Journal
Software – Practice and Experience
Volume 30, 2000
Pages 685-722.

'Three Empirical Studies of a Software Reuse Reference Model'

(1) Dr. David C. Rine, Professor of Computer Science and Information & Software Systems Engineering, Department of Computer Science, School of Information Technology and Engineering, George Mason University, Fairfax, Virginia, 22030-4444, (703) 993-1530, drine@cs.gmu.edu.

(2) Dr. Nader Nada, Department of Computer Science, The Naval Postgraduate School, Monterey, California, nnada@cs.nps.navy.mil.

Address correspondence to author (1).

Key Words: Reuse, Reuse Practice, Software Reuse, Reference Model.
Abstract: The contribution of this paper is three empirical studies supporting a reference model for the practice of software reuse. Our research thesis is that software development based upon a software reuse reference model improves quality of products, productivity of processes and product time-to-market for many software development enterprises. The definition and investigation of such a model has been carried out using three steps. First, the reference model is developed based on existing software reuse concepts. Second, this reference model is empirically studied using three studies: one using a survey method, one using a case studies method, and one using a legacy studies method. Third, the impact of the reference model on software development productivity, quality, and time-to-market is empirically derived.

1. SOFTWARE REUSE REFERENCE MODEL

1.1. Introduction to Reference Models

Reference models serve as a means of comparing different systems in a domain. A reference model provides a guide against which systems in the domain can be evaluated. In this paper the domain is the set of software development systems such that each system is related software engineering activities of a given enterprise incorporating assets reuse.

The reference model presented in this paper identifies software reuse technical and organizational factors. It is the authors' general thesis that the lack of such a reference model makes it difficult to design improved reuse approaches in a systematic manner. The task of generating reference models for software development activities relies on having 'standard' representations that satisfy established or specified criteria for interpretation. Robertson [Robertson 94] and Sommerville [Sommerville 96] point to the

importance of reference models in software development, both for software products and for software development activities.

1.2. Importance of Reuse Reference Models

Many organizations in both the private and public sectors are investing money, time, and resources into software reuse. They hope to improve their competitive edge and time-to-market through increased productivity (decreased effort) in the software development process and increased quality in the software products developed [Rine 98]. For many software organizations it will therefore become increasingly important to investigate and measure the relationship between the level of the organization's software reuse utilization and management plan effectiveness in terms of effort with reuse, product quality, and time-to-market.

The development team is another important aspect enhancing the management of any reuse program. Failure to try to reuse assets is primarily a management problem. According to survey respondents, the most common root cause of this failure mode is resource constraints, followed by lack of incentive to reuse, time constraints, lack of clarity on reuse utility, and lack of education [Frakes 94b].

Since software assets that are specified in the software product model are directly derived from requirements specifications, it is a common practice that the decision to pick up particular reusable assets for product development happens during various development phases [Fonash 93, Wasmumd 93].

It is important that systematic software reuse is a key business strategy that software mangers can employ to dramatically improve their software development processes, to decrease time-to-market and costs, and to improve product quality [Griss 93].

The reason for identifying the activities of software Reuse Reference Model (RRM) based on improvement factors is that the RRM provides important relationships between the technical and organizational activities of software reuse within a software engineering process. Utilization of an appropriate software RRM in a software development organization allows software engineering management to identify both technical and organizational activities needed for an improved software reuse implementation plan [Jacobson 97, Lim 98]. Software development includes both business (organizational, cultural) and engineering (technical) activities. Any organization, either starting or currently practicing reuse, should review its current or intended reuse implementation plan, compare its plan to the plans of successful leaders in the reuse industry, and revise their plans, as necessary, based on an empirically justified software RRMs.

In this paper a model (RRM) is first presented, and then empirically studied. Defined levels of the utilization of RRM by an organization are also developed as a qualitative RRM Level (RRML) metric in section 2.2.3. The organizations used in this empirical study (See section 4.1) with high RRML levels of RRM (reuse capability) practice the

activities described in section 3. These technological and organizational activities presented in section 3 are shown from the data collected to be improvement factors.

We have found that past software RRM formulations [Rine 98, Nada 97] based on our reuse research [Nada 97, Baldo 97] did not identify or measure both of the software reuse technical and organizational factors necessary to imply reuse success. Software developers need to achieve better evaluations and measures of software reuse and business improvement factors, as well as the factors' predictive relationship to software productivity and quality.

Furthermore, our researched RRM, reported in this paper, serves as a point of reference for groups of similar software development organizations that are interested in adopting, utilizing, or managing software reuse. Moreover, our developed RRM implementation Level (RRML, see section 2.2.3.) is a measure representing similar organizations' utilization levels of the RRM activities (factors) in the given ordinal range. The ordinal range is divided into five linearly ordered level values: L1 to L5, that represent the amount of RRM activities utilized or practiced (examples: product-line, common architecture, COTS components, etc.).

1.3. The General Problem and Main Contribution

Many software development organizations believe that investing in software reuse will improve their process productivity and product quality, and are in the process of planning for or developing a software reuse capability [Baldo 97, Frakes and Isoda 94, Tracz 86-94, Troy 94, Tirso 93, McClure 92]. Unfortunately, there is still not enough data available on the state-of-the-practice of utilizing or managing software reuse. The majority of current information available on software RRMs comes from the literature [Parnas76, Basili91, Cusumano91, McCain91, Gold93, Wasmund 93, Card94, Fafchamps94, Gibbs94, Frakes 95, Nader 97, Rine and Sonnemann 98, Morisio, Ezran and Tully 99], where reports on improved measures for successful reuse are needed. A critical problem in today's practice of software reuse is a failure to develop necessary details to support valid software RRMs.

The main contribution of our research reported in this paper is an empirically studied RRM for the practice of software reuse. The research provides evidence that using a valid RRM among the software reuse community will help improve the competitive edge and time-to-market of many software development enterprises through increased productivity in the software development process and increased product quality. The research uses an initial set of legacy studies and lessons learned studies [Zelkowitz and Wallace 98] which we collected from software development enterprises. Our research describes activities of those software development organizations having a high RRML. These RRM activities are derived from their practice of reuse where there is a high correlation with reuse utilization and management with decreased effort and increased quality.

2. RESEARCH METHOD AND RESEARCH HYPOTHESIS

2.1. Research Thesis

This research asserts [Nada 97] that there is need of a more effective software RRM that incorporates both technical and organizational factors of software reuse and that identifies impact of such factors on software development productivity, quality, and time to market. This assertion is broken out into two specific research theses (T1, T2) that are tested in out study:

(T1) Thesis-1. Level of Reuse (RMML) determines effectiveness improvements in productivity, quality and time to market.

(T2) Thesis-2. Percentage of Adopted RRM Factors (PARF) is an indicator of the implementation Level (RMML), i.e. levels of reuse success of an organization's Reuse program.

The results in section 4 determine the extent to which these theses are supported.

2.2. Research Method

2.2.1. Deriving the Reuse Reference Model

In this section we summarize the fifteen step research approach taken in deriving and empirically supporting the theses about our Reuse Reference Model.
Step 1. Develop a software reuse reference model (RRM) from current knowledge.
A. Review literature on software reuse to identify reuse technical and organizational factors and explore their relationship to software development productivity, quality, and time to market.
B. Develop a modified version of the Sonnemann and Rine [Rine 98] model by:
i. Eliminating candidate factors that are weak or irrelevant to reuse by comparing and contrasting software reuse lessons learned from literature search, and
ii. Utilizing results of the previous software reuse empirical studies [Bollinger91; Cruickshank91; SER 95; QSM 94; Sonnemann 95; Patterson 95; Applied Expertise 94; Fonash 93; Griss 93; Frakes 95; Kiely 98; Boehm 81, 87, 92].
This step includes the refining of the studies comprising different technical and organizational activities of software reuse.
Step 2. From the RRM decompose the research problem into the following four sub-problems:

- Determine if the implementation Level (RRML) of the software RRM activities of those organizations (Org – See Tables 1a-1d) sampled is predictive of:
  A. Products development productivity (effort) level

B. Time-to-market level
C. Product quality level

- Determine if the Percentage of adopted factors of the RRM from those organizations (Org – See Tables 1a-1d) sampled is predictive of the implementation Level (RRML) of the software RRM activities.
- Determine at which phase of the software life cycle organizations may get the greatest reuse benefits.
- Determine to what extent organizations collect and utilize reuse metrics and incorporate reuse effort estimation as part of their software development life cycle.
- Determine if a reuse process is a common practice and an integrated part of the organizations' software development process.

Step 3. Identify the software reuse population from which both the case studies sample and survey sample are selected:

- Software reuse factors (Sonnemann' list, 109)
- NASA software reuse workshop (Patterson' list, 200)
- Reusable software components resources (Levine' list, 50)
- IEEE Software Reuse Group (workshop list, 90)
- Software Productivity Consortium members (list, 200)

Limitations on Population Identification: Software reuse is still an immature function. There is no national or international software engineering database or census bureau representing a population of software development organizations who practice software reuse.

Step 4. Stratify population and Identify experimental variables for the case studies and the survey:

- Stratification (Organization type, Application type, etc.)
- Control variables (Software Reuse Reference Model implementation Level "RRML" (See section 2.2.3.) Organization Size, Application Size)
- Dependent variables (Effort , Quality, Time-to-market)
- Independent variables (RRM factors)

Step 5. Develop a survey instrument to apply to each member of the survey sample of industry and government projects from the software reuse population (Step 3.).

The survey instrument includes 34 questions on the RRM technical and organizational activities, productivity (effort), quality, time-to-market Levels and Percentage of Reuse (Step 2.).

Step 6. Identify population of 24 organizations (Org) to be sampled and to become Reuse Case Studies.

In order to understand the specific context of each case study, an overview template is provided based on 20 attributes that have been defined to characterize the software applications development represented within each case study.

Step 7. Sample, contact and interview the 24 different case studies organizations (Org).

Limitation on Sample Data Collection: Some, but not all, of the persons interviewed from each organization and reporting data may have been estimating the data items supplied, and so the data may not always be applicable across the entire organization.

Step 8. Identify population of N organizations (Org) to be sampled and to become Reuse survey participants.
Step 9. Apply the survey instrument to each of the N of organizations of the sample-Survey industry and government reuse practices.
Step 10. Collect and analyze sampled case studies and survey data results. Apply statistical techniques to test theses.
Step 11. Apply data from both samples to test theses [Hoel54, Pfleeger94]:

- Apply statistical techniques to test theses.
- Apply statistical tools -The statistical SAS and JMP software packages are used to determine the survey frequencies and the correlation's between the dependent variable and the independent variables.

