate the issues associated with choosing the appropriate process model, providing management with the questions that need to be answered to make intelligent decisions, understand the trade-offs, and perform risk analysis. There are a variety of goals we can generate. For example: to determine which process model should be chosen for a particular product, to improve our performance or evolve a better definition of any of the models for a particular product line.

In what follows we will generate a sample goal for maintenance and provide a partial list of the questions involved. Some of the answers will be obvious, either in the measures they require be taken or the information required form the experts; others will not. Thus a goal for maintenance in the context of the reuse framework might be:

Purpose:
To evaluate the new product requirements in order to reuse as much of the available products as possible.

Perspective:
Examine the cost and future evolution of the development from the point of view of the organization.

Environment:
Along with the standard environmental factors, such as resource factor, problem factors, we would like to pay special attention to the three context dimensions of the reuse framework.

Requirements Domain:
Clearly we are using product objects from the same application domain, although we have the ability to choose candidate components from other application domains.

Solution Domain:

This defines the process models, methods and tools that were used in the development of the existing product. If the same processes are to be used for the evolved project then there is not problem with reuse. However, the reuse model allows us to change the processes (and thus possibly to the product structure) at the cost of reusing less of the prior project. If there are to be changes then we must evaluate the cost of modification of the process and resulting product relative to the gains for process change.

Knowledge Transfer Mechanism:

If the maintenance group is the same as the development group then there is no transfer of knowledge required. If they are different then there are concerns that must be evaluated with respect to application, process and product knowledge of the maintainers and the kinds of documentation available.

Product Definition:

In considering the product, we actually have several. The new product to be built, i.e. the new version of the system, and the old versions plus any other systems that are relevant.

Product Dimensions

New Product:

How many requirements are there in total for the new system?

Old Product:

What is the mapping of requirements to system components?
What is the measure of the complexity of the traceability?
How independent are the components to be modified?
What is the complexity of the system and the individual system components?

Repository:

What candidate components are available in the repository and what are their context, transformation and object classifications?

Difference between new and old:

How many requirements are there that are not in the old system? (Categorize by size, new vs. modification of old vs. deletion of old, etc.)
How many components must be changed, added, deleted? (categorized by size and type of change)

Changes/Defects

How many errors, faults, failures (categorized by class) are associated with the requirements and components that need to be changed?
What is the profile of changes to the original system prior to this change?

Cost

What is the cost of understanding the new requirements?
What is the estimated cost of building a new system, reusing the experience and parts of the old project?

What was the cost of the old system in total?
What was the cost of each version?
What is the estimated cost of modifying the old system to meet the new requirements?

Customer Context

How will the new system be used?
What are the potential future modifications based upon our analysis of customer profiles, past modifications and the state of technologies?

Perspective:
cost and future evolution of the development

Model of Perspective: cost of modification of the design of the system vs. the expected future modifications
Parameters:
the life time of the system
the cost of future evolution of the system
the cost of evolving the old system versus rebuilding from old parts

Feedback:
Is the model appropriate?
How can the model be improved?
How can the estimations be improved?
How can classifications be improved?
How can activities be improved?

The Goal/Question/Metric paradigm allows us to develop other goals for reuse. These can be developed for whether the reuse object is a process or a product. Consider the following examples:

Evaluate the modification activity within the reuse process in order to improve it. Examine the cost and correctness of the resulting object from the point of view of the customer.

Predict the appropriate maintenance process model in order to perform the correct one. Examine its cost with respect to the customer needs and the future evolutions of the system from the point of view of the corporation.

Evaluate the standard corporate design method in order to assess how it should have been tailored for the current project. Examine its effectiveness from the point of view of the designer.

Evaluate the components of the existing product in order to determine whether to reuse them. Examine their independence and functional appropriateness from the point of view of their use in future systems.

Predict the ability of a set of code components to be integrated into the current system from the point of view of the developer.

Motivate the development of a reusable set of components in order to engineer them for reuse. Examine the reward structure from the point of view of the manager and developer.

Reuse Enablers

There are a variety of support mechanisms necessary for achieving maximum reuse that have not been sufficiently emphaisized in the literature. In this paper we have discussed several of these: a set of maintenance models, a mechanism for choosing the appropriate such models based upon the goals and characteristics of the problem at hand, and a measurement and evaluation mechanism. To support these activities there is a need for an improvement paradigm that aids the organization in evaluating, learning and enhancing the software process and product, a reuse-oriented evolution environment that motivates and supports reuse, and automated support for that model as well as the measurement and evaluation process.

