AN EMPIRICAL STUDY OF SOFTWARE REUSE: THE IMPACT OF THE OBJECT-ORIENTED PARADIGM AND HUMAN FACTORS

by

John Allan Lewis

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

April, 1991

Blacksburg, Virginia

AN EMPIRICAL STUDY OF SOFTWARE REUSE: THE IMPACT OF THE OBJECT-ORIENTED PARADIGM AND HUMAN FACTORS

by

John Allan Lewis

Committee Chairperson: Sallie M. Henry
Computer Science

(ABSTRACT)

Little or no empirical validation exists for many of software engineering's basic assumptions. While some of these assumptions are intuitive, the need for scientific experimentation remains clear. Several assumptions are made about the factors affecting software reuse. In particular, the object-oriented paradigm and various human factors are hypothesized to affect the successful reuse of software components. This dissertation describes a controlled experiment designed to evaluate the impact of the object-oriented paradigm and human factors on software reuse. The human factors under investigation include managerial influence and cognitive abilities. This experiment concludes (a) the object-oriented paradigm makes significant contributions to productivity, (b) language differences are far more important when programmers reuse than when they do not, and (c) the object-oriented paradigm holds a particular affinity to the reuse process, (d) reuse results in higher productivity than no reuse independent of language paradigm, (e) the level of management encouragement does affect the reuse process, and (f) the cognitive ability of visualization does relate to effective reuse.

Acknowledgements

My sincere thanks go to the following people, without whom this research would not have been possible:

Dr. Sallie Henry, my research advisor and friend, for her unwavering help from inception to defense.

My research committee: Dr. Rex Hartson, Dr. Dennis Kafura, Dr. Bob Schulman, and Dr. Bob Williges, who truly helped make this research a success with their guidance, knowledge, and support.

The subjects of the experiment, who cooperated fully with the myriad of demands I placed on them.

Sharon Mancuso, my fiancee, who was always there when I needed her. During this process she put up with more than anyone can possibly believe.

My parents, Len and Dorothy Lewis, for their support and love throughout my extended education. This is it. I'm finally going to find that four-letter word, "work".

Thank you all very much.

Table of Contents

List of Figures

List of Tables

List of Tables (continued)

Chapter 1: Introduction

1.1 Motivation

The use of precise, repeatable experiments to validate any claim is the hallmark of a mature scientific or engineering discipline. Far too often, claims made by software engineers remain unsubstantiated because they are inherently difficult to validate or because their intuitive appeal seems to dismiss the need for scientific confirmation.

Software reusability is one area where unverified claims exist. The potential for reusability to improve the software development process is immense, yet a variety of factors affect its impact. Several theories exist about factors which either help or hinder the reuse process, but for the most part these theories have not be verified. These influential factors must be better understood, and their impact on reuse clarified, before software developers can fully make use of the benefits of reusing software components.

The research described in this dissertation investigates some claims made about the factors which affect software reusability. A controlled experiment was designed and executed to evaluate the impact of specific influential factors.

1.2 Reusability

Studies related to software reuse are important because of the key role reuse assumes in improving software productivity. Brooks asserts that while no single development will result in an order-of-magnitude increase in productivity, software reuse is an area where the greatest results can be achieved because reuse addresses the "essence," as opposed to the "accidents," of the development problem [BROF87].

However, reusable software is not being exploited to its full potential. According to Freeman, the state-of-the-practice of reuse in the United States is embarrassing [FREP87]. On an evolutionary scale, he puts reuse technology in an awakening stage, slowly approaching an early utilization period.

Software reuse is a robust topic with many areas that could be investigated empirically. Several issues pertaining to reuse are consistently addressed in the current literature. These issues are:

- Method of reuse (composition vs generation),
- Problem domain analysis,
- Method of accessing components, and
- Abstraction level of reusable components.

The following subsections describe how these issues of software reuse relate to the current research.

1.2.1 Composition versus Generation

Two fundamental approaches to reuse can be described as composition technologies and generative technologies [BIGT87]. In composition technologies, concrete "building blocks" such as subroutines, objects, designs, etc. are integrated and modified to form a final target product. Generative technologies are more abstract in that the reused entities are often intertwined in the tools used to generate the end products. In such an approach, a problem specification is presented to tools which analyze the problem and derive a solution.

Generative technologies reuse knowledge pertaining to how a particular product is constructed and attempt to automate the process. For example, code patterns are used in application generators. These patterns are defined in the generator itself. Transformation systems apply rules (the reused entity) to generate the new state of the product [HORE84].

Unfortunately, generative approaches often promise more than they deliver. Usually, it is some representation of the solution which must actually be presented to the generative tool, which then merely modifies the form of the solution [BROF87].