Step 12. Compare the results of both the survey and case studies.
Step 13. Interpret the results against the research theses:

Step 14. Compare sampled organizations' (Steps 1-12) project effort, development time (time-to-market) from different domains with previously collected software development organizations data from the QSM (Quality Software Management Corp. Database) software industry averages.
Step 15. Develop conclusions.
Step 16. Report the results and interpretations. Mail the summary of results to all the survey and case studies' participants.

2.2.2. Research Questions about our Proposed Software Reuse Reference Model

Our literature search and related theoretical work uncovered a number of possible software reuse factors, as well as reuse relationships between these factors and productivity, quality, and time-to-market. The goal of our research is to develop RRM whose utilization in software development lifecycle will increase productivity and quality and decrease time-to-market. From this goal the following five questions about RRM are derived:

Question 1. (Thesis 1) Is the Level of Reuse (RMML) (RRML, see section 2.2.3.) predictive of:
A. Products development productivity (effort) level
B. Time-to-market level
C. Product quality level
Question 2. (Thesis 2) Is the Percentage of Adopted RRM Factors (PARF) predictive of the implementation Level (RRML), i.e. levels of reuse success?
Question 3. At which phase of the software life cycle may organizations get the greatest reuse benefits?

Question 4. How should organizations collect and utilize reuse metrics and incorporate reuse effort estimation models as part of their software development life cycle?
Question 5. Is a reuse process common practice an integrated part of the organizations' software development process?

We have tried to answer these questions through interpretation of the results reported in section 4.

2.2.3. Reuse Reference Model Levels (RRML) Analysis

The RRML levels were assigned to each project based on the following model:

Level 1 (L1), Very Low,
Level 2 (L2), Low,
Level 3 (L3), Average,
Level 4 (L4), High, and
Level 5 (L5), Very High

The linearly ordered RRML is intended to produce realistic results. Level 3 requires neutral answers on questions Q3-Q21 and Q30-Q33 (the total median) , Levels 4, and 5 require agree or strongly agree answers on the same questions. Level 2 requires disagree answers to these questions. Level 1 will be made up of those projects that fail the level 2.

3. The Reuse Reference Model: Technological and Organizational Factors in RRM

The software RRM incorporates both technical and organizational factors that might be needed to establish an improved software reuse practice in the organization. Technical factors consist of: (1) technologies (tools) that support reuse: CASE tools and common interface, e.g., CORBA, DCOM, and COTS; and (2) software development with reuse processes (technical procedures): domain modeling, product-line approach, common architecture, quality control, and best development practices. (3) Organizational factors (e.g. organizational procedures) influencing software development with reuse: management of reuse program, market place analysis, financing and marketing forecast, and training.

Sections 3.1, 3.2 and 3.3 identify and define each possible factor in the RRM. The definition of each RRM factor has the following form: factor description, justification for its inclusion, and indication as to whether of not it came from the previous model [Rine98].

3.1. Technologies (Tools) That Support Reuse

Description: Tools that support the software reuse process, construction, testing, and management of software components.

Justification: These tools need to be evaluated, assembled and integrated into a variety of specialized environments to support the software reuse process, development and management team.

Prior Model: No

Architects and component developers need tools to develop models, as well as coding and testing. Managers and librarians need metrics tools, project management tools, and tools to manage multiple versions and configurations of components. Once the software development organization has been assessed and a plan of software reuse defined, a set of activities that support reusable assets based methodology installation should be incorporated. These activities include the selection and installation of appropriate methods and tools, and the definition of evaluation criteria for these tools.

3.1.1. Software Uniform Modeling and Design Notation

Description: Unified Modeling Language (UML) furnishing syntax and semantics for precisely describing system and business models.

Justification: UML is a key enabler using graphical and textual modeling notation that provides a clear and precise language allowing software engineers to describe an architecture and components that will ensure the development of robust and reuse-supporting systems [Jacobson 97].

Prior Model: No

The software community has been looking for and proposing solutions to "software problems" for many years. Until Components-Based Development (CBD), object technology was the last "solution." One of the keys to CBD's success is a standard infrastructure for components. [Kiely 98].

A uniform modeling and designing notation is needed that provides an accepted way of describing components' functions and properties, which would be critical in designing collaboration between components. Much of the industry is adopting the Unified Modeling Language (UML), which combines several prior modeling and design notations.

3.1.2. Standardized Software Component Interfaces

Description: A vendor interoperability standard designed to allow interoperability between different component [Fonash93, Ferrentino94, Booch96, Fichman97].

Justification: Standardized software component interfaces gives a precise description of types, exceptions, and type declaration that permits multiple language bindings, regardless of the platform, author, or vendor infrastructure.

Prior Model: No

A standardized software component interface is needed that lets any application in any language access components' features by, for example, binding to the component model or interface definition language. The OMG is trying to make both Microsoft's Distributed Component Object Model (DCOM) and Org21 Microsystem' JavaBeans specifications interoperable with its final Common Object Request Broker Architecture (CORBA) specification and whose infrastructure will conform to UML.

The Object Request Broker (ORB) specification is the part of CORBA from the OMG that describes a "software bus"-- a mechanism that handles communication between distributed objects in a system. The ORB allows for distributed, heterogeneous client-server objects interaction over a network. The ORB makes meta information describing object interfaces available to all objects in the system. With this objects may access other objects as clients without prior knowledge of their locations. Any object connected to the ORB can play the role of both a client and server object. That is, it can initiate calls to other objects and respond to requests for services from other objects on the ORB.

3.1.3. Trends of COTS

Description: Commercial-Off-The-Shelf (COTS) software components that are procured for integration into software system.

Justification: In some cases COTS are more cost-effective with less time-to-market than to reusing in-house-developed software components.

Prior Model: NO

Software reuse sources may come from inside the organization and from outside the organization. For at least some of the reused software, an organization may find it to be more cost-effective with less time-to-market to reuse existing Commercial-Off-The-Shelf (COTS) or Internet distributed software than to reuse in-house-developed software components. This is related to the traditional buy-versus-build tradeoffs. However, even though COTS or Internet distributed software may require less effort (greater productivity) and less time to acquire, this externally acquired software for reuse may be very expensive and time-consuming to port between computer systems and to integrate between incompatible interfaces. In order to solve this problem, should one decide to acquire instead of build, externally reusable software must be both portable/interoperable and easily integrated. COTS components provide an excellent source of reusable software components, that are often overlooked [Staringer 94].

Two important technical sets of problems facing the use of COTS are design/specification interfacing problems and integration problems. Let us consider interfacing problems in COTS components reuse. COTS components generally provide several functions needed for software products but not necessarily all of them. As a result of this fact, a precise mapping between specification/design and implementation

cannot be accomplished until the COTS components have been selected. Here, the intent to maximize COTS introduces a number of design variables. Sometimes it is not an easy task to integrate COTS components. In such case software developers have to find other options to overcome such problems.

3.2. Software Development with Reuse Processes (Technical Procedures)

At this point in our presentation, it is important to point out that in our case study and survey study participants are asked to make the following distinctions between domain model, product line, product family and common architecture.

Domain Model. A definition of all concepts and relationships between concepts in a domain of discourse.

The intent of a domain model is domain **knowledge** representation.

Product Line. A set of products sharing a common, managed set of features that satisfy specific needs of a selected market or mission.

The intent of a product line is market and mission **needs** or **requirements**.

Product Family. A set of systems built from a common set of assets.

The intent of a product family is core **capabilities** of an organization.

Common Architecture. An abstract architecture comprised of abstract components and abstract connectors such that each abstract component is a generalization of components from specific architectures and each abstract connector is a generalization of connectors from specific architectures.

The intent of a common architecture is reuse of an abstract **engineering design structure**.

The intent of each of these four principles of reuse is usually different. Take, for example, the domain of accounting. Representing the domain of accounting may be for the purpose of establishing an area of study, a professional discipline, standards for the practice of accounting or understanding problems in accounting. Knowledge is the intent. Representing sets of tools to help solve accounting problems of small businesses is for the purpose of doing business and meeting small business needs. Requirements is the intent. Acquiring and representing basic resources, personnel, expertise and infrastructure is for the purpose of an organization to be able to do its work. Capabilities is the intent. Systems are not reliable, maintainable, dependable or available if they are not constructed with good engineering design principles. Quality structure is the intent.

So, a total reuse capability needs to include all four of this different principles. These principles are (1) reuse of correct common knowledge, (2) reuse of correct common

requirements, (3) reuse of correct common capabilities, and (4) reuse of correct common structure.

We assume in our study that these four principles of reuse are different.

3.2.1. Domain Modeling

Description: A domain modeling activity represents a semantically closed abstraction of any system, used to model an appropriate range of commonality and variability of application(s) or system(s) domain.

Justification: Finding a systematic way of identifying a domain model, commonality and variability, potentially reusable knowledge, and a knowledge representation will highly enable software reuse.

Prior Model: Yes.

Domain modeling is a systematic approach for identifying the knowledge commonalties, similarities, and variabilities necessary to characterize and standardize any product-line within a domain. While domain knowledge can be represented using domain modeling, a domain is often realized for product-lines or product families, and its associated process for producing instances of those families. Domain engineering (procedure) can be an ongoing activity for defining, implementing, and evolving a domain and its domain model. Domain engineering uses business objectives and domain knowledge to create and standardize the product-line that the domain model supports. Domain engineers can later be used to help specify correct common architectures. On the basis of business objectives the domain manager creates a "master plan" for the development and evolution of the domain [O'Connor 94].

Domain analysis and modeling are concerned with understanding the domain so that domain abstraction can be modeled as reusable requirements, design patterns, or code component [Sommerville 96]. A domain-specific, architecture-driven approach to reuse will yield the greatest reuse impact and, thus, is important to address from both an engineering and a management perspective [STARS 95].

3.2.2. Product-Line Approach

Description: A product-line is a group or family of closely related products made by the same process and for the same purpose, differing only in style, model, or size.

Justification: A product-line approach groups related systems into application families and applies commonality and applies reuse across the family.

Prior Model: Yes.

Organizations that have developed similar products have a higher probability of success in developing new products in the same area or domain. Organizations whose strategy is

to continually enter new areas (pop up businesses), and who are not associated with a given area or application domain, are not likely to be able to fully leverage reuse. Because products in a product-line are related (have similar functionality or user requirements), there is usually a degree of high commonality across a given product-line [Sonnemann 95]. Moreover, product-line have a sufficiently market lifetime so that an organization can make a profit from it.

