SOFTWARE REUSE: A FRAMEWORK FOR RESEARCH

Victor R. Basili

John Bailey

Bok Gyu Joo

H. Dieter Rombach

Department of Computer Science

University of Maryland

College Park, MD 20742

(301) 454 - 2002

SOFTWARE REUSE: A FRAMEWORK FOR RESEARCH

ABSTRACT

Reuse of software products, processes, and knowledge will be the key to enabling the software development industry to achieve the dramatic improvement in productivity and quality which is required to satisfy the anticipated demands. Although experience shows that certain kinds of reuse can be successful, general success in this area is elusive. A reuse technology which allows broad and extensive reuse could provide the means to achieve the desired order-of-magnitude improvements.

This paper provides a framework for necessary research with respect to reusability. We take a broad view of reusability in order to capture as many dimensions of the concept as possible. The framework consists of five dimensions which can be used to describe any instance of reuse. In addition to the objects of the reuse (product, process, or knowledge), we discuss the mechanism of, the timing of, the goal for, and the support for the reuse. Through the examination of scenarios where some success in this area has already been achieved, we develop a set of desirable characteristics of reusable objects. We also discuss on the strategies that allow the smooth transition toward reuse-oriented software environments. Finally, we apply a formal method for measurement and evaluation to a topic of reuse, which yields potential areas for further research in reuse.

1. INTRODUCTION

Reuse is the key to progress in any area. If we do not reuse previously developed ideas and products then everything must be created from scratch and no progress can be made. In the development of software, we routinely reuse knowledge in the form of experience, processes in the form of methods, and products in the form of tools. Although the reuse of software components is less commonplace, there are certain well-known situations, such as in the case of mathematical or I/O routines, where the development and use of reusable components has been very successful.

The research objective in this area is to discover how to achieve a systematic way of reusing existing products, processes, and knowledge to get the maximum cost benefit. Successes in the reuse of software are due to both the characteristics of the reused items themselves and to the environment in which they are reused. Here are the general problems we have to deal with:

1) Creating items so they are reusable in the future (item characteristics),

This work was supported in part by the Air Force Office of Scientific Research under Contract AFOSR-F49620-85-K-0008, by the National Aeronautics and Space Administration under Grant NSG-5123, by the VITRO , and by the Penn Central

2) Finding appropriate reusable items (environment characteristics), and

3) Integrating them to build a new system (both item and environment characteristics).

The rest of this paper is organized into four sections. The next section presents our framework for describing software reuse and includes some brief examples. The section 3 uses that framework to give an overview of currently successful reuse practices and shows how the framework can be used to identify other, currently unpracticed, possibilities for reuse. Following that is a section which describes how an existing research paradigm can be applied to help identify, study, and quantify the issues of software reuse. The last section uses this research template to describe one of the reuse research areas currently under examination at the University of Maryland.

2. A REUSABILITY FRAMEWORK

In order to discuss reusability, definitions of its various dimensions and aspects are needed. In this paper we take a broad view of the term to capture as many of the issues as possible [1] We suggest that with respect to any instance of reuse there are two major dimensions, the object of reuse and the manner or mechanism by which that object is reused. Figure 1 shows a diagram of these two principal dimensions. Three other dimensions which are probably more appropriately called as attributes since they are not necessarily orthogonal, include the timing of the reuse, the goal for the reuse, and the support for the reuse. We believe that the same framework can also be used to discuss the production of reusable components by characterizing any such creation activity by how the results are expected to be reused.

Object (What do we reuse?):

We can reuse products, processes, or knowledge. Products can be anything from requirements documents to code, or from application libraries to tools. Processes include techniques, methods, and heuristics. Knowledge includes engineering, management, and applications

[1] The dimensions presented here are an extension of the work of P. Freeman [7].

knowledge and experience. Note that there are overlaps among these objects, for example management knowledge of a tracking process could be embodied in a tool.