Generative technologies are a powerful approach to reuse and deserve further research. However, the current state-of-the-practice of generative technologies is less advanced than composition technologies. "Building block" approaches are based on methods to find, modify, and integrate reusable components. Therefore, composition technologies are intimately connected to the practiced methods of developing software.

This research investigates the ability to use existing components to form an implementation of a new project. Therefore, this study addresses reuse from a composition point of view as opposed to a generative one.

1.2.2 Domain Analysis

Any given problem can be considered to reside in one or more particular domains. Characteristics of problems are used to group them into domain categories. For example, using BNF in language representation is one use of domain analysis [HORE84]. A compiler-compiler is a completely automated result of applying domain analysis. Research has shown that analysis of the domain characteristics can benefit the reuse process.

Neighbors defines the Draco approach to software development based on reuse [NEIJ84]. Draco uses a domain analysis as a fundamental description of problem areas. Domain languages are then used to describe these areas. A domain description has five parts: a parser, a prettyprinter, a set of source-to-source transformations, a set of software components, and domain-specific procedures to guide transformations. The refinement process restates the problem from one domain language into another.

This process involves a great deal of human effort. The domain analyst is responsible for examining needs and requirements of a particular domain while the domain designer specifies different implementations for objects and operations in terms of existing domains (reuse of analysis information). The system designer is responsible for checking refinement consistency, guided by assertions and conditions specified in the refinements.

The process of analyzing and describing a problem domain is difficult and time consuming. Only 10-12 fully usable Draco domains have been built.

Another example of the use of domain analysis comes from [LENM87]. An IBM study used "building blocks" of encapsulated, unmodifiable code and data, following the model of Ada generic packages. These components were the result of specific domain analysis. No investment was made in building blocks which did not have a specific, practical application. Reuse rates were seen from 10 to 50 percent, and quality of code as measured by defect rate improved dramatically.

The focus of this research is to investigate factors which affect software reuse. Domain analysis addresses a fundamental approach to the reuse process, but not a specific influential factor. Therefore, while domain analysis itself is not specifically addressed, the effect of domain influence is noted. The experiment is constructed such that a wide range of software domains is employed.

1.2.3 Component Access

Efficient, accurate methods to find and evaluate a potentially reusable component are essential to successful reuse. Several research endeavors have concentrated on this problem.

Prieto-Diaz and Freeman describe a faceted classification scheme based on reusability-related attributes [PRIR87]. Program functionality is expressed as the triple <function, object, medium> and the programming environment is expressed as the triple <system type, functional area, and setting>. Classification of a software component is described as a sextuple which is a combination of these two descriptions. A thesaurus is used for vocabulary control and a user-weighted conceptual graph is used for related terms.

A prototype library system using this classification scheme has been implemented. Once a group of potentially reusable components is found (based on the sextuple description), evaluation techniques are used to rank them in order of reusability. The ranking process uses metrics based on five reuse attributes: size, structure, documentation, language, and reuser experience.

Fischer describes several component access tools based on the object-oriented paradigm [FISG87]. Browsers are used to search the class hierarchy structure in order to find appropriate object descriptions. The object-oriented paradigm and its relationship to reuse is discussed in further detail in Section 1.4.

Reusable "clearing houses" have been proposed as a method of giving access to reusable components [LUBM88]. A clearing house is a huge database of potentially reusable code and/or documentation. Additions to the database are controlled to some degree and components are catalogued against some criteria. A given clearing house can be based on specific restrictions (such as language) or by functionality. The Ada Software Repository is an example of a clearing house with over 25.5 Mbytes of source code and over 10 Mbytes of documentation, accessible over ARPA net [TRAW87a]. One problem with clearing houses is that proprietary data and economic concerns make companies reluctant to freely provide additions to the database of components.

The relatively small set of reusable components have appropriate naming conventions and documentation designed for the reuser, and therefore do not warrant the use of a search and access tool. Furthermore, the experiment is conducted in a Unix environment, which facilitates the process of component identification and evaluation [KERB84] [BARB87]. Results will be based on the assumption that potentially reusable components can be found by some evaluation criteria and with an adequate level of efficiency. The small reuse set is appropriate because the specific nature of the components is controlled (see Chapter 2).

1.2.4 Abstraction Level

The level of abstraction of a reusable component is a logical measure of its distance from the running computer system. For example, requirements are more abstract than designs which are more abstract than source code. Freeman asserts that reuse efforts should concentrate on the total range of underlying information at the various levels software development process, not merely the deliverables [FREP83].