3.2.3. Common Architecture

Description: Common architecture is a description of subsystems and components of a software family of systems and the relationships between them [Lamb88, Withey93, Kogut94, Leary94].

Justification: A common architecture across a product-line it structurally supports should allow an organization to more quickly develop a critical mass of reusable software components and leverage reuse.

Prior Model: Yes

A common architecture provides standard interfaces and data formats, as well as common experience leveraged from earlier developments. The architecture defines the rules for developing software components. This aids in the interchangeability of reusable software components across the product-line. Just as important, a common architecture lessens the need to make reusable software components highly generic or flexible because the environment in which they will be used is well defined and fixed. The architecture links developed and reusable software components to the physical problem space (hardware, operating system, and application packages such as database or graphics).

A common architecture makes a solid base on which to build systems [O'Connor 94]. A standard architecture can lessen the impact of major changes in end-user requirements. [Staringer 94]

3.2.4. Software Component Quality Based upon Reuse

Description: Quality is a measure of how the system fitness for use and meeting the users satisfaction and expectation [Lim94].

Justification: Reused components have been tested in operational systems.

Prior Model: Yes

Based on the results of the software industry practices, the system quality is increased when it is developed from previously developed components. Reused components from working systems, should be more reliable than newly developed components. These components have been already tested and exposed to an actual operational environment. Component generalization for reuse should meet certain quality metrics and may be certified as having reached the required quality standards.

The effect of software component reuse is positive, principally because of the higher opportunity that reuse provides for fault reduction [Sonnemann 95].

Software reusability can be used to achieve much higher levels of software reliability by amortizing the debugging costs among products incorporating the reusable component.

3.2.5. Improved Practices

Description: The process of collecting quality (improved) current practices for software reuse as adopted and employed technology by software developers.

Justification: Making these reuse experience and knowledge available will provide the software engineers and managers with a realistic and practical models and lessons learned on how to analyze, predict, estimate, measure, adopt, manage, improve, and extend software reuse.

Prior Model: NO

Applying a certain technology, such as reuse, requires a change in attitude and needs to allow sufficient time for development team members to mature in their roles. Every developer must have the time to reach some level of maturated experience. Most developers need to be able to reuse their existing assets to build new ones and to learn something useful from their past experience.

Some valid experiences with preferred parts and group technology approaches in manufacturing can also be applied to software. For software, this suggests that we introduce design methods (or models) based on preferred (qualified) software parts. Commonality and standardized requirements lead to preferred parts. The most effective reuse programs concentrate on the identification and development of a small, high-quality set of needed and useful components. In fact, as the amount of a new product made from existing components increases, we observe corresponding improvements in costs, time, and quality [Griss 93].

3.2.6. Reuse Metrics

Description: A software reuse metric is any measurement that relates to a software system, process, or related documentation.

Justification: Estimates and measurements of these metrics can be used in the refinement of the software reuse projects and predict an associated products reusability.

Prior Model: No.

Metrics play an important role in the determination of quality for both the technical and organizational reuse factors. Metrics are distinguishing traits, characteristics, or

attributes. These attributes are both static and dynamic. Metrics fall into two classes: (1) Control metrics are used by management to control the software process and (2) Predictor metrics are measurements of a product attribute.

Considering the use of metrics for technical factors, for example, Fonash [Fonash93] grouped the software component reuse metrics into five major categories: general, quality, parameterization, coupling, and cohesion. Quality, parameterization, coupling, and cohesion are software engineering principles that correspond to the reuse attributes. The general category is for attributes that are not in the four software engineering categories.

The following are suggested control reuse metrics:
- Reuse team size (persons)
- Reuse percentage (Size)
  [Levels: 25% Low, 26-60% Average, 61-90% High]
- Reuse percentage may consist of the following categories:
  Reused percentage of developed components from other projects, Reused percentage of commercial off-the-shelf components, Developed percentage of components for reuse by other projects; and Developed percentage of components unique to the identified projects. Total = 100 %.
- Decreased level of development effort (Man-Month)
  [Levels: 25% Low, 26-60% Average, 61-90% High]
- Decreased level of development time (Month)
  [Levels: 25% Low, 26-60% Average, 61-90% High]
- Decreased level of time-to-market (Month)
  [Levels: 25% Low, 26-60% Average, 61-90% High]
- Increased level of product quality (Reliability = errors/KSLOC)
  [Levels: < = 2 High, 3-5 Average, > 5 Low]
- Maintenance cost reduction (Man-Month)
  [Levels: 3 times Low, 4-6 times Average, 7-up times High]
- Overall software development cost reduction (Man-Month)
  [Levels: 20% low, 21-50% Average, 51-up% High]
- Return-On-Investment (ROI= $)
  [Levels: 2-5 times Low, 6-10 times Average, 11-up High].

The following are other reuse metrics [McClure 92]:

Commonality of a Component: is used to measure how frequently a component recurs across a set or family of systems.

Reuse Threshold of a Component: is used to indicate the minimum number of times a reusable component must be reused to pay back the investment in creating/preparing it for reuse.

Reuse Merit of a Component: is used to measure the reusability of a component relative to the average reusability of a component of the same type.

Reuse Creation Cost of a Component: is used to measure the cost of creating/preparing a component for reuse. This includes the costs of purchasing the component, identifying it via domain analysis, and preparing it for reuse (i.e., generalization, standardization, certification, classification, documentation).

Reuse Usage Cost of a component: is used to measure the reuse-related costs incurred each time the component is reused. This includes the costs of finding, understanding, modifying/specializing and integrating the reusable component.

Reuse Maintenance Cost of a Component: is used to measure the costs of supporting a reusable component.

Degree of Commonality of a system (or Business Area): is used to measure the proportion of a system's (or Business Area's) components that are common.

Degree of Reusability of a system (or Business Area or corporation): is used to measure the proportion of a system's (or Business Area's) components that are reusable. Note that the Degree of Reusability is probably less the Degree of Commonality because it is not cost-effective to create and support for reuse all common components.

Reuse Target Level of a system (or Business Area or corporation): is used to indicate the minimum proportion of a system (or for a Business Area, the minimum proportion for systems in that Business Area) or (for a corporation, the minimum proportion for all systems in the corporation) that is reusable.

3.3. Organizational Factors Influencing Software Development with Reuse

3.3.1. Management of Reuse

Description: Software reuse program management is an essential activity to make sure that software development based upon reuse is consistent with the organization's policies, goals, and requirements.

Justification: Successful reuse management is essential if a software project is to be supported by top management and to be developed on schedule and within budget

Prior Model: Yes.

In 1995 Sonnemann [Sonnemann 95] conducted an exploratory study of software reuse success factors. His research investigated the relationship between software reuse and many of the theories proposed by literature to provide greater reuse capability (success factors). Sonnemann identified several leading indicators of software reuse capability (success factors) including management commitment, investment strategy, business strategy, organizational structure, reuse process maturity, management that understands reuse issues, and software reuse advocate(s) in senior management.

3.3.2. Financial and Market Forecasting Factors

Description: Reuse technology must satisfy the organization's financial ground rules and impact the market forecasting strategy.

Justification: Financial constraints limit the range of opportunities for reuse within the organization and forces iteration in the market forecasting process.

Prior Model: No

The funding profile for reuse projects is quite different from conventional software projects. Typically, several years of up-front investment are needed before payoff is realized. Managers are reluctant to make long-term investment without some guarantee of success. They must consider market opportunities versus investment risks. The solution may be to develop return-on-investment (RoI) models. For example, given an initial investment cost (Io) in a corporate reuse program, an enterprise would expect to break even at some point in time of its product-line life cycle phase.

Software reuse is not a new strategy; however, the current emphasis on systematizing reuse and focusing such efforts toward narrow, well-defined application areas, or product lines is novel. This restricted focus enables reuse of large-scale components, such as software architectures and entire subsystems, that best leverage the knowledge, expertise and resources within an organization, thereby enabling greater future payoff after initial investment in software reuse.

If software developers do market analysis and forecast user needs, they should be able to predict user needs [Griss 93]. Software reuse programs require significant investment and substantial time to mature. Middle managers have been reluctant to adopt software reuse because of the up-front costs and slow return on investment (three to four years) [Joos 94].

A certain number (N) of reuses of components are necessary to break even on any reuse investment (I).

3.3.3. Market Place Factors

Description: The future of the software-intensive-systems marketplace belongs to organizations that can profitably deliver high-quality products in shorter time with less cost in response to diverse or rapidly changing customer needs.

Justification: Shorter production cycles, higher quality, and cost reduction can have great impact on overall profit and competitive advantage and market place than non-reuse development, or even cost-focused reuse-based development.

Prior Model: No.

Market place, domain modeling, and product-line approach are related and all are required to enhance and support the software engineering business strategy based on reuse. Although, these are potential factors that influence both software productivity and quality, many of the previous software cost or quality estimation models did not take them into consideration. Yet, one of the important motives of reuse is the potential for leveraging valuable assets as part of the effort in developing quality products, as well as leveraging time to introduce products to markets.

Prior to entering the software reuse market, a business should perform a reuse capability analysis. The objective is to maximize the actual reuse within an identified target for reuse opportunities with respect to business capabilities. The software asset producer, broker, and consumer roles are important, separable aspects of reuse that form distinctive patterns of activity within the reuse engineering idiom [STARS 94].

If a software developer's products are of a higher quality and produced at a lower cost than its competitors, it should be able to gain a larger share of the market or at least increase profit [Frakes 93] [O'Conner 94].

Software reuse will help software developers to have better, more accurate, and early bid estimates, because they know more about the system because a significant portion will be built from reusable components. Software development and building new systems out of reusable components will bring systems to market faster [Frakes 92].

3.3.4. Training Factors

Description: Classroom training supplemented by literature and additional courseware tools are essential educational vehicles for reuse technology success.

Justification: Training is a way of insuring that engineers have the necessary software reuse skills to work on a particular project.

Prior Model: Yes.

An organization with a corporate reuse education program has significant higher median levels of code reuse [Sonnemann 95]. Technical, organizational, and educational infrastructure is essential to reuse, and must be designed and managed to support it [STARS 94]. As technology and business culture changes, it is essential that a continual program of training about evolving technological and organizational factors be instituted for software organizations. Often, the press of current project schedules and pre-existing "fire-fighting mentality" cause attempts at training to be short-circuited. Most software professionals recognize the need for training, but subordinate it to the need to "get projects out the door."