Mechanism (How do we accomplish reuse?):

This dimension is related to how well intact an original object remains when it is reused. At one end of the scale is verbatim reuse of an object. Next would be the use of instances of a general object, which might be parameterized in the case of code products (as with Ada generics), or which might be tuned for some new usage, such as with reused knowledge or processes. Next on the scale would come reusable templates which require fleshing out before yielding a usable object. An example of this would be a requirements document outline (a reusable product template) or a management framework which recommends using reviews, formal test plans, or configuration control, for example, without specifying exactly how a project should be controlled (a reusable process template). At the other end of the scale would be the unconstrained modification of existing products to suit new needs. This is the most common mechanism for reusing software products at this time.

Timing (When do we reuse something?):

This is used to define the activity or phase of the development life cycle which is gaining productivity through reuse, e.g. planning, requirements definition, testing, documentation, etc. Although this attribute is largely constrained by the product being reused it is included here to help further categorize and specify any example of reuse.

Goal (What is the target for the reuse?):

Also somewhat constrained by other factors, this attribute is included to help describe the disparity between the environment of the original object and that of the reuse of that object. For example, the reuse of a product might be limited to one machine architecture and operating system or it might be possible to retarget the product across many different machines or environments. Or, the design methodology (process) previously used to develop a communication system might be used again on a distributed network problem.

Support (What assistance is there for access, modification, or porting an object?):

This attribute is needed since it can have an important affect on how a reusable object will be created in the first place, or whether it is likely to be reused later. If there is a formal mechanism for retrieval, such as through a high level application-specific language (such as with math library routines), there is a higher chance that reuse will be achieved. Or, if a tool or method is known to exist which can assist in the modification of an object, that object might be created initially in a way which would facilitate its modification by that tool.

Examples of Reuse

Here are the examples of reuse practices illustrating the coverage of all the dimensions:

Example 1: Program code (Object) will be reused by manually selecting and modifying functions from previous systems (Mechanism and Support) during the design and coding phase (Timing) of software for a new satellite but operating on the same machine environment (Goal).

Example 2: Management knowledge in the form of metrics and their associated interpretations on previous projects (Object) will be resued during the life cycle of the development process (Timing) by comparing the current metric values against prior values (Mechanism), supported by a historical knowledge base which has report generating capabilities (Support) for projects built in the same environment (Goal). [6].

Example 3: Ada packages (Object) will be resued by parameterization and instantiation (Mechanism) with the support of a library cataloging and retrieval system (Support) during the system building phase (Timing) for a different operating system on a different machine (Goal).

3. REUSABILIY: STATUS AND APPROACHES

In this section we discuss some examples and scenarios of reuse in the context of the preceding framework. The purpose is to illustrate the way in which the framework can be used to identify any reuse situation and to demonstrate how this process of using the framework can lead to useful insights. The section is organized first by the object dimension of the framework and

secondarily by the reuse mechanism dimension. We include in this section a discussion on the desirable characteristics of reusable objects and of the development environment in which they are accessed and integrated. Also, we illustrate how it is possible to use the framework to conceive of transition strategies which could increase our current reuse capabilities.

3.1. Reusable Products

Product Reuse Without Modification

The most familiar examples of reused products are mathematical and scientific routines. The reuse of such products has been successful because they have been accepted as both standard and useful by a large user community, they are specified in a language already common to that community, and their functional behavior is well understood. Additionally, their operand types lie in a common domain, they can be automatically accessed from a high order language, and they are non-trivial to build from scratch. Keeping in mind these attributes which appear to be responsible for the reusability of these products, we can look at other kinds of reuse identifiable by the framework.

At the other end of the size spectrum for reusable products are self-contained applications which can become parts of larger systems, such as editors, compilers, spreadsheets, databases, etc. The standardization and acceptance of these products often follows their availability, but contributes to their success just the same. Also, the command languages of the operating systems on which these products reside can be thought of as the languages used to access them.

In the middle of the size spectrum we find less success. At this point we would expect to find components from which self-contained applications can be constructed. Some of the factors required for reuse success may be found here but there are evidently other factors which have impeded progress. For example, most systems for payroll, communications, control, etc., are developed from scratch even though there is a significant demand for them. Perhaps the fault lies in the fact that there are no universal specifications for such systems. As in the previous example, we have failed as yet to establish a universal language for most of these higher level application

areas.

If we examine products other than software end products, such as requirements, design, documentation, etc., we find even less reuse. Again, the reason for this seems to be the lack of wide-spread acceptance of languages in these domains.