Reuse at higher abstraction levels has greater potential for productivity gains than reuse at the source code level. Previously developed components at high abstraction levels can be integrated into the target project development at an earlier time than less abstract ones. The earlier reuse affects all aspects of the continuing development process, and therefore results in greater impact. Reusing design information has been the fundamental point of some reuse strategies (e.g., [NEIJ84]).

Wegner describes abstraction and reusability as two sides of the same coin [WEGP83]. Both are concerned with capturing the reusable essence of a component, but emphasize different aspects of the relationship. An abstraction level determines a set of associated reusable attributes, and conversely each set of reusable attributes fundamentally define an abstraction level (the class of components which possesses the attributes).

The problem with reuse at higher levels of abstraction is that no standards exist for the fundamental representation of design and requirements concepts [BIGT87]. Source code has well-defined representations with specific construction rules. Designs and requirements have no such standard representations. Designs which can be machine processed are often too close to source code and suffer the same abstraction restrictions. Less syntactically specific design schemes can not be automatically processed and therefore the reuse potential is reduced.

The problems of representation and standardization are limiting and need further research. Furthermore, only limited code reuse has been achieved, implying that there are still influences that hamper successful reuse of entities which have specific representations.

Therefore, this research addresses the software reuse issue from the abstraction level of code reuse. For the factors of interest, code assessment is appropriate and facilitates the evaluation process. Designs and test plans are considered only in the assessment of the completeness of a project. The next section specifically addresses the reuse of source code.

1.3 Code Reuse

The feasibility of code reuse is given credibility by observations that immense amounts of code are redundant. By one estimate, as little as 15% of new code is unique, novel, and specific to individual applications [JONT84]. It makes intuitive sense to reuse those parts of previous projects which parallel current work and therefore put the majority of development effort into new aspects of the system. Reuse is also seen as a means to increase reliability and maintainability due to the reuse of high-quality, well-tested products.

Isolated success has been achieved in the area of code reuse. Lanergan reports that the Raytheon company has achieved up to a 60 percent reuse rate across developed products, resulting in a 50 percent increase in productivity [LANR84]. The Toshiba corporation in Japan uses a software factory development approach with well-defined evaluation baselines and management techniques [MATY87]. Reuse is an integral part of this development scheme. Tracz reports that the Japanese software factory approach results in the development of 800-3200 SLOC/month (new and reused code), compared to a rate of 100-500 SLOC/month in the United States [TRAW87b].

Developers and users of potentially reusable products are unnecessarily hampered because they lack specific knowledge concerning the many technical and human factors which influence software reusability [WOOS87]. These factors affect whether existing components can be:

- evaluated for potential reuse,
- adapted to the required situation, and
- integrated into the new product.

Furthermore, the cost of the reuse process must stay below the cost of the development process or the main purpose of reuse is defeated [BARB87].

These influential factors come from several sources. While we agree with Meyer [MEYB87] and others about the importance of the technical aspects of software reusability, we pointedly disagree with the assessment that managerial, organizational, and economic problems are less important and should be resolved as an afterthought of technological sophistication. Tracz specifically addresses the myth that software reuse is only a technical problem [TRAW88].

Several overall views of the reuse process have been presented in the literature [BARB87] [BIGT87] [FREP87] [HORE84]. These views provide the necessary framework from which to launch an empirical investigation in an area as vast as software reuse. Of course, not all factors which affect the reuse issue can be investigated in one study. Two areas of interest have been chosen due to their potential for increasing the effectiveness of the reuse process. The first area focuses on language issues. Specifically, the experiment compares the use of the object-oriented paradigm to the use of the procedural paradigm with respect to software reusability. The second area addresses human factors. The effect of managerial influence and inherent cognitive abilities are investigated relative to the reuse process. These two areas are described further in the following sections.

1.4 Object-Oriented Paradigm

Standard procedural approaches are common in current development practice. Using a procedural approach, good software engineering design and implementation can result in well-structured, reusable code. However, the reuse results may be incidental to other practices. Software researchers are currently searching for specific strategies which will result in effective, reusable components.

The object-oriented paradigm is promoted as an excellent software development approach [STRB88] and its effects on software reuse is one area which begs further empirical investigation. Studies of the object-oriented paradigm are important because, according to Biggerstaff [BIGT87], the object-oriented paradigm has a good balance between power and generality, and therefore compliment the reuse process. In his opinion, procedural based solutions also have a good balance, but are considered less effective than object-oriented solutions.