For example, the object-oriented technology insertion stage of transition defines the development environment and allocate the required resources. The critical activities during this stage include defining techniques, selecting tools, and training the development team. [Fayad 96]

Software engineering requires: three kinds of education should be conducted:
- Generic methods and concepts, directed toward both management and technical staff to focus on generic software engineering concepts and methods;
- Tools and technologies, provided for practitioners and focusing on a specific software tools and its use; and
- General business practices, intended for all business professionals, management, and staff to improve their skills and communication [Pressman 88, 92].

4. RESULTS

This section begins in Section 4.1 by describing the results of the 1997 software reuse case studies and survey. These results are used in Sections 4.2 and 4.3 to answer the five research questions listed in the previous section 2.2.2.

4.1. Summary of Case Studies Empirical Results.

Our case studies research was comprised of examining data collected from 27 cases. Of these 27, we were able to collect complete data in 25 of the cases, but data from 2 of the 27 cases (Orga, Orgb in Table 1) was not complete. Of the 25 cases completed, for 1 organization we performed 2 cases. Hence, there were 24 organizations (Org1 to Org24). Considering our research [Nada 97] using the sample 27 case studies' results summarized in Table 1(a, b, c) we found the following to be true: (Interpretation of Case Studies Results)

1. The Software Reuse Reference Model (RRM) implementation levels RRML, derived from technical and organizational features, are predictive of effort, quality, and time-to-market.
2. The fact that reuse is still not a common practice in many companies is due to three factors. First, companies considering the introduction of reuse have to face economical and financial problems. Second, companies considering the introduction of reuse have to face organization-culture change impacts the software development life-cycle. Third, companies considering the introduction of reuse have to face assessment and evaluation of the actual benefits of reuse in terms of time and cost savings, improvement in the quality of products, and decreased effort.
3. When software is reused multiple times, the defects fixed from each reuse accumulate, resulting in higher quality. Furthermore, reuse reduces the time-to-market and effort needed to develop and maintain a software product, and increases the RoI.

Furthermore, Table 1-d summarizes the data presented in detail in Tables 1-a, 1-b, and 1-c. The survey sampled the population from four types of organizations with type identifiers as follows:

i. Small organizations with small projects (G11)
ii. Small organizations with large projects (G12)
iii. Large organizations with small projects (G21)
iv. Large organizations with large projects (G22).

4. From Table 1-d we see that large organizations sampled tend to use metrics, quality control and management of reuse processes more than small organizations sampled.

Considering the RRM factors and the corresponding case studies' results we conclude the following:

1. RRM Factors: Improved Development Practices, Management of Reuse Process.
Results: We found that software reuse at Org1, Org2, Org9, Org10, , Org11, Org12, Org15, Org18, and Org21 is more systematic compared to the practices at Org5, Org16, Orga, Org20, Org23 and Org24. The former have relatively higher investments in resources than the latter. We found that this is especially true when reuse is planned and implemented at the enterprise level, such as in the case of Org2, and Org11, Org12, (Table 1. Exp. Yr. and Team Size).

As an aside, these former organizations' reuse investments emphasize how a strong commitment on the management side is essential to reach reuse break even thresholds [Tracz86, 89, 94; Tirso 93; Troy 94; Sonnemann 95; SER 96].

2. RRM Factors: Reuse Metrics, Quality Control and Effort Models.
Results: We found that identifying and tracking certain reuse metric measurements at Org1, Org2, Org3, Org9, Org10, Org11, Org12, Org13, Org15, Org17, Org18, Org19, Org20, Org21 and Org24 gave them advantages over other organizations such as Org1, Org7, and Org8 in the formers' ability to assess and evaluate the actual benefits of reuse in terms of reduced time-to-market, improvement in products quality, increased productivity, and costs savings [Nada 97].

3. RRM Factors: Domain Model, Common Architecture, Product-line Approach, Best Development Practices and Experiences.
Results: We found that pilot projects carried out at Org1, Org2, Org9, Org10, Org11, Org12, Org15, Org16, and Org18 were successful in terms of the following:
a. achievement of the objectives set at the beginning of the projects;
b. identification of guidelines and methods for the development of reusable components (development FOR reuse);
c. identification of guidelines and methods for the development of new applications from reusable components (development WITH reuse);
d. construction of domain model, common architecture, or product-line approach [SER 96].

Table 1-a. Case Studies Summary

Organization	Domain	SW Type	Size KLOC	Team Exp.	Team Size	Focus
Org1 *	Laser Printers	Embedded	600 ESLOC	7	84	Family of Product
Org2	Avionics, C&C	Embedded	30	10	enterprise	Single Product-line
Org3	CASE Tools	Client/server	200	10	3	Family of Products
Org4	Synthesis	Tools	up to 100	6	enterprise	Family of Products
Org1 *	SW Generators	Tools	6	7	5	Vertical SW
Org5	Avionics/C&C	Embedded/RT	25	4	5	Family of Products
Org6	Healthcare	Real-time	NA	2	4	Single Product-line
Org7	Telecomm.	Client/server	8	3	NA	Single Product-line
Org8	Telecomm.	Embedded	256	19	NA	Family of Products
Org9	Avionics	Embedded	15-30	2	2-5	Single Product-line
Org10	Avionics	Embedded	15-30	10-11	3	Family of Products
Org11	Telecomm.	Data intensity	5-1000	10	enterprise	Family of Products
Org12	Banking	Client/server	2000 Classes	4	enterprise	Family of Products
Org13	MIS/Web Tools	Client/server	20-46	5	2	Vertical/Horizontal
Orga	Avionics	Embedded	135	5	15	Vertical
Org14	SW Tools	Client/server	250-300	4	5-8	Single Product-line
Org15	Avionics	Embedded	150-500	20+	4	Single Product-line
Org16	SWTools/Control	Client/server/RT	10-1000	1	6-10	Vertical/Horizontal
Org17	SW /Web Tools	Client/server	500-1000	5	12	Horizontal
Org18	Reuse Tools	Client/server	> 100	16	NA	Horizontal
Org19	Avionics	Embedded	88	5	8	Vertical (Prod.-Line)
Org20	Command and Control	Client/server	532	2	30	Single Product-line
Org21	Avionics/Gyro	Embedded/RT	10-20	25	1-12	Group of Products
Orgb	System Software	Distributed Objects	1000+/>80 Classes	4	< 5	Vertical/Horizontal
Org22	Avionics/Control	Embedded	50	15	~6	Family of Products
Org23	Cellular Network	Embedded	30-5000	3-4	5	Vertical (Prod.-Line)
Org24	Command and Control	Embedded/RT	20-88	4	enterprise	Family of Products

Table 1-b. Case Studies Summary

Organization	Reuse %	Less Effort	Less Schedule	Less T-T-M	Better Quality	Reuse Tech.
Org1 *	50-70%	Average	Average	Low-Average	High	NA
Org2	Average	Low	Low	Low	Average	Repository
Org3	High > 60%	Average	low	Low	High	Yes
Org4	Average	High	High	High	Average	FRITS
Org1 *	High	High	Average	Average	NA	Transformers
Org5	High	Average	Low	Average	NA	Domain Model
Org6	High	Average	Low-Avg.	Average	NA	UML, CORBA
Org7	Avg. 45%	Avg. 40%	Avg. 40%	NA	High	Domain/OLE/COM
Org8	50-90%	50-90%	50-90%	50-90%	NA	Domain Model
Org9	85%	Average	Average	Average	Avg.-High	EPICS
Org10	High	High	Average	Low	High	Repository
Org11	Avg.-High	Avg.-High	Average	Average	Avg.-High	Yes
Org12	High	High	High	High	NA	Repository
Org13	Average	Average	Average	Average	Average	Yes
Orga	High	High	Average	NA	NA	Domain Model
Org14	Low	Low	Low	Low	Average	Repository
Org15	NA (50+)	NA (50%)	NA (50%)	NA(60%)	High	Domain Model
Org16	30/80	30/75	Avg.-High	Avg.-High	NA-High	Transformers
Org17	Low	High	High	High	Average	Yes
Org18	High	High (84%)	High	High	Average	Frame
Org19	High (73%)	Avg.-High	High (70%)	High (80%)	Average	NA
Org20	Average	High (65%)	Avg. (28%)	Avg. (28%)	Low-Avg.	Yes
Org21	Average	High	Average	Average	Average	NA
Orgb	Low	Average	Average	Average	NA	Repository/UML
Org22	Avg. (30%)	NA	NA	NA	Low	NA
Org23	NA	NA	NA	NA	NA	NA
Org24	High (67%)	High (61%)	Avg. (33%)	Avg. (33%)	High (0.05)	NA

Table 1-c. Case Studies Summary

Organization	Reuse Metrics	Less Maintenance	Cost Reduction	ROI	CMM L.
Org1 *	Yes	Average	NA	2-4	2
Org2	Yes	Low (complex)	Low	Low	2-5
Org3	Yes	Average	Average	High	NA
Org4	NA	Average	Average	Average	NA
Org1 *	Yes	Average	Average	Average	2-3
Org5	Yes	Low	Average	Low	NA
Org6	NA	Low	Low	NA	2-3
Org7	NA	NA	NA	NA	3

Org8	NA	Average	Average	Low	1-4
Org9	Yes	TBD (Avg.)	TBD (Avg.)	TBD	NA
Org10	NA	Low	High	Average	NA
Org11	NA	NA	Avg.-High	NA	NA
Org12	NA	Avg.-High	High	NA	2
Org13	Yes	Average	Average	Average	NA
Orgas	Yes	NA	Average	NA	NA
Org14	NA	NA	Low	Low	2
Org15	NA	Average	NA	Average	3
Org16	NA	Low	High	High	NA
Org17	Yes	Low	Average	Low	NA
Org18	Yes	High	High	High	NA
Org19	Yes	High	Avg.-High	Average	NA
Org20	Yes	Avg.-High	High	Avg.-High	3
Org21	NA	NA	High	Average	1-2
Orgb	NA	NA	NA	NA	2-3
Org22	NA	NA	NA	NA	2
Org23	NA	NA	NA	NA	NA
Org24	Yes	NA	High (61%)	NA	3-5

Table 1-d. Organizations and RRM and Factors.