Product Reuse Using Templates

An example of a reusable template is a requirements format specification which must be expanded with specifics to describe the requirements for a given system. The standard format contributes to the understandability of the requirements and helps ensure that the necessary information is included. Templates are also found as module headers in design and code documentation. Code macros and generics also act as templates which are automatically expanded into final products according to the some well-specified parameterization. The insight available from this analysis might lead us to examine the feasibility of automatic parameterization of requirements and design documents, as well.

Product Reuse With Modification

Reusing an end product by modifying it is a common practice. In all maintenance activities, the desired system is sufficiently close to the maintained system to warrant a modification or enhancement effort instead of a new development. This is an important example of reuse, since the existing system becomes a (modified) component of the new system.

It is also common during the development of any software product to examine and adapt examples of a similar, previously developed product. There is currently little or no automatic support for these adaptations, although it is possible to conceive of tools which could assist in such tasks. This points to the fact that, as we examine the framework in search of ways to improve productivity through reuse, we might find alternate, equally promising, paths. In this case, we might wonder whether the availability of tools to support adaptation would result in a greater increase in productivity than would the use of methods to create reusable components which can be composed without modification to build new systems.

3.2. Reusable Processes

Software development already relies on reusable processes, as does any organized discipline. Existing methods, techniques, and heuristics all contribute to our ability to produce software at all. Since reusable processes are already an established part of the practice, it is common to overlook them when discussing software reuse. However, it is useful for our purposes of identifying research issues to apply the same analysis to processes that we have applied to products.

Reusable Processes Without Modification

Currently, few software development methods or techniques are so well specified that they can be applied in a completely deterministic manner. Any such processes are typically converted into tools (compilers, translators, test generators, etc.) which can then be thought of as reusable products in our framework. This suggests that completely specified processes are more desirable than non-deterministic ones since they can be automated.

Reusable Processes Using Templates

The majority of defined methods and techniques would fall into this category. The user of a generic process is given guidance but must determine the exact results at each step according to his or her knowledge of the specific goal and application area. If one considers tool support (another attribute in the framework) for automating this activity, which amounts to instantiating process templates, the idea of process generators results. The resulting processes would be more deterministic, resulting in an increase in productivity.

Reusable Processes With Modification

Although the distinction between modifiable processes and generic processes (above) is probably not a clear one, we could define this category to include immature methods or techniques which have not yet been well defined or proven through experience. Any such process would need to be refined and adapted through use, and only if it proved successful would it warrant a more complete definition. Reasonable components of this category would be heuristics and rules of thumb, and not what we typically refer to as methods.

3.3. Reusable Knowledge

It is impossible to completely dissociate knowledge from the other two objects of reuse, products and processes. However, knowledge in the form of experience can be reused without reusing a particular process or product. For that reason, knowledge was added to the set of objects in the framework.

Because this discussion borders on cognitive psychology, we will not attempt to expand upon it here. However, we note that the use of the framework helps us understand that some knowledge is in a form which allows it to be captured by a tool, knowledge base, or even a training curriculum, and thus can be made widely available. Knowledge which is not so well specified can still be an important object of reuse. This knowledge is typically exchanged among programmers and reused informally.

3.4. Desirable Product Attributes

This section describe the attributes of a product which contribute to its reusability.

(A) Understandability

The products to be reused later by other people should be well documented and understandable. Imagine that you have to understand completely the components made by others and modify them to make it fit into your system. Then you cannot get much gain from reusing them may be little. The user have to spend a little effort in reading them and understanding what it is and what it does.

(B) Formalism

To achieve higher understandability and to make sure the correctness of products they are needed to be formally specified as possible. That formalism used in products is a sign of the maturity of the application area. This issue becomes very important in case where the reusable products are program modules which might be reused directly in the system to be built.

(C) Correctness

This requirement for correctness becomes more important than ever, especially when we are reusing the program modules made by others. Here, correctness requirement is more important than reliability requirement, because the latter cannot be tested appropriately for the unknown environment. This means that the program components should be correct with respect to the specification of the components. Also we hope that the components be reliable or functionally correct for possible uses of them. This requirement is not expensive one considering the fact that those correct products will be reused over and over again, not only once.