Tracz makes several points which tend to support the use of the object-oriented paradigm to promote successful reuse [TRAW88]. For instance, despite the software tools which currently exist to ease reuse efforts [FISG87], special tools are not employed in current successful software reuse situations and are not the answer to the problem. Reuse success comes from formalizations of process and product, which an object-oriented environment creates. Furthermore, unplanned software reuse is costly. Software must be designed for reuse, with an emphasis on interface and modularization. The object-oriented paradigm stresses these characteristics. However, Tracz admits that, due to the many factors involved, no language will solve the the reuse problem inherently, including an object-oriented one.

The fundamental characteristics of the object-oriented paradigm seem to meet the needs of the reusing developer. Encapsulation capabilities create self-contained objects which are easily incorporated into a new design [KERB84]. Data and functionality are enclosed in objects by a specific class definition, and are held private to an instantiated object of that class. If a component is close enough to the target project, or general enough such that it has multiple purposes, the components can be reused with little or no modifications.

The data-based decomposition of objects, resulting in class-hierarchies and inheritance, promotes reuse far more than the top-down approach which promotes "one-of-a-kind" development [MEYB87]. Inheritance is the property of deriving subclasses from existing (base) classes while inheriting properties of the base class. A specific class might not be appropriate in a given situation, but a new class could be derived from an existing one using the properties of the old class which apply.

Inheritance also provides a hierarchy of classes which can be used in the searching and evaluation tasks inherent in software reuse. This hierarchy provides a variety of abstraction levels on which to view the reusable components. According to Wegner, greater abstraction is the key to to greater reusability, and object-oriented languages provide abstraction far better than procedural languages [WEGP83].

Inheritance is considered to be a mandatory characteristic of an object-oriented language [CARL85]. While Ada demonstrates several inherent characteristics which positively affect software reuse [TRAW87a], it does not support inheritance, and therefore lacks a fundamental property which a true object-oriented language provides.

While the characteristics of the object-oriented paradigm and the qualities which support successful reuse seem to compliment each other, little empirical evidence has been given to support this relationship. Therefore, this research specifically addresses the object-oriented language factor.

The language aspect of this research follows a distinct path through Biggerstaff's framework of software reuse [BIGT87], which is reprinted in Figure 1. In his framework, overall reuse is divided into composition and generative technologies. While the generative approach has potential, it is not the focus of this experiment. This research addresses only the composition-based approaches. Biggerstaff further divides the composition technologies into two techniques, one emphasizing standard libraries and the other promoting specific organizational principles. The research described in this dissertation compares these two techniques.

Features	Approaches to Reusability				
Component Reused	Building Blocks		Patterns		
Nature of Component	Atomic and Immutable Passive		Diffuse and Malleable Active		
Principle of Reuse	Composition		Generation		
Emphasis	Application Component Libraries	Organization & Composition Principles	Language Based Generators	Application Generators	Trans-Formation Systems
Typical Systems	• Libraries of Subroutines	• Object-Oriented • Pipe Archs	• VHLLs • POLs	• CRT Fmtrs • File Mgmt	• Language Transformers

Figure 1: Biggerstaff's Reuse Framework

Specifically, this research compares a standard procedural approach to the techniques of the object-oriented paradigm. The experiment was designed and executed in order to measure the relative affects of a procedural language and an object-oriented language in terms of software reuse.

Similar experiments have been successful in making comparisons of the object-oriented and procedural approaches relative to other aspects of software development. Meyer provides an involved example of design and implementation using the object-oriented language Eiffel and compares it to a procedural based approach [MEYB87]. Another study determined that the object-oriented paradigm is quantitatively more beneficial than a procedural approach in terms of software maintenance [HENS90]. An interesting point made in that research is that subjects viewed the object-oriented techniques as more difficult to accomplish, even though all objective data supported the hypothesis that using it resulted in fewer maintenance tasks and reduced maintenance effort. This result clearly illustrates the danger of relying only on anecdotal evidence to assess software engineering technology.

1.5 Human Factors

The second area explored in this experiment are the human factors which affect software reuse. Despite any high technology used to improve state-of-the-practice reuse, it is the human mind which develops reusable products and decides if existing code should be reused. The study of human factors is a major psychological paradigm affecting the manner in which code is developed [CURB87]. Even though human qualities are often difficult to objectively quantify, they play a major a role and are therefore a primary issue in this research.

Two sources of human influence are addressed in the experiment. First, the research explores the role of the organization's commitment to the reuse process. Second, the experiment investigates the cognitive skills, attitudes, and experience of the reusing programmers. These secondary human factors will not be addressed specifically by the experimental design, but rather as additional analysis as the data permits.

In a realistic software development environment, a majority of the work performed and the techniques by which that work is accomplished is greatly influenced by the production manager. Management's attitude is consistently mentioned in current literature as an important factor governing the successful reuse process [BIGT87] [TRAW88].