	COTS	DM	MRP	BP	Training	FM	Tools	PL	CA	Metrics	MP	CQ	QC	Org. Type
Org1	X	--	--	X	--	--	--	X	X	--	X	X	--	11
Org2		--	--	X	--	X	X	--	--	--	--	X	--	12
Org3		X	X	X	X	X	X	--	--	--	X	X	X	22
Org4	X	--	X	X	X	--	--	--	--	--	X	X	--	12
Org1		X	X	X	X	--	X	--	--	X	--	X	X	22
Org5		--	X	--	--	--	X	X	X	--	--	X	--	22
Org6	X	X	--	--	--	--	X	X	X	--	--	X	--	22
Org7	X	X	X	X	--	--	X	--	--	X	--	X	X	21
Org9	X	--	X	X	--	X	X	X	X	--	--	--	--	22
Org9		--	--	--	--	--	--	--	--	--	X	X	--	21
Org10		--	--	--	--	--	--	X	X	--	X	X	--	21
Org11		--	--	--	--	X	--	X	X	--	--	X	--	11
Org 12		--	X	X	X	X	X	--	--	--	--	--	--	21

Org 13	X	--	X	--	--	--	X	X	X	--	--	X	X	22
Org 4	X	--	--	--	--	--	--	--	--	--	--	--	--	22
Org 15		--	--	--	--	--	X	--	--	--	--	X	--	22
Org 16		--	X	X	X	--	X	--	--	X	X	--	--	22
Org 17	X	X	X	X	--	X	X	X	X	--	X	X	--	11
Org 18	X	X	X	X	--	X	X	X	X	--	X	--	--	22
Org 19		--	--	--	--	--	X	X	X	--	--	X	--	22
Org 20	X	--	X	X	--	X	--	X	X	--	--	--	--	22
Org 21	X	--	--	X	--	X	X	X	X	--	X	X	--	21
Org 22		--	X	--	--	--	X	X	X	--	--	--	X	22
Org 23	X	X	X	X	--	X	X	--	--	--	--	--	--	22
Org 24	X	X	--	--	--	--	--	--	--	--	X	--	X	21

Table 1 (d) . Entry Definitions.

X: Agree and Strongly Agree	FM: Financing and Marketing
--': Disagree and Strongly Disagree	PL: Product Line
COTS: Commercial Off The Shelf	CA: Common Architecture
DM: Domain Modeling	MP: Market Place
MRP: Management of Reuse Process	CQ: Component Quality
BP: Best Practices and Experience	QC: Quality Control

4. RRM Factors: Management of Reuse Process and Best Development Practices and Experiences.
Results: We found that pilot reuse project results at Org1, Org2, Org9, Org10, Org11, Org12, Org15, Org16 and Org18 convinced top management of the validity of reuse introduction on a larger scale. We also found that top management became more committed to support further efforts to widen the reuse culture and to spread reuse across the company [Sonnemann 95; SER 95].

5. RRM Factors: Reuse Metrics and Effort (Productivity) Models, Best Development Practices and Experiences and Management of Reuse Process.
Results: We found that pilot projects in Org1, Org2, Org9, Org10, Org11, Org12, Org15, Org16 and Org18 are monitored to determine the strengths and weaknesses in the adopted reuse implementation process to identify possible areas for improvement before introduction in more wide spread projects throughout the enterprise. At the same time cost saving trends are identified and the basis for a future Return-On-Investment (RoI) expectations is established.

As an aside, this incremental approach to reuse introduction is very sensible and usually recommended, but as far as the objective data that can be obtained from the pilot applications should always consider that they are usually only partial results. In other

words, real savings may be obtained only in the long term and there is no quick silver bullet that can lead to fully appreciate the benefits of software evolution and reuse programs for minimal costs, and create order of magnitude improvements in less than several years (3-5 years based on the SER experiences) [SER 95].

6. RRM Factors: Training, Management of Reuse Process, Market Place and Demands, Product-line approach, Common Architecture, Financing and Marketing Forecast.
Results: We found that reuse culture establishments within Org1, Org2, Org9, Org10, Org11, Org12, Org16, Org18 and Org24 required the following:

a. Training of development team members,
b. Management involvement in selecting pilot projects and development teams,
c. Management commitment and support,
d. Identification of future software product markets to leverage reuse.

7. RRM Factors: Management of Reuse Process, Financing and Marketing Forecast, Best Development Practices and Experiences, Reuse Metrics and Effort Models, Market Place and Demands, Case-Tools.
Results: We found that in Org1, Org2, Org9, Org10, Org11, Org12, Org15, Org18 and Org24 systematic reuse requires accurate analysis of organizational needs, management support, implementation of reuse measurements, identification of expected key benefits, and adequate technological support. Once the necessary organizational, methodological and technical issues related to reuse have been identified and put into practice, their reuse programs need appropriate automated support, especially when the number of the reusable assets grows over the threshold of few components and/or when the organization is geographically distributed.

8. RRM Factors: Use of Software Standards CORBA and DCOM, Reuse Metrics and Effort Models, Best Development Practices and Experiences, Case-Tools.
Results: We found that experiences in Org3, Org4, Org5, Org6, Org7, Org8, Org9, Org10, Org13, Org15, Org16, Org17, Org20, Org21, Orga and Org23 and demonstrated that the following problems are among the major inhibitor causes:

a. Difficulties in properly managing software components databases.
b. Lack of comprehensive standards and reuse metrics.
c. Immaturity of reuse technologies, and a lack of enough empirical data on effectiveness and efficiency improvements.

9. RRM Factors: Domain Modeling, Product-Line, Market Place and Demands, Management of Reuse Process.
Results: We found that in Org1, Org3, Org6, Org7, Org8, Org9, Org10, Org11, Org12, Orgb, Org16, Org18, Org20, ORP, and Org24 high achieved reuse rates are related to the following high potential reuse incentives:

a. Domain stability.
b. Market focus.

c. High products commonality, e.g. Product line or family.
d. Non volatile or highly constrained performance requirements.

10. RRM Factors: Management of Reuse Process, Domain Modeling, and Case-Tools.
Results: From the diversity of domain and size of the organizations studied we found no real influence in using certain methodologies over others.

11. RRM Factors: Management of Reuse Process, and Case-Tools.
Results: From the organizations studied we found that success in choosing certain reuse technologies or tools in the organizations studied depends on:
a. Organizational culture.
b. Use of pilot projects to evaluate technologies and tools.
c. Suitability and customizability of technologies or tools to the organization's environment.
d. Planned transition for the usage of the adopted technologies or tools.

12. RRM Factors: Management of Reuse Process and Quality Control.
Results: We found that Org1, Org12, Org18, Org20 and Org21 achieved a higher level of reduction in software maintenance effort by incorporating reuse in their managed software development life-cycle.

13. RRM Factors: Common Architecture, Domain Modeling, and Management of Reuse Process.
Results: We found that Org6, Org7, Org8, Org14 and Orgb develop their software products based on ad-hoc reuse approaches without any organized or planned attempt to exploit commonalties between applications, although significant overlaps in requirements between applications in the same domain usually exist. We found that ad hoc reuse of parts from existing applications occurred, even reuse of an entire application by adapting it to satisfy a slightly different set of requirements.

14. RRM Factors: Common Architecture, Domain Modeling, and Product-Line.
Results: We found that in Org1, Org2, Org9, Org10 Org11, Org12 and Org16 applications form part of an informal application family, and could have been created from common architecture, with a set of reusable components providing the tailoring for specific product requirements and platforms.

As an aside, this common architecture approach offers many benefits in terms of:
- Reduced development effort, owing to the fact that development work which has already been carried out elsewhere will not be repeated;
- Reduced development time, i.e. time-to market because there is less to do;
- Better quality, because new applications is built from proven parts;
- Consistency of applications' content because they are sharing the same components.

15. RRM Factors: Domain Modeling, Common Architecture, and Product-Line.

Results: We found that Org1, Org2, Org9, Org10, Org11, Org12 and Org16 used the domain engineering method to analyze and model the requirements in the domain, to identify what were stable and what were varied and assessed the applicability of the common architecture approach to the domain. The analysis of existing applications provided major input to this analysis. The analysis convinced these organizations that the common architecture approach would fulfill their needs. Also the models that were produced proved useful as a means to conciliate the views of different product teams.

16. RRM Factors: Common Architecture, Domain Modeling, Product-Line, Software Standards CORBA or DCOM, and Market Place and Demands.
Results: We found that Org1, Org4, Org9, Org10, Org11, Org12, Org15, Org16, Org17, Org18 and Org24 achieved faster time to market and/or improved quality, because they decided to base future development on common architecture and product-line approaches. Their objectives were to develop one common architecture for several types of applications, and in have it to comply with the standards of more than one market.

4.2. Summary Interpretation of Corresponding Survey Results

The survey of industry, academia, and government software developers was carried out to collect supporting data for two theses about the software RRM and to determine whether or not the RRM factors are applicable across organizations and are predictors of software reuse capability such as 'improvement.' It has been asserted that incorporation of such a RMM would promise three things:
1. Decreased products development effort;
2. Decreased time-to-market;
3. Increased product quality.

The survey examined 34 questions about the software RRM that incorporates both the technical and organizational elements that might be needed to establish improved software reuse practice in the organization.

A. Technical Factors by Question Number:

1. Technologies (or Tools e.g., Software Standards CORBA, DCOM, COTS), metrics, effort models that support reuse and Case-Tools.
Survey Questions: SQ14, SQ18, SQ21, SQ22, SQ34

2. Software development with reuse elements (technical procedures) such as: Domain Modeling, Product-line Approach, Common Architecture, Quality Control, and Best Development Practices experience.
Survey Questions: SQ3, SQ4, SQ5, SQ8, SQ11, SQ12, SQ15, SQ16, SQ17, SQ23-SQ25, SQ32, SQ33

B. Organizational Factors (Organizational Procedures) by Questions Number:

Management of Reuse Program, Market place, Financing Marketing Forecast and Training.
Survey Questions: SQ6, SQ7, SQ9, SQ10, SQ13, SQ19, SQ20, SQ21, SQ23-SQ25, SQ27-SQ31.
We also divided the survey population into four groups (types) as follows:
1. Small organizations with small projects (G11)
2. Small organizations with large projects (G12)
3. Large organizations with small projects (G21)
4. Large organizations with large projects (G22)

Ninety-three responses were received. Most of them were sent by on-line web site submission forms. Some other forms were submitted by personal interview, mail, or fax.
We tried to avoid having more than one project from the same organization. We sometimes accepted more than one project from the same organization if they each belonged in different categories (G11, G12, G21 and G22).