The Improvement Paradigm: The improvement paradigm [1] is a high level organizational process model in which the organization learns how to improve their product and process. Within this model the organization should learn how to make better decisions on which process model to use for the maintenance of their future software products based upon learning from past performance. The paradigm is defined as follows:

1. Planning. There are three integrated activities to planning that are iteratively applied:

   (a) Characterize the current project environment. It provides a quantitative analysis of the environment and a model of the project in the context of that environment. In the context of maintenance, the characterization should provide product dimension data, change and defect data, cost data and customer

context data for earlier versions of the system to be modified, information about what classes of candidate components are available in the repository for the new system, and any information feedback from prior projects about experience with the different models for the types of modifications required.

(b) Set up goals and refine them into quantifiable questions and metrics using the goal/question/metric paradigm, for successful project performance and improvement over previous project performances. This consists of a top-down analysis of goals that iteratively decomposes high-level goals into detailed sub-goals. The iteration terminates when it has produced sub-goals that we can measure directly. For maintenance this involves the development of specific G/Q/Ms as specified in the prior section.

(c) Choose and tailor the appropriate construction model for this project and the supporting methods and tools to satisfy the project goals relative to the characterized environment. Understanding the environment quantitatively allows us to choose the appropriate process model and fine tune the methods and tools needed to be most effective. For example, knowing the effect of prior applications of the various maintenance models and methods in creating new projects from old systems allows us to choose and fine tune the appropriate process model and methods that have been historically most effective in creating new systems of the type required from older versions and component parts in the repository.

2. Analysis. Analyze the data to evaluate the current practices, determine problems, record the findings and make recommendations for improvement. We must conduct data analysis during and after the project. The goal/question/metric paradigm provides traceability from goals to metrics and back. This permits the measurement to be interpreted in context ensuring a focused, simpler analysis. The goal-driven operational measures provide a framework for the kind of analysis needed.

3. Learning and Feedback. This step involves the organization and encoding of the quantitative and qualitative experience gained from the current project into a corporate information base to help improve planning, development, and assessment for future projects. The results of the analysis and interpretation phase can be fed back to the organization to change the way it does business based upon explicitly determined successes and failures. In this way, we can learn how to improve quality and productivity, and how to improve definition and assessment of goals. We can start the next project armed with the experience gained from this and previous projects. For example, understanding the problems associated with each new version of a system, provides insights into the need for redesign and redevelopment.

A Reuse-Oriented Environment: Reuse can be more effectively achieved within an environment that supports reuse [3,8,13]. Software engineering environments provide such things as a project data bases, and support the interaction of people with methods, tools and project data. However, experience is not controlled by the project data base or owned by the organization. Reuse only exists implicitly.

We need to be able to incorporate the reuse process model into the context of development. We need to combine the development and maintenance models in order to maximize the context dimensions. We need to integrate characterization, evaluation, prediction and motivation into the process. We need to support learning and feedback to make reuse viable. We propose that the reuse model can exist within the context of the improvement paradigm, making it possible to support all of the above requirements.

The TAME Project: The improvement paradigm and the reuse oriented process model require automated support for the data base, encoded experience, and the repository of prior projects and reusable components [2,3,14]. We need to automate as much of the measurement process as possible, and provide a tool environment for managers and engineers to develop project specific goals, and generate operational definitions based upon these goals that specify the appropriate metrics needed for evaluation. The evaluation and feedback cannot be done in real time without automated support. Automated support will help in the post mortems analysis.

The goal of the TAME system [2] is to instantiate and integrate the improvement and goal/question metric paradigms and help in the tailoring of the software development process. But it can also support the reuse-oriented process model. The TAME environment model contains basic mechanisms for supporting systematic learning and reuse. To help with systematic learning it provides support for recording experience, off-line generalizing or tailoring of experience, and formalizing experience. To help with systematic reuse it supports mechanisms for using existing experience and on-line generalizing or tailoring of candidate experience. In this way it attempts to integrate both learning and reuse into an overall evolution model.

The application of the TAME system concept to maintenance will provide a mechanism for choosing the appropriate maintenance process model for a particular project and provide data to help us learn how to do a better job of maintenance.