Managerial influence is an external stimulus which affects the software developer in a variety of ways. If the manager's attitude is apathetic or weak toward reuse, then it is left to the individual programmers to decide what emphasis to give to reuse. If, however, management's attitude encourages the reuse process, programmers might put more effort into correctly performing reuse tasks. Freeman makes the point that managers and technical people need different forms of reuse training [FREP83].

Managerial influence raises another question. Can too much emphasis be placed on reuse? If programmers are pressured to reuse, inappropriate code might be integrated into the new system causing modification delays that otherwise would have been avoided.

The managerial attitude will also determine the amount of technical support which is available. If management is a strong supporter of reusability, potential components will be organized in some way and made accessible to the programmer. If potential components are easy to access and evaluate, the possibility of successful reuse is enhanced.

Next, consider the individual programmer. A programmer's attitude alone can have a tremendous impact on the reuse process. If a programmer believes that his code is superior, in general, to other people's code, he may not be inclined to use software developed by others even if it is applicable and well implemented. Likewise, the opposite attitude might lead to equally detrimental effects. A zealous reuser might bypass the development process to the point that not enough effort is put into the evaluation process. This could result in attempting to use inappropriate code and eventually lead to time delays worse than if new code had been written.

The "not invented here" syndrome has been identified as a possible influence in software reuse [TRAW87b]. Programmers may view reuse as an indication that their particular development skills are unnecessary and think that resisting reuse is a form of job security. Again, reuse education can teach programmers that the ability to successfully reuse is as important as other development abilities, but the problem needs to be fully understood.

The academic ability of the individual might also affect his ability to reuse. First, consider a talented software developer, as measured on some appropriate scale. He might realize that appropriate effort put into evaluation and modification of existing code can increase productivity in the long run. On the other hand, perhaps that same programmer will be inclined to write code from scratch due to his past success.

Now consider a programmer with less talent. He might be inclined to reuse code developed by others which is generally recognized as high quality. But the characteristics of a good programmer might be essential in the evaluation process and therefore the less talented programmer might make incorrect decisions which lead to eventual failure.

Academic ability may be a function of experience. Experts are better at encoding new information than novices [CURB89]. Such differences may also affect the ability to reuse. In the classification scheme presented by Prieto-Diaz, the experience level of the reuser is used as a "fuzzy" modifier of the other factors used to determine which component from a qualifying set should be reused [PRIR87].

Beyond the programmer's attitudes and experience, individual abilities may play an important role in whether a person can successfully reuse. Cognitive skills inherent in an individual could affect how well a programmer evaluates a software component for applicability and integrates it into a new situation. The cognitive skills investigated in this experiment are:

- Integration: the ability of a person to combine objects into a unified whole,
- Perception: the speed at which a person detects small differences in objects, and
- Visualization: the ability of a person to visualize changes in objects.

These particular skills were chosen because they are intuitively connected to the mental processes used when reusing software.

Measuring human factors is difficult, but feasible. Valid testing strategies can evaluate cognitive abilities. Many other independent variables are directly measurable. One possible list of such variables is given in [EVAG89] and include demographic, academic, experience, and behavioral variables. Some of these factors are difficult to quantify, but techniques exist to collect such data in a statistically valid manner. The specific data collected and the collection methods used are discussed in Chapter 3.

1.6 Summary

While the reuse potential of both the object-oriented paradigm and human factors have been promoted, little scientific experimentation has been conducted. The research described in this dissertation evaluates the impact of several well-defined factors from these areas which have been hypothesized to affect the reuse process.

Specifically, the object-oriented paradigm is compared to the procedural paradigm relative to the reuse process. Distinct levels of managerial influence are compared to determine its impact on reuse. Furthermore, programmer disposition and cognitive abilities are evaluated.

Chapter 2 lists the questions to be addressed in this research and describes the design of the experiment. Chapter 3 discusses the data collected and the collection methods used. A description the analyses which address the language paradigm questions is given in Chapter 4. The human factors analyses are discussed in Chapter 5, which deals with the managerial influence questions, and Chapter 6, which describes the programmer attitude and cognitive ability issues. Finally, Chapter 7 summarizes the empirical results and discusses future work in the area.

Chapter 2: The Reuse Experiment

2.1 Introduction

Given the fundamental purpose of this research, namely to empirically investigate particular factors which are hypothesized to affect reuse, the specific questions of interest were formulated and the format of the experiment was designed. Since the motivation for the empirical design is taken from the questions, the initial focus of this chapter is the research questions themselves.