4.2.1. Technologies (or Tools e.g., Software Standards CORBA, DCOM, COTS), Metrics, Effort Models that Support Reuse and Case-Tools.

In this section and in section 4.2.2. that follows, for every survey question (SQ) we have placed following it in parentheses the RRM Factor (RRMF) or Factors that was/were used to categorize each SQ. The key words in each survey question were used from the definition of each RRMF as presented in section 3. Hence, let us turn to the first set of survey questions based on this correspondence.

Survey Questions: SQ14, SQ18, SQ21, SQ22, SQ34.

SQ14. We have an efficient requirements analysis tool that links our most common end-user requirements with the reusable software component(s) that satisfy them. (CASE Tools)
Findings: 62% of the projects 'do not use requirements analysis tool' that links their most common end-user requirements with the reusable software component(s)

SQ18. Our architecture has proved at least one of the following: CORBA/OLE/DCOM..., etc., to be useful means in standardizing interfaces and data formats. (Use of Standards CORBA, DCOM..., etc.)
Findings: about 50% of the projects did consider or approve at least one of the following:
(CORBA/OLE/DCOM...,etc.) to be a useful means of standardizing interfaces and data formats.

SQ21. Our configuration management system does an exceptional job keeping track of the projects that use each reusable software component.
(Management of Reuse Process, Reuse Metrics, Effort Models, and Case Tools)
Findings: 57% of the projects keep track of the projects that use each reusable software component.

SQ22. Identify the percentage of reusable and unique software components used in the identified project.

____ % Reused developed components from other projects.
____ % Reused Commercial Off-The-Shelf components.
____ % Developed components for reuse by other projects.
____ % Developed components unique to the identified projects.

Total = 100 %.

Findings: 59% of the projects reuse about 41-80% from other projects and COTS according to the following percentage reuse of assets:

Level 1, 0-20,
Level 2, 21-40,
Level 3, 41-60,
Level 4, 61-80, and
Level 5, Greater Than 80.

SQ34. My organization uses an effort estimation model that incorporates SW reuse as a factor. (Metrics and Effort Estimation Models)

Findings: 34% of the projects are using an effort estimation model that incorporates software reuse as a factor

4.2.2. Software Development With Reuse Activities (Technical Procedures) including: Domain Modeling, Product-line Approach, Common Architecture, Quality Control and Best Development Practices Experience.

Again, for every survey question (SQ) we have placed following it in parentheses the RRM Factor (RRMF) or Factors that was/were used to categorize each SQ. The key words in each survey question were used from the definition of each RRMF as presented in section 3. Hence, let us turn to this second set of survey questions based on this correspondence.

Survey Questions: SQ3, SQ4, SQ5, SQ8, SQ11, SQ12, SQ15, SQ16, SQ17, SQ23-SQ25, SQ32, SQ33.

SQ3. If you have a domain engineering section supporting the identified project, do you believe that the section support is (was) very effective and efficient. (Domain Modeling and Product Line)

Findings: 42% of the projects do not have a domain engineering section supporting the identified project at all. (No Domain Modeling) 38% of do have a domain engineering section.

SQ4. If you have software reuse repository support to the identified project do you think that the repository support is (was) effective and efficient. (Domain Modeling and Product-line)

Findings: 25% of the projects do not have software reuse repository, 41 % of the respondents who do have software reuse repositories think that the repository support is (was) effective and efficient.

SQ5. The organization has developed similar products in the past (in context of the identified project) and they are well understood by project team members. (Best Development Practices Experience.)
Findings: 47% of the projects have developed similar products in the past (in context of the identified project) and they are well understood by project team members.

SQ8. Pilot projects were used effectively to refine software reuse "best practices" prior to adopting them for routine use. (Best Development Practices Experience.)
Findings: 45% of the projects have used pilot projects effectively to refine software reuse "best practices" prior to adopting them for routine use. This confirms RQ5 and supports our findings from the case studies. (Best Development Practices Experience.)

SQ11. Projects document their software reuse lessons learned, which in turn are used to improve the organization's software reuse process. (Best Development Practices Experience and Management of Reuse Process)
Findings: 35% of the projects document their software reuse lessons learned, which in turn are used to improve the organization's software reuse process. 10% do not practice any documentation about reuse lessons leaned at all.

SQ12. Our software reuse efforts are concentrated on the identification and development of a small, high-quality set of needed and useful components. (Reuse Quality and Quality Control)
Findings: 59% of the projects considered reuse of a high-quality set of needed and useful components.
SQ15. Our software development process is built around a core set of reusable software components as the foundation for our products. (Common Architecture, Domain Modeling, Product line, Marketing Forecast, Management of Reuse Process)
Findings: 76% of the projects considered a core set of reusable software components as the foundation for their products in their development process. Many of the projects (57%) considered that during the requirement analysis phase, 43 % during the design phase, and 38% during coding phase (SQ26 - SQ28).

SQ16. We have done a thorough domain analysis of our products. (Domain Modeling)
Findings: 39% of the projects do practice a thorough domain analysis, 40% do not.
****** This confirms the SQ3 findings about Domain Modeling***

SQ17. We rely heavily on a common architecture across our product line. (Common Architecture and Product-Line)
Findings: 59% of the projects rely heavily on a common architecture across their product line. 25% do not use the common architecture or product-line approach.

- Software reuse promises numerous benefits. Rank your organization top

three motives for implementing software reuse. Please select from the list of available choices.
1. Better bid estimates.
2. Reduce development effort and costs.
3. Faster time to market
4. Better predict user needs.
5. Increase market share.
6. Successfully enter new markets.
7. Increased commonality across projects.
8. Better understanding of customer requirements
9. Improved quality.
10. Decreased product maintenance.

SQ23. Primary Motives ___

SQ24. Secondary Motives ___

SQ25. Tertiary Motives ___

(Common Architecture, Quality Control, Market Place, Marketing Forecast, Reuse effort Models, Management of Reuse Process)

SQ23. Findings: 59% of the projects considered the development effort and costs reduction as their primary reuse motive. 10% considered the faster time to market their primary reuse motive. 8% of the projects considered the better bid estimates their primary reuse motive.

SQ24. Findings: 29% of the projects considered the faster time-to-market the as their secondary reuse motive. 25% considered the increased commonality across projects as their secondary reuse motive. 18% considered the development effort and costs reduction as their secondary reuse motive. 9% of the projects considered the improved quality their secondary reuse motive.
SQ25. Findings: 19% of the projects the increased commonality across projects as their tritary reuse motive. 17% of the projects considered the improved quality their tritary reuse motive. 16% of the projects considered the decreased product maintenance their tritary reuse motive. 15% considered the faster time to market as their tritary motive
14% considered the improved quality their tritary motive

SQ23, SQ24, and SQ25 findings: the projects main reuse motives are:
- Development effort and costs reduction;
- Faster time-to-market;
- Increased commonality across projects;
- Improved quality.

SQ26. We have seen significant improvement in these areas (reference your three highest priorities identified in questions 23, 24, and 25). (Common Architecture, Quality

Control, Market Place, Marketing Forecast, Reuse effort Models, Management of Reuse Process)
Findings: 51% of the projects achieved significant improvement in the above mentioned areas.
****** This confirms our findings from the case studies***

SQ27. During (after) the project requirements phase we realized high commonality with the requirements of previous project(s). (Common Architecture, Domain Modeling, Product-line, and Management of Reuse Process)
Findings: 57% of the projects realized high commonality with the requirements of previous project(s) during (after) the project requirements phase.

SQ28. During (after) the project design phase we realized high commonality with the design of previous project(s). (Common Architecture, Domain Modeling, Product-line, and Management of Reuse Process)
Finding: 43% of the projects realized high commonality with the design of previous project(s) during (after) the project design phase.

SQ29. During (after) the project coding phase we realized high commonality with the code (documentation) of previous project(s) during (after) the project coding phase. (Common Architecture, Domain Modeling, Product-line, and Management of Reuse Process)
Finding: 38% of the projects realized high commonality with the code (documentation) of previous project(s).
****** SQ27, SQ28, and SQ29 finding: the projects tried to get most of reuse benefits by considering the reuse approach during the early phases of the software development life cycle.***

SQ31. Our decisions to bid on new projects or enter new markets are based heavily on our software reuse capabilities. (Market Place, and Financing and Marketing Forecast)
Finding: 36% of the project teams think that their decisions to bid on new projects or enter new markets are based heavily on their software reuse capabilities. (Market Place and Marketing Forecast)

SQ32. Our reusable software components have proven through operational use to be highly reliable. (Quality Control and Reuse Quality)
Findings: 60% of the projects reusable software components have proven through operational use to be highly reliable.
****** this confirms the case studies' findings.***

SQ33. We have a valuable process for certifying reused software components. (Quality Control and Management of Reuse Process)
Finding: 45% of the projects do not have a process for certifying the reused software components.

4.2.3. Organizational Activities (Organizational Procedures) by Questions Including: Management of Reuse Program, Market place, Financing Marketing Forecast, and Training.

Again, for every survey question (SQ) we have placed following it in parentheses the RRM Factor (RRMF) or Factors that was/were used to categorize each SQ. The key words in each survey question were used from the definition of each RRMF as presented in section 3. Hence, let us turn to this third set of survey questions based on this correspondence.

Survey Questions: SQ6, SQ7, SQ9, SQ10, SQ13, SQ19, SQ20, SQ21, SQ23-SQ25, SQ27-SQ31.

SQ6. Senior management has demonstrated a strong support for software reuse by allocating funds and personnel over a number of years. (Management of Reuse Program, Financing Reuse)
Findings: 56% of the projects senior management has demonstrated a strong support for software reuse by allocating funds and personnel over a number of years.

SQ7. There is a strong influential individual(s) in senior management who actually advocates and supports developing a software reuse capability. (Management of Reuse Program)
Finding: 52% of the projects agreed that there is a strong influential individual(s) in senior management who actually advocates and supports developing a software reuse capability.
****** SQ6 and SQ7 support each other***

SQ9. Our staff has a very competent software reuse education and training courses (internal and/or commercial). (Training)
Finding: 29% of the staff in the projects have a very competent software reuse education and training courses (internal and/or commercial).
****** Lack of training***

SQ10. Education for senior management is a very important aspect of our software reuse education and training program. (Training and Management of Reuse Program)
Finding: 51% of the project teams believe that the education for senior management is a very important aspect of their software reuse education and training program.