(D) High level of abstraction

In general, the higher level abstraction/ object/ function a component represents, the more gain we can get from the reuse of it. A function is more reusable than single program statement, Module which implements abstract data type with its data structure and possible operations to it, and the module which implements the concept object in object oriented programming is more useful than a simple function. Also application specific systems will be the most useful.

(E) Adaptability

The components to be reused need to be adaptable. This means that they are general so that they can be reused in as many similar systems as possible, and they require us less effort in changing and integrating them into a new system being built. One promising technique for easy adaptability is parameterization. This idea has been used for a long time since the concept of subroutine is emerged in programming. The generic feature of Ada language supports much of this technique.

3.5. Environment for Products Reuse

In the new software development environment, where reuse of products plays major role in software development, we might have a large collection of the reusable components, the tools to maintain this collection, and tools to support integrating those components to build a new system. The major issues for us to achieve high degree of reuse of products are:

(a) What kind of mechanisms are necessary to easily find out the components needed ?

(b) How can we achieve efficient implementation of reuse ?

(c) What kind of supporting tools do we need?

A. Cataloging and Searching

The tasks of maintaining the large collection of components and allowing the users to easily find out the components they need are critical to reduce the cost of reuse. We have to have the effective tools to support cataloging the components and searching them. To do that, we might have to borrow the ideas and techniques from the data base management system area, artificial intelligence (knowledge representation techniques), and system science (techniques of building systems with components).

B. Standardization

For effective reuse environment, we need to achieve some standards on the description of products and possibly on the structure of products. The standards are necessary for each type of components: from a plain code module to a large module which contains the requirements specification as well as other documents down to code, We have achieved some progress on this standardization effort as in IEEE requirement specification standard [1] and the design standard [2].

C. Reuse Implementation

Once we found the an objects suitable for our needs, we have to change or adapt it to fit into new system by some way. Most primitive type of implementing reuse is hand modification. This technique has been the most typical form of reuse of the various documents from the program codes to the requirements specifications. Ideal cases would be direct reuse of objects without any modification. This technique has been used on some program codes and documents. This gives us much gain in productivity but it has been rarely successful.

Greater gain could be achieved by finding out the way of automating this implementation tasks previously done by hand modification. One promising technique is to make the products

using parameterizing technique and then to make the instances of them when needed. In case of program modules, some people have suggested such ideas [8] and the generic feature of Ada language makes it much easier.

D. Integration Tools

Other than managing the large amount of reusable objects and making the tools to support the implementation of reuse we need a system to help building a software by integrating the components. Also the existing tools to support software development need to be tuned to this new environment.

3.6. Transition Strategies

As stated earlier, the study of reusability must address both the creation of objects of reuse and the reuse of those objects. If we concern ourselves with a study of how both these concerns might be integrated into current software developments we can hypothesize about how improvements can be made in both areas. Our study so far, which has been facilitated by the proposed framework, has revealed some possible directions which we can take to improve our overall reuse of products, processes, and knowledge.

It appears that we consider most successful those examples of product reuse where no modification is necessary. This has been demonstrated with math, statistical, I/O, and other low level routines for which there is a well-specified language which implements an automatic retrieval mechanism. This has also been demonstrated with complete programs which are reused in larger problem solutions and are accessed by an operating system call in some command language. If we attempt to extrapolate these successful positions across the other domains of reuse we can imagine software components which are automatically accessed by a development tool and processes which are entirely deterministic.

The Ada language includes features which support the specification of, the creation of, and the composition of program components. If successful, these features should motivate both the development of reusable components and high level languages and technologies which can take

advantage of them. Ultimately, if this facility can reduce the cost of creating and reusing such components below the current costs of recreating functionality from the lowest level primitives, this strategy will succeed.

The development of component technology for applications development is clearly an important topic for two major reasons. First, it may be our only hope for an order of magnitude improvement in productivity. It is unreasonable to expect that good methods and tools alone will account for the needed improvement. Second, it will act as an impetus for higher quality. The cost effectiveness of applying correctness technology is easier to justify on a product that will be used many times than it is for something that will be used only once.