From these questions the experimental design was established, providing an adequate, reliable framework in which to answer each question. Some questions are directly addressed by the subject - group breakdown, while the others are answered without regard to group division. Given the careful control needed to draw valid conclusions, specific steps are taken to reduce excess sources of variability in the experimental design.

The next section lists the research questions answered in this study. Then, Section 2.3 discusses the experimental design used to address the questions. Finally, Section 2.4 summarizes the experiment and the overall approach of the research.

2.2 Research Questions

The goal of the experiment is to answer the following questions with respect to the successful reuse of software components. For these questions and the corresponding analyses, productivity is defined as the inverse of the effort expended to produce a specific software product. Therefore, the less effort expended by a programmer to develop a given system, the higher his productivity. Effort is measured in several quantifiable ways and is discussed in Chapter 3. For clarity, the questions are divided into categories based on the primary factors addressed.

The first set of questions deals with the language paradigm issues, namely the comparison of the object-oriented approach and the procedural approach. The analyses pertaining to the language paradigm issues and the associated conclusions are discussed in Chapter 4.

Language Paradigm Issues

1) Does the object-oriented paradigm promote higher productivity than the procedural paradigm?

2) Does the object-oriented paradigm promote higher productivity than the procedural paradigm when programmers do not reuse?

3) Does the object-oriented paradigm promote higher productivity than the procedural paradigm when programmers reuse?

4) Does the object-oriented paradigm promote higher productivity than the procedural paradigm when programmers are moderately encouraged to reuse?

5) Does the object-oriented paradigm promote higher productivity than the procedural paradigm when programmers are strongly encouraged to reuse?

6) Does the object-oriented paradigm provide incentives to reuse above those of the procedural paradigm?

The next set of questions addresses the influence of the production manager. Chapter 5 discusses the analyses and conclusions of the managerial influence issues.

Managerial Influence Issues

7) Does reuse promote higher productivity than no reuse?

8) Does management's moderate encouragement to reuse promote higher productivity than no reuse?

9) Does management's strong encouragement to reuse promote higher productivity than no reuse?

10) Does management's moderate encouragement to reuse promote higher productivity than management's strong encouragement to reuse?

11) Does management's strong encouragement to reuse promote improper reuse activities?

12) Past what point, in terms of percentage of a component reused for a target system, is it no longer beneficial to reuse?

The final set of questions addresses cognitive abilities and attitudes of the individual programmers. The analyses and conclusions for these questions are presented in Chapter 6.

Programmer Cognitive Ability and Disposition Issues

13) Does a programmer's inherent ability to combine objects into an integrated whole relate to his ability to reuse effectively?

14) Does a programmer's inherent ability to quickly detect small differences in objects relate to his ability to reuse effectively?

15) Does a programmer's inherent ability to visualize changes in objects relate to his ability to reuse effectively?

16) Does a programmer's disposition toward reuse relate to his ability to reuse effectively?

17) Does a programmer's background experience relate to his ability to reuse effectively?

All of the research questions are answered by the careful application of an appropriate experimental design and appropriate analysis of valid and complete data. The design of the experiment is discussed in the next section. The data used to test the hypotheses is discussed in Chapter 3.

2.3 Experimental Design

Some reuse experiments employ hypothetical, question-and-answer situations where the subjects do not actually perform all the various tasks inherent in the reuse process. We believe, however, that to accurately determine influential factors, the experimental subjects must perform all of the following tasks: evaluating potentially reusable products, adapting them to the new situation, and integrating them into a functionally complete product. It is important to create, as accurately as possible, a representative situation while maintaining a valid experimental design [CURB80].

The experiment is based on a target system developed by each of a set of senior-level software engineering students. The use of students as subjects, while sometimes considered unrealistic, is justified in this case due to two overriding considerations. First, empirical evidence by Boehm-Davis indicates that students are equal to professionals in many quantifiable measures, including their approach to developing software [BOED84]. Although new techniques are learned and further refinement does occur, a programmer's basic approach and development habits are formed quite early in their professional development.

Second, given the amount of control necessary to execute this experiment, students are the only viable alternative. When employing the scientific method and focussing on specific variables of interest, it is imperative that other sources of variability be reduced or eliminated. Therefore, aspects of the development environment and outside influences need to be monitored and controlled. This level of control could not be accomplished in a commercial setting without making the experiment trivial. The efficacy of students as subjects is supported for within-subject experiments by Brooks [BROR80].

The subjects in the experiment developed a specified target system. The system specification is couched in the guise of computerizing a fictional company and is separated into two tasks. The specific functional operations making up the system were abstracted from commercial software development experience. Both tasks involve a variety of programming techniques including data management, numerical processing, and graphics.