SQ13. We conduct market analysis of our products market place to effectively determine what domains to be modeled (or what COTS to be used) in order to develop reusable SW components to be placed in our software reuse repository. (Market Place, Market Forecast, Domain Model, COTS, Product-line)
Findings: only 29% of the projects conducts market analysis of our products market place to effectively determine what domains to be modeled (or what COTS to be used) in order to develop reusable software components
****** SQ13 finding confirms SQ3 finding.***

****** Low level of Domain modeling achieved***

SQ19. Software developers and maintainers precisely follow a software reuse process that is defined and integrated with the organization's software development process. (Management of Reuse Program)
Finding: 33% of the projects' developers and maintainers precisely follow a software reuse process that is defined and integrated with the organization's software development process.
****** Reuse process is not common and integrated part of most of the organizations' software development process.***

SQ20. Performance of the software reuse process is carefully measured and analyzed to identify weaknesses, and plans are established to address the weakness. (Management of Reuse Program, Best Practices and Experiences)
Findings: 24% of the projects measure and analyze the software reuse process carefully to identify weaknesses, and plans are established to address these weaknesses.
****** Lack of software reuse metrics***
****** This also confirms SQ19 finding about the lack of integrating the software reuse with the organizations' software development process.***

SQ30. Management has created new business opportunities that take advantage of the organization's software reuse capability and reusable assets. (Management of Reuse Program, Financing and Marketing Forecast, Market Place)
Findings: 34% of the projects Management has created new business opportunities that take advantage of the organization's software reuse capability and reusable assets.
****** Limited success of market forecasting based on reuse***
****** This confirms the findings of SQ24, and SQ31.***

Table 2 (a, b). represents the survey summary in terms of percentages of responses supporting RRM elements. High percentages (well above 50%) support the perceived importance of senior management support (Q6), reliance on a common architecture (Q17), importance of tracking and reliability of reusable components (Q21, Q32), reliance on common requirements (Q27), and the reliance of reuse in development effort and costs reduction.

Table 2.a. Survey Summary

Question #	Results
2	67% considered the problem addressed by the project to be difficult/very difficult
3	42% of the projects do not have a domain engineering section
4	25% of the projects do not have software reuse repository
5	47% developed similar products in the past (Best Practices and Experiences)
6	56 senior management has demonstrated a strong support for software reuse
7	52% strong influential individual(s) in senior management advocates software reuse
8	45% have an effective pilot project (s), this result matches with the case studies findings and RQ5
9	29% have a very competent software reuse education and training (* lack of training)
10	51% believe that the education for senior management is a very important issue
11	10% do not practice any documentation about reuse lessons leaned.
12	51% high quality products is one of primary motives of software reuse
13	29% projects do market analysis of their products market place and domain models
14	25% only use requirements analysis tool for reuse (the vast majority do not)
15	see SQ26-SQ28
16	40% do not practice domain analysis, see also SQ3
17	59% rely heavily on a common architecture
18	50% considered or approved (use of Standards/ CORBA, DCOM, ..etc.)
19	33% precisely follow a software reuse process as part of their software life-cycle
20	24% precisely follow a software reuse process to identify weekness and plans
21	57% projects keep track of each reusable software component.
22	59% projects reuse about 41-80% from other projects and COTS
23	59% The primary motive of reuse is development effort and costs reduction
24	29% the secondary motive of reuse is faster time-to-market
25	17% the tritary motive of reuse is improved quality product
26	51% achieved significant improvement in the above mentioned areas. SQ23-SQ25
27	57% realized high commonality with the requirements of previous project(s) (Req's)
28	43% realized high commonality with the design of previous project(s) (Design)
29	38% realized high commonality with the code (documentation) of previous project(s).
30	34% created new business opportunities that take advantage of their software reuse
31	36% decisions to bid on new projects or enter new markets are based on reuse
32	60% projects reusable software components have proven to be highly reliable
33	45% do not have a process for certifying the reused software components
34	34% adopting reuse effort models and metrics (lack of reuse effort models and metrics)

Table 2.b. Survey Summary

	Percentage of the Survey Replies
Level 1	5
Level 2	21
Level 3	45
Level 4	24
Level 5	5

Table 2.b. Translating Responses into Levels.

Level 5	Strongly Agree (Q3-21, 30-33)
Level 4	Agree (Q3-21, 30-33)
Level 3	Neutral (Q3-21, 30-33)
Level 2	Disagree (Q3-21, 30-33)
Level 1	Strongly Disagree (Q3-21, 30-33)

4.3. Research Questions

In this section five research questions stated in section 2.2.2 will be answered.

Research question 1. Is the implementation level of the software reuse reference model elements predictive of:

A. Products development productivity (effort) level
B. Time-to-market level
C. Product quality level

Based on the survey data analysis there is a correlation between the RRML (See section 2.2.3) and the organization improvement level of their software development effort, time-to-market, and quality. (Questions 23-26) This supports Thesis 1.

Of the 93 software development organizations in the survey sample 63 organizations reported one of the following three primary motives (See SQ23 of section 4.2.2):

- Reduce development effort and costs.
- Improved quality.
- Faster time to market.

The remaining 30 organizations reporting primary motives as being other than these three.

Of those organizations having as their primary motivation (goal) to increase software development productivity (reduced effort and costs) analysis of variance reported in Figure 1 shows a very high F value of 28 and a very low Prob > F value in comparing Reduced Effort to RRML.

Of those organizations having as their primary motivation (goal) to decrease (faster) time-to-market of developed software analysis of variance reported in Figure 2 shows a very high F value of 23 and a very low Prob > F value in comparing Less-Time-To-Market to RRML.

Of those organizations having as their primary motivation (goal) to increase developed software product quality analysis of variance reported in Figure 3 shows a more modest F value of 6 and a somewhat higher Prob > F value in comparing Increased Quality to RRML.

Because of the smaller sample sizes, it was decided to include statistical methods that do not require the assumption of larger samples. Hence, we included F-test values to justify the assumption of the equality of variances, which is needed in the t-test where the t-test is applied to test the differences between the means [Hoel 54] of the variables in our study.

This analysis leads to support of Thesis 1 and an affirmation of Research question 1.

Research question 2. Is the reuse percentage predictive of the implementation level of the software reuse reference model elements?
Based on the survey data analysis there is no correlation between the reuse percentage and the implementation level of the software reuse reference model activities

Research question 3. At which phase of the software life cycle organizations get the greatest reuse benefits?
Based on the following results:
- 57% of the projects realized high commonality with the requirements of previous project(s) at requirements phase.
- 43% of the projects realized high commonality with the design of previous project(s) at design phase.
- 38% of the projects realized high commonality with the code (documentation) of previous project(s).
We concluded that considering the reuse approach during the early phases of the software development life cycle organizations get the greatest reuse benefits. (Questions 27-29)

Research question 4. How should organizations collect and utilize reuse metrics and incorporate reuse effort estimation models as part of their software development life cycle?

Considering the survey data analysis we found that here is a lack of collecting reuse metrics and using effort estimation models with reuse.

Only 24% of the projects measure and analyze the software reuse process carefully to identify weaknesses, and have plans established to address these weaknesses.

Research question 5. Is a reuse process common practice and an integrated part of the organizations' software development process?

Considering the survey data analysis only 33% of the projects developers and maintainers precisely follow a software reuse process that is defined and integrated with the organizations' software development process.

4.4. Summary Interpretation of Corresponding QSM Study

Recall that a sample of 93 organizations were surveyed. Out of these 93 organizations 27 organizations agreed to be a sample for more in depth case studies. Out of these 27 case studies 19 were found to have features matching features in a related Quality Software Management (QSM), Inc. database of over 4000 software development projects collected Worldwide over a period of 15 years.

QSM helped us to compare these 19 selected organizational projects out of the George Mason University (GMU) data set with their database to further quantify benefits of software reuse. The data analysis demonstrated evidence of improvements realized through application of reuse.

With the QSM database we have carried out the following:

A. Determined schedule change and "industry average" Productivity Index (PI).
B. Determined schedule change and "average of the selected projects" Productivity Index (PI)
C. Compared the results in (1.) and (2.) above.
D. Found the schedule reduction for each industry domain
E. Found the effort reduction for each industry domain.

For example, in the Business domain the GMU with RRM projects average PI (19.6) is higher than the QSM industry wide average PI (17.3), and the schedule time to complete development has decreased.

The QSM study results are summarized in Table 3.

From the QSM study results we conclude that Thesis 1 is supported and that the software RRM implementation level is a predictor of (a) Products development effort (b) Time-to-market.

Table 3. Summary of the QSM Study Results

Industry Domain	*Project Productivity Index*	*Industry Productivity Index*	*Schedule Change*	*Development Effort Change*	*Project Productivity Index Change*
Business	19.6	17.3	Decreased	Decreased	Increased
System Software	13.9	13.6	Decreased	Even	Increased
Telecomm.	12.0	11.8	Even	Decreased	Decreased
Command & Ctrl	11.3	11.3	Even	Decreased	Even
Real-time	9.4	7.5	Decreased	Decreased	Increased
Avionics	12.3	8.3	Decreased	Decreased	Increased

Also, through this QSM analysis we found that the RRM is domain independent, meaning that the reference model benefit does not depend on any particular applications software domains.

5. SOFTWARE REUSE REFERENCE MODEL CONCLUSION AND CONTRIBUTIONS

This section contains summary interpretations of the research results presented in the previous section (Section 4).

The objectives of this research are: 1. The developing of a software RRM, 2. Empirically supporting the RRM, 3. Providing the empirical impact of the software RRM implementation level on the software effort, quality, and time-to-market.

To empirically support the model we used three different research approaches:
1. Carrying out 27 case studies of industry and government software developers;
2. Surveying the industry and government software developers;
3. Carrying out Quality Software Management (QSM) study of 19 selected GMU projects.

The organizations used in the empirical part of this reported research with the highest software reuse capability use the elements in RRM.

The empirical part of this research also supports RRM as domain independent in that the model does not seem to depend on any particular applications software domains.

The research results support the 1995 [Sonnemann 95] early study thesis that there is a set of improvement factors (software reuse leading indicators) that are common across

organizations and have some predictability relationships to software reuse. Based on the case studies and survey results, we confirm that the leading indicators of software reuse capability areas follows: product-line approach, architecture which standardizes interfaces and data formats, common software architecture across the product-line, design for manufacturing approach, domain engineering, management which understands reuse issues, software reuse advocate(s) in senior management, state-of-the-art tools and methods, precedence of reusing high level software artifacts such as requirements and design versus just code reuse, and trace end-user requirements to the components that support them.