In order to transition to the use of components, however, these components must be created. It is not clear that we would be able to create these in a vacuum without examining the components which are typically called for in existing designs. Although it may not be the best decision to model a reuse environment after a conventional environment, it might be the most expedient one for the time being. One of the research topics being investigated currently at the University of Maryland is how existing program components which are not necessarily written to be reusable can be converted into eminently reusable ones. This is intended to allow an original development to proceed without the encumbrance of being concerned with developing for reuse, and yet to still be able to ultimately reuse as much of the output as possible for future developments. This topic is further described in the last section using the research paradigm described in the next section.

4. MEASUREMENT AND EVALUATION

The reuse framework as well as the discussion of various approaches to reusability illustrates the wide range of possibilities that exist in employing reuse. We need to understand better the characteristics of reusable products and the interactions among the products, methodologies, productivity, and quality. A sound scientific approach to assuring progress in this area is to quantitatively evaluate existing examples of reuse and the effects of new methods and tools for their support of reuse, with respect to the resulting productivity and quality. This implies the use of

measurement and experimentation.

4.1. Research Goals

The research goals are:

(1) To develop an evaluation methodology to formulate, quantify, and achieve reuse-related goals, based on data collected from experiments and case studies.

(2) To conduct experiments to understand the characteristics of reuse and the impact of reuse on productivity and quality, to evaluate the benefits of methods and tools with respect to reusability, and to feed back this new understanding into reuse research community

(3) To design and develop tools to support the evaluation activities stated, and to assist in formulating further evaluation goals and questions.

This is not just a simple application of existing measurement and evaluation technology [3]. There exist challenges from the standpoints of both reusability and measurement. The challenges which are germane to the topic of reusability include (1) defining reuse (the previously described framework is intended to facilitate this), (2) identifying object and environment characteristics which support reuse, and (3) identifying the impact of reuse on productivity and quality. The challenges which are germane to measurement include (1) defining measures of reuse (which could vary with the style of reuse), (2) redefining measures of productivity (since conventional measures of developed lines of code would penalize reuse), (3) redefining measures of quality (possibly shifting the emphasis from simple reliability to correctness), and (4) isolating the impact of reuse on any observed changes in productivity and quality.

4.2. Approaches

There already exist approaches to measurement and evaluation that have proved successful in other application areas. The approach proposed in [3] is based on the goal/ question/ measure paradigm. The proposed approach consists of an evaluation methodology, templates for formulating goals and questions, a strategy for experimentation, and a tool set providing a measurement infra-structure.

Goal/Question/Measure paradigm:

The measurement and evaluation process requires a mechanism for determining what data is to be collected, why it is to be collected, and how the collected data is to be interpreted [5]. This mechanism asks for determining the goals of the measurement and refining each goal in a traceable way into a set of quantitative questions that defines, in turn, a specific set of data for collection.

Evaluation methodology:

The methodology consists of seven steps [3]:

- Formulating evaluation goals
- Refining goals into subgoals and questions
- Establishing appropriate measures
- Planning the analysis layout and analysis methods
- Designing and testing data collection schemes
- Performing the investigation concurrently with the data validation
- Analyzing and interpreting data in the context of the prior developed questions, subgoals, and goals

Templates for formulating goals and questions:

The task of formulating goals and questions proved to be rather complicated. Based on our experience of applying this evaluation methodology we developed a framework for supporting this task. This framework structures and guides the task of formulating goals and questions by providing templates including the important aspects to be covered. In the following this framework is presented and annotated with a reuse example; this example covers the evaluation of our reuse example 1 from a correctness perspective:

(A) FORMULATION OF GOALS:

PURPOSE OF STUDY: To (characterize, evaluate, predict, motivate) the (process, product, model, metric) in order to (understand, assess, manage, engineer, learn, improve, compare)

it

e.g., to evaluate the reuse practice in current projects in order to understand it.

PERSPECTIVE OF STUDY: Examine the (cost, effectiveness, reliability, correctness, maintainability, etc.) from the point of view of the (developer, manager, customer, corporation, etc.)

e.g., examine the correctness of reused components from the point of view of a developer.

ENVIRONMENT OF STUDY: List the various process factors, problem factors, people factors, etc.