Previous research investigating the factors affecting software reuse have concentrated on two issues: (1) the effect of software engineering characteristics of code components, such as readability, structured code, etc., and (2) the techniques used to find appropriate code components from a set of possible candidates. Neither of these issues are the focus of this study. Code quality was allowed to vary only within the controlled confines of "adequate" testing and software engineering standards. All completed projects were verified to meet a set of requirements concerning documentation, code quality, and functional correctness. Furthermore, subjects were given no special tools for searching or identifying candidate components. It is assumed that any assistance in this area would only improve the reuse results. This study focuses on language issues, specifically the comparison of the object-oriented paradigm to the procedural approach, and human factors, such as managerial influence and cognitive abilities.

In the experiment, reusable code components were made available to the subjects implementing the target system. To affect further control, the code component sets were specifically generated for this study. Therefore, the research consists of two phases. The first phase was preparatory, in which potentially reusable components were designed and implemented. The experiment was executed in the second phase, in which the target system was developed by a set of subjects, who are unrelated to the programmers who designed and implemented the reusable components. These two phases are described in the following sections.

2.3.1 Phase One: Component Development

Two sets of potentially reusable components were created during phase one. One set was implemented in a procedural based language, Pascal, and the other in an object-oriented language, C++. The choice of languages was not arbitrary. Specifically, we deliberately chose not to use C as the procedural programming language. A goal of the research is to make as clear a distinction as possible between the object-oriented approach and the procedural approach. Since C is basically a subset of C++, we feared the similarities of the two languages might cloud that distinction. C++ was chosen over Smalltalk because we believed the powerful programming environment of Smalltalk [GOLA83], not available to a Pascal programmer, might jeopardize the comparison. Ada was not used because it lacks the concept of inheritance, and is therefore not a true object-oriented language. Finally, both C++ and Pascal emphasize strong typing, thus controlling another possible source of variation.

The component developers had experience using Pascal and therefore no further emphasis was placed on it. Careful training in the object-oriented paradigm and C++ was conducted to emphasize proper design and implementation criteria. Practice tasks were given in C++ to equalize the experience factor.

Both component sets were implemented on Apple Macintosh II's running A/UX. They were designed to be functionally equivalent, though design and coding techniques naturally vary between the two language paradigms. Equivalence was guaranteed by ensuring that all code meet the same fundamental functional and error-handling requirements. All developed code was required to pass the same level of testing thoroughness.

Furthermore, all code was required to meet fundamental software engineering criteria. Source code with potential to be reused must be evaluated by the new system developer to see if it is similar enough in function to warrant reuse. The evaluation will be facilitated if the source code is readable, well documented, well structured and well designed. Tracz compares evaluating a code fragment to the process of buying a used car [TRAW87c]. What are its standard features? Its mileage? Maintenance record? Reputation? Therefore these characteristics of the reusable code components were held constant. The same developer created the Pascal and C++ version of a given component and were required to alternate the order of development for each successive component created.

Knowing the requirements of the target system to be implemented in the second phase, each component was designed to have a specific level of applicability. The levels of reuse can be described as:

1) completely reusable,
2) reusable with "small" changes (< 25%),
3) reusable with "large" changes (> 25%), and
4) not applicable for reuse.

With respect to the target system, the component sets were designed to contain elements from each reuse level. The 25% delimiters of levels 2 and 3 are only intuitive guidelines and refer to the amount of code that must be added, deleted and modified to create a component that meets the target system's requirements. Providing components which span a wide range of applicability ensures a realistic, verbose domain from which subjects evaluate and choose components.

Knowing the details of the final product, the component sets can be designed to contain elements from each reuse level. As an example, consider a code component needed in the final product to evaluate data from a quality control stress test. The numeric data are stored in a set of vectors and must be evaluated by a specific algebraic test. If a component is developed for reuse level 1, it should completely perform the evaluation as needed. Level 2 reuse might dictate that the evaluation procedure be fundamentally consistent with the target project, but differ in slight ways requiring straightforward changes, such as equation modification. If the component is designed to be a level 3 reuse candidate, the component would be somewhat similar in concept, but differ greatly in function. A level 3 component might do the evaluation very differently from the type of test required. Or perhaps the components perform similar evaluation techniques on data set up in a completely different manner (i.e. a tree). Finally, if level 4 is desired, no component will be developed that will remotely resemble the evaluation functionality.