This research also investigated to see if software reuse had a predictive relationship to productivity, time-to-market and quality.

Based on the survey and the case studies' results we concluded that Thesis 1 is supported and that software RRML (See section 2.2.3) in the organization is predictive of products development process productivity, time-to-market, and product quality. So the organizations that scored high RRML achieved the following:

- Increased level of process productivity
- Increased level of product quality
- Decreased level of development time
- Decreased level of time-to-market.

These findings will help software developers in the implementation of the RRM. These findings will also serve as guidance for systematic improvement of reuse practices in an organization.

REFERENCES

Applied Expertise, Inc., Software Reuse Benchmarking Study: Learning from Industry and Government Leaders, Report for the US Department of Defense Software Reuse Initiative, January, (1996).

Baldo, J.; Moor, J. and Rine, D., Software Reuse Standards, Standard Views: ACM Perspectives on Standardization, 5(2), 50-57 (1997).

Booch, G., Object Solutions, Managing the Object-Oriented Project, Addison-Wesley, (1996).

Basili, V. and J. Musa, The Future Engineering of Software: a Management Perspective, COMPUTER, September, 90-96 (1991).

Bell Canada, Software reuse case study - TRILLIUM, ARPA Report, 13 September 1993.

Boehm, B., Software Engineering Economics, Englewood Clifs, Prentice-Hall, NJ., 1981.

Boehm and P. Papaccio, 'Understanding and controlling software costs', IEEE Transactions on Software Engineering, October, 14(10), (1987).

Boehm, B., and W. Scherlis, Megaprogramming (preliminary version), ARPA Paper, (1992).

Bollinger, T., and S. Pfleeger, Economics of Reuse: Issues and Alternatives, INFO SOFT, December, 643-653 (1991).
Card, D., and E. Comer, Why do so Many Reuse Programs Fail?', IEEE Software, September, 114-115 (1994).
Cruickshank, R., and Gaffney, J., The Economics of Software Reuse, REUSE-ECON-MODEL- 91128-MC, Software Productivity Consortium, Herndon, Virginia, (1991).
Cusumano, M., Japan's Software Factories: A Challenge to U.S. Management, NY, Oxford University Press, (1991).
Fafchamps, D., Organization Factors and Reuse, IEEE Software, September, 31-41 (1994).
Fayad, M., Tsai, W., and Fulghum, M., Transition to Object-Oriented Software Development, Communication of the ACM, February, 39(2), 109-121(1996).
Fichman, R., Kemerer, C., Object Technology and Reuse: Lessons from Early Adopters, IEEE Computer, October, 47-59(1997).
Fonash, P., Characteristics of Reusable Software Code Components, Ph.D. Dissertation, George Mason University, 1993.
Frakes, W., and Isoda, S., Success Factors of Systematic Reuse, IEEE software, Septmber, 15-191(994).
Ferrentino, A., SNAP: The Use of Reusable Templates to Improve Software Productivity and Quality, Template Software, Inc., Herndon, Virginia, 1994.
Fonash, P., Metrics for Reusable Software Code Components, Ph.D. Dissertation, George Mason University, Fairfax, Virginia, Fall 1993.
Frakes, W., Software Reuse Empirical Studies, Virginia Polytechnic Institute and State University, Department of Computer Science, Blacksburg, Virginia, 11 October 1992.
Frakes, W., and W. Riddle, Instituting and Maturing a Practical Reuse Program, International Perspectives in Software Engineering, Summer 2nd Quarterly, 18-26 (1993).
Frakes, W., and Isoda, S., Success Factors of Systematic Reuse, IEEE Software, September 15-19 (1994a).
Frakes, W., and C. Fox, Sixteen Questions about Software Reuse, Software Engineering Guide Draft Paper, November, 1-25 (1994b).
Frakes,W., and Fox, C., Quality Improvement Using a Software Reuse Failure Modes Model, Software Engineering Guild Draft Paper, November, 1-13 1(994c).
Frakes, W., and Fox, C., Sixteen Questions about Software Reuse, Communications of the ACM, 38(6), 75-87 and 112, June (1995).
Gibbs, W., Software's Chronic Crisis, Scientific American, September, 86- 95 (1994).
Gold, J., Reusabilty Promise Hinges on Libraries, IEEE Software, 13(1), January 86-92(1993).
Griss, M., Software Reuse: from Library to Factory, IBM Systems Journal, 32(4), 548-566 (1993).
Hoel, P., Introduction to Mathematical Statistics, John Wiley and Sons, New York, (1954).
Jacobson, I., Griss, M., Jonsson, P., Making the Reuse Business Work, IEEE Computer, October, 36-42(1997).
Joos, R., Software Reuse at Motorola, IEEE Software, September, 42-47 (1994).

Kiely, D., Are Components the Future of Software?, IEEE Computer, February, 10-11 (1998).
Kogut,P., and K. Wallnau, Software Architecture and Reuse: Senses and Trends, Tutorial for Tri-Ada Conference, 7 November (1994).
Lamb,D., Software Engineering: Planning for Change, Prentice Hall, (1988).
Leary, J., Information Architecture - an Architectural Basis for Evolution of Large Scale Software Systems, Software Engineering Institute (SEI) Paper for NPGS Workshop, 2 February, 1994a.
Leary, J., Information Architecture Notions, briefing slides presented to Defense Simulation and Modeling Office Meeting, 19 October 1994b.
Lim, W., Effects of Reuse on Quality, Productivity, and Economics, IEEE Software, September, 11(5), 23-30, (1994).
Lim, W., Managing Software Reuse', Englewood Cliffs, NJ. Prentice-Hall, (1998).
McCain, R., Introduction to Reuse Technology and Application, slides - Defense Systems Management College, September (1991).
McClure, C., The Three Rs of Software Engineering: Reengineering - Repository - and Reusability , Englewood Cliffs, NJ. Prentice-Hall, (1992).
Morisio, M., Ezran, M. and Tully, C., Introducing Reuse in Companies: a Survey of European Experiences, Proceedings of the 1999 Symposium on Software Reuse, ICSE-99, IEEE and ACM (1999)
Nada, N., Software Reuse-Oriented Functional Framework, Ph.D. Dissertation, George Mason University, Fall (1997).
O'Connor, J., et al., Reuse in Command-and-Control Systems, IEEE Software, September, 70-79 (1994).
Parnas, D., On the Design and Development of Program Families, IEEE Transactions on Software Engineering, SE-2, 1-9 (1976).
Patterson, F., Reengineerability: Metrics for Software Reengineering for Reuse, Ph.D. Dissertation, George Mason University, Fall (1995).
Pfleeger, S., Design and Analysis in Software Engineering, Part 1: The Language of Case Studies and Formal Experiments, Software Engineering Notes , (19) 4, 18-19 (1994).
Pfleeger, S., Design and Analysis in Software Engineering, Part 3: Types of Experimental Design, Software Engineering Notes, 20 (2), 14-16 (1994).
[Pfleeger, S., Design and Analysis in Software Engineering, Part 5 Analyzing the Data, Software Engineering Notes, (20) 5, 14-16 (1994).
Pressman, R., Making Software Engineering Happen: A Guide for Instituting the Technology, Prentice Hall, (1988).
Pressman, R., Software Engineering: A Practitioner's Approach, McGraw-Hill, (1992).
Quality Software Management (QSM), Inc., Report on Independent Research Study of Software Reuse: Using Frame Technology, September (1994).
Rine, D., Supporting Reuse With Object Technology, Special Issue on Software Reuse Processes and Products, October, Computer, 43-45, IEEE Computer Society (1997).
Rine, D., Development of Reusable Software for the World Wide Earth Observing and Earth Information System : EOEIS, October, Internet News, (1997).
Rine, D., and Sonnemann, R., Investments in Reusable Software: A Study of Software Reuse Investment Success Factors., The Journal of Systems and Software, 41, 17-32 (1998).

Robertson, P., and De Ferrari, Systematic approaches to visualization: is a reference model needed?, Scientific Visualization, Academic Press Ltd., Ch.18, 287-305 (1994).
SER, Software Evolution and Reuse Consortium (SER), Solutions for Software Evolution and Reuse, SER Deliverable SER-D2-A, (1995).
Sonnemann, R., Exploratory Study of Software Reuse Success Factors, Ph.D. Dissertation, George Mason University, Fairfax, Virginia, Spring (1995).
Sommerville, I., Software Engineering, 5th Edition, Addison-Wesley, New York, (1996).
Software Productivity Consortium (SPC), Reuse Adoption Guidebook, SPC-92051-CMC, Ver 02.00.05, Software Productivity Consortium, Herndon, Virginia, November (1993).
Staringer, W., Constructing Applications from Reusable Components, Software, IEEE Computer Society, September, 61-68 (1994).
STARS, Loral Federal Systems, Air Force/STARS demonstration project experience report, ARPA Report, July (1994).
STARS, Unisys Corporation, Army STARS Demonstration Project Experience Report, ARPA Report, 25 August 1994.
Tirso, J., and H. Gregorious, Management of Reuse at IBM', IBM Systems Journal, 32(4), 612-615 (1993).
Tracz, W., Software Reuse: Motivators and Inhibitors, Stanford University paper presented at the Workshop on Future Directions in Computer Architecture and Software, 5-8 May (1986).
Tracz, W., Where does Reuse Start?, transcript - keynote address for the Reuse in Practice Workshop sponsored by IDA, SEI, and SIGAda at Software Engineering Institute, Pittsburgh, Pennsylvania, 11-13 July (1989).
Tracz,W., Domain-specific Software Architecture Frequently Asked Questions, ACM SIGSOFT Software Engineering Notes, 19(2), 52-56, April (1994).
Troy, R., Software Reuse: Making the Concept Work, Engineering Software, Special Editorial Supplement, 16-20, 13 June (1994).
Wasmund, M., Implementing Critical Success Factors in Software Reuse, IBM Systems Journal, 32(4), 595-611 (1993).
Withey, J., Implementing Model Based Software Engineering in Your Organization: an Approach to Domain Engineering, technical report, CMU/SEI-93-TR, Software Engineering Institute, Pittsburgh, Pennsylvania, November 1993.
Zelkowitz, M. and Wallace, D. Experimental Models for Validating Technology, Computer, 31(5), 23-31, IEEE Computer Society (May 1998).