(B) GENERATION OF QUESTIONS:

DEFINITION OF THE PROCESS:

QUALITY OF USE (characterize the process quantitatively and assess how well the process is performed):

e.g., What kind of products is reused?

DOMAIN OF USE (characterize the object of the process and evaluate the knowledge of object and domain by the performers of the process):

KNOWLEDGE OF DOMAIN:

e.g., How many of the actually reused components were known in advance?

VOLATILITY OF DOMAIN:

e.g., How many of the reused components had to be modified?

COST OF USE (characterize the cost for performing each of the subactivities of the activity being performed):

e.g., What effort is required to test reused components (modified/unmodified)?

EFFECTIVENESS OF USE (characterize the results of the process and evaluate the quality of that results):

RESULTS:

e.g., How many (and what types of) errors were found in reused components as compared to new-developed components?

QUALITY OF RESULTS:

FEEDBACK FROM USE (characterize the major problems with the application of the process so that it can be improved):

e.g., What characteristics of components impact correctness?

A similar template could be applied for product questions. This template includes the definition of the product and the evaluation of the product with respect to a particular quality (in this case it would be correctness).

Approach to experimentation:

Experiments are classified by number of projects studied and number of teams involved per project [4] According to this classification four different types of studies exist:

a) Single project studies (one team, one project)

b) Multi-Project variation studies (one team per project, more than one project)

c) Replicated project studies (More than one team, one project)

d) Blocked Subject-Project studies (More than one team per project, more than one project)

Evidently, studies of class a) or b) can be conducted in real project environments; they can not be controlled and, therefore, the results frequently lack statistical significance. Studies of class c) and d) can hardly be conducted in real project environments because of economical constraints; it is impossible to replicate a large development project ten times just for the purpose of statistical significance of measurement results. The latter kind of studies are usually conducted as controlled experiments. Nevertheless, both types of experiments are necessary in the context of reuse research. We need to be able to evaluate early hypotheses concerning reuse with statistical significance; for this purpose we conduct controlled experiments of class c) or d). As a second step we want to validate whether these early results are still true in real project environment; for this purpose we conduct case studies of class a) or b).

Tool set for measurement:

In order to apply any reuse measurement and evaluation technology to real projects, tool support is required. This tool support includes a data repository for all the measures taken, tools for measurement and evaluation, and a tool assisting in the process of formulating new measurement and evaluation goals and questions.

One of the reasons that various attempts to increase productivity and quality with reuse failed in the past, was the lack of feedback process that (1) could guide research in the right directions and (2) could provide the confidence in the effectiveness of the presented solutions to have an impact on productivity and quality. One of the reasons for believing that research in the area of reuse might be more successful this time is the fact that the technology for measuring and evaluating open questions, that means providing this necessary feedback process, have matured.

5. Applying the Measurement and Evaluation Paradigm

The following are two illustrations of how the research paradigm described in the previous section can be applied to direct research into two software reuse topics.

5.1. Code Component Transformation to Increase Reusability

Goals

Purpose of Study: To evaluate a process for transforming code to improve its reusability in order to assess and improve it.

Perspective of Study: To evaluate the effectiveness of transforming code to make it more reusable versus writing it initially for reuse from the point of view of the developer.

Environment of Study: The evaluation will be performed in the context of satellite ground support software, using participants who are experienced in the application domain, and performed in a minimally-disruptive manner.

Questions

(1) Does the transformation process take less time than the additional time it takes to develop a component for reuse?

(2) Is a transformed component more reusable than one which was developed for reuse?

(3) Can the transformation process be generally learned and applied?

(4) How many errors are introduced by the transformation process?

(5) How many errors are introduced by attempting to develop reusable components?

(6) Are there aspects of the transformation process that can be automated?

(7) Can the transformation process be improved?

Metrics

Time to transform a component

Time to generate a similar reusable component

Frequency with which a reusable component is selected for reuse

Assimilation time for a transformed component

Assimilation time for a component written for reuse

Changes to a transformed component

Changes to components written for reuse

Time to learn the transformation process

Similarity of transformations performed by different participants

Experimentation

Ethnographic study will be used to obtain feedback on the learnability of the process, the suitability of the transformations, and usability of the transformed components. Experimentation will be used to supply the required measures. Participants will learn a transformation process and transform components to increase their reusability. Participants will be given the description of several components, some of which were designed for reuse and some of which were transformed

from other components, and be asked to submit designs for a project in the same application domain using the described components and a minimum of new code.