Summary

The approach to maintenance depends on the nature of the problem and the size and complexity of the modification. This paper recommends that we view maintenance as a reuse process. In this way the maintainer is provided with a reuse model and a framework for viewing maintenance that permits a measurement framework to be applied. A new model of a reuse-oriented evolution process can be developed in which the existing models can be defined. Existing models can then be analyzed within this framework, allowing an organization to evaluate the strengths and weaknesses of the different approaches and provides feedback in refining the various process models and creating an experience base from which to support further management and technical decisions.

The approach provides support for defining activities, determining options, and evaluation. If the approach is not adapted then it is difficult for an organization to know which process model to use for a particular project, whether they are evolving the system appropriately, and whether they are maximizing quality and minimizing cost over the life of the system.

References

[1] V. R. Basili, "Quantitative Evaluation of Software Methodology," Dept. of Computer Science, University of Maryland, College Park, TR-1519, July 1985 [also in Proc. of the First Pan Pacific Computer Conference, Australia, September 1986].

[2] V. R. Basili, H. D. Rombach "The TAME Project: Towards Improvement-Oriented Software Environments," IEEE Transactions on Software Engineering, vol. SE-14, no. 6, June 1988, pp. 758-773.

[3] V. R. Basili, H. D. Rombach "Towards a Comprehensive Framework for Reuse: A Reuse Enabling Software Evolution Environment," University of Maryland Computer Science Technical Report, UMIACS-TR-88-92, December 1988.

[4] V. R. Basili, H. D. Rombach, J. Bailey, and B. G. Joo, "Software Reuse: A Framework," Proc. of the Tenth Minnowbrook Workshop on Software Reuse, Blue Mountain Lake, New York, July 1987.

[5] V. R. Basili, A. J. Turner, "Iterative Enhancement: A Practical Technique for Software Development," IEEE Transactions on Software Engineering, vol. SE-1, no. 4, pp. 390-396, December, 1975.

[6] V. R. Basili, D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," IEEE Transactions on Software Engineering, vol. SE-10, no. 6, pp. 728-738, November 1984.

[7] L. Belady and M. Lehman, "A Model of Large Program Development, IBM Systems Journal, vol.15, no.3, 1976.

[8] Ted Biggerstaff, "Reusability Framework, Assessment, and Directions," IEEE Software Magazine, March 1987, pp.41-49.

[9] T. P. Bowen, G. B. Wigle, J. T. Tsai, "Specification of Software Quality Attributes," Technical Report RADC-TR-85-37, Rome Air Development Center, Griffiss Air Force Base, N.Y. 13441-5700, February 1985.

[10] Federal Information Processing Standards, "Guideline on Software Maintenance," U.S. Dept. of Commerce/National Bureau of Standards, FIPS PUB 106, June 1984.

[11] P. Freeman, "Reusable Software Engineering: Concepts and Research Directions," Proc. of the Workshop on Reusability, September 1983, pp. 63-76.

[12] R. B. Grady, "Measuring and Managing Software Maintenance," IEEE Software, Vol. 4, No. 5, September 1987, pp. 35-45.

[13] IEEE Software, special issue on 'Reusing Software', vol.4, no.1, January 1987.

[14] IEEE Software, special issue on 'Tools: Making Reuse a Reality', vol.4, no.7, July 1987.

[15] G. A. Jones, R. Prieto-Diaz, "Building and Managing Software Libraries," Proc. Compsac'88, Chicago, October 5-7, 1988, pp. 228-236.

[16] B. P. Lientz, E.B. Swanson, and G. E. Tompkins, "Characteristics of application software maintenance," Communications of the ACM, Vol 21, No. 6, June 1978, pp. 466-471.

[17] R. Prieto-Diaz, P. Freeman, "Classifying Software for Reusability," IEEE Software, vol.4, no.1, January 1987, pp. 6-16

[18] H. D. Rombach, V. R. Basili, "A Quantitative Assessment of Software Maintenance: An Industrial Case Study," in Proc. Conf. Software Maintenance, Austin, TX, Sept. 1987, pp. 134-144.

[19] Mary Shaw, "Purposes and Varieties of Software Reuse," Proceedings of the Tenth Minnowbrook Workshop on Software Reuse, Blue Mountain Lake, New York, July, 1987.

[20] W. Tracz, "Tutorial on 'Software Reuse: Emerging Technology'," IEEE Catalog Number EHO278-2, 1988.

[21] S. S. Yau, R. A. Nicholl, J. J.-P. Tsai, and S.-S. Liu, "An Integrated Life-Cycle Model for Software Maintenance," IEEE Transactions on Software Engineering, Vol. SE-14, No. 8, August 1988, pp. 1128-1144.