6. Conclusion

In order to exploit all aspects of reuse in software development and maintenance, and thereby increase both productivity and quality, a framework to help identify and describe relevant research topics has been proposed. This framework is intended to help researchers understand exactly what part of the problem any given study addresses and, using the suggested research paradigm for measurement and evaluation, and to facilitate and promote communication about reuse topics.

It is hoped that this paper will serve as a point of reference to those involved in reuse research and that any efforts to identify the ways to improve or increase reuse, or any results which claim success in promoting reuse, will be phrased in terms of the framework presented here. The authors welcome comments or suggestions on these ideas, and feedback on how successfully they can be integrated into an existing or a planned reuse research project.

References

(1) IEEE Guide to Software Requirements Specifications, ANSI/IEEE Std 830-1984, 1984.

(2) H. J. Barnard, R. F. Metz, and A. L. Price, "A Recommended Practice for Describing Software Designs: IEEE Standards Project 1016," IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, February 1986, pp 258-263.

(3) V. R. Basili, "Quantitative Evaluation of Software Methodology," Proceedings of the 1st Pan Pacific Computer Conference, Australia, September 1985

(4) V. R. Basili, R. W. Selby, and D. H. Hutchens, "Experimentation in Software Engineering," IEEE Transactions on Software Engineering, Vol. SE-12, No. 7, July 1986, pp 733-743.

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

(6) V. R. Basili and C. L. Ramsey, "ARROWSMITH-P - A Prototype Expert System for Software Engineering Management," IEEE Proceedings of the Expert Systems in Government Symposium, McLean, Virginia, October 1985, pp 254 - 264.

(7) P. Freeman, "Reusable Software Engineering: Concepts and Research Directions," Proceedings of the Workshop on Reusability in Programming, September 1983 pp 63 - 76

(8) J. A. Goguen, "Parameterized Programming," IEEE Transactions on Software Engineering, Vol. SE-10, No. 5, September 1984, pp 528 - 543.

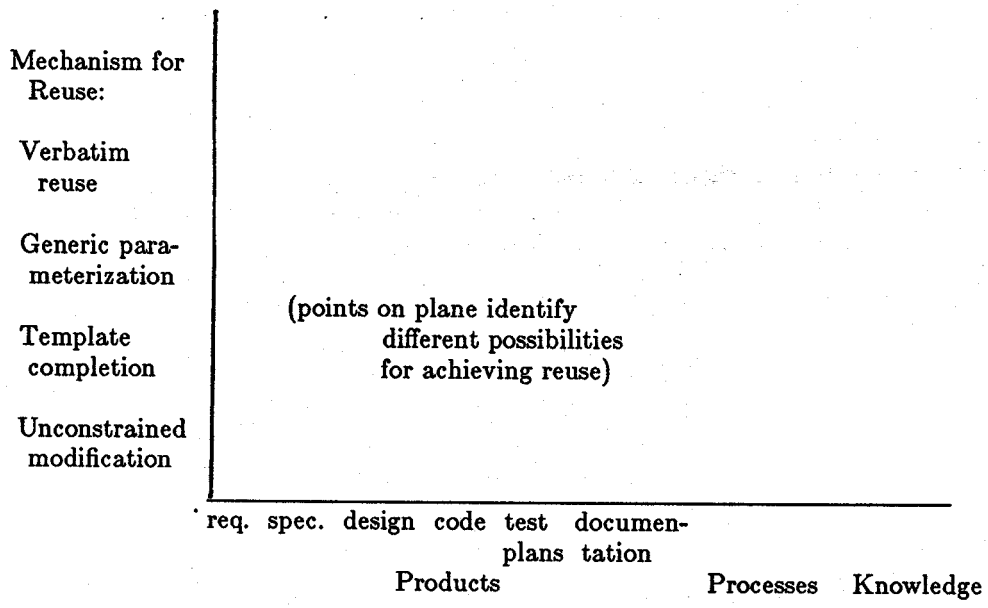

Figure 1. Primary reuse dimensions of Object and Mechanism