2.3.2 Phase Two: Project Implementation

Using the two sets of components, independent subjects were assigned the task of implementing the target project. The subjects were divided into six groups, pictured as cells in Figure 2. Half of the subjects implemented the project in Pascal, the other half in C++. Furthermore, subjects within each language category are divided into equal thirds, defining the amount of reuse they should perform. One group could not reuse at all, therefore implementing the project using only newly developed code. A second group were encouraged to reuse components from the appropriate reuse set as they saw fit. The third group was instructed that they should reuse if anything remotely appropriate if available in the reuse set.

		Managerial Influence		
		Cannot Reuse	Moderate Encouragement	Strong Encouragement
Language Paradigm	Procedural (Pascal)	3.05	3.10	3.07
	Object-Oriented (C++)	3.13	3.20	3.20

Figure 2: Subject / Group Breakdown with Mean Grade Point Averages.

The rows in Figure 2 primarily address the language paradigm question. The columns represent various management influences. Comparisons of the groups will directly answer several of the questions presented in Section 2.1. Other questions are answered with analysis performed without regard to group breakdown.

The subjects were divided into the groups randomly, but were statistically blocked across their computer science grade point averages. The blocking was accomplished by dividing the students by grade point average into high and low groups, then randomly assigning subjects to blocks from alternating levels. This blocking was an effort to reduce variability within each group. Figure 2 also shows the mean computer science grade point average for all subjects in a given group. An anova test comparing the grade point averages of subjects showed no significant differences between groups ($p < 0.9795$).

The functional requirements of the system are divided into two equal tasks. The two tasks separate the functional aspects of the project into "employee management" and "business management" sections. Employee management functions deal with processes such as an employee database, payroll, security control, and cost center management. Business management functions are concerned with the details of shop floor control, quality control testing, warehouse management, and customer interactions. An introductory material section was provided with information both groups need to perform their respective tasks. The specifications for the system are reprinted in Appendix A.

Although the two tasks focus on different aspects of running a business, they were designed to be comparable in computational complexity. Both are divided into seven subtasks, each of which has a counterpart in the other task that are designed to require approximately the same amount of effort to develop. If preliminary analysis shows that results are not affected by the difference between the two tasks, this factor can be ignored in subsequent analysis, thereby increasing statistical power in addressing the questions of primary interest.

To further control this aspect of the experiment, half of the subjects designed and implemented the employee management task first, while the other half of the subjects designed and implemented the business management task first. Then each half switched, resulting in both system tasks being developed by each subject. This organization offsets any learning benefit of doing a particular task first.

2.4 Summary

The research questions presented at the beginning of this chapter establish the goals of the empirical investigation. Pertinent information is derived from the questions, yielding a list of variables needed to address each point and a foundation on which to base the experiment. For this research, two factors were chosen as the pivotal points on which the experiment would be based: the object-oriented paradigm and the managerial influence.

Other factors of interest, such as programmer disposition and cognitive abilities, are addressed outside of this primary framework.

Component sets were developed for Pascal and C++ in the initial phase of the research, with careful attention to functional equivalence between languages. A set of independent subjects were divided into groups with varying degrees of reuse incentive and language requirements. These groups used the component sets to develop a target system.

Data were collected during the development process to assess the productivity of each subject. Based on this information, responses to each research question can be formulated and conclusions drawn.

The next chapter describes the data collected in more detail and defines the collection methods used. The remaining chapters use the data to address each question and draw conclusions.

Chapter 3: Experimental Data

3.1 Introduction

Given a feasible and well-controlled experimental design, the rigorous analysis of data collected during the experiment determines the conclusions that can be made about the hypotheses. This chapter describes the exact nature of the data used in this experiment and the methods used to collect it.

The variables measured are directly related to the objectives of the experiment, represented by the questions in Chapter 2. Some information is used to measure the factors under investigation. Other data are used to assess the influence those factors had on the process or used to control the experimental design itself.

Data collection can be accomplished in several ways, depending on the nature of the variables themselves. In this experiment information was gathered straight from the experimental design, from objective measurements taken from subjects, and from data collected by the subjects themselves. Care was taken to ensure that the data collected was valid and complete.

The experimental data taken directly from the experimental design include such variables as language used and level of managerial influence. This validity of this data is assured by direct control of the experiment itself.

Objective measurements such as cognitive abilities were measured by tests given to the subjects. Validity is assured since the scoring of the tests was performed according to the specifications given in the test battery directions and no subjects were involved in the process.

Productivity and reuse data were collected by the subjects using tally sheets during the development process. To assure this data's validity, subjects were informed that their names would not be associated with these data, and that the values themselves would have no bearing on their course evaluation. They were also told that a negative impact on their course evaluation would occur if they did not record their development information honestly and completely. The tally sheets were coded such that no subject name was ever connected to particular data.

The next section defines the specific variables collected and describes their purpose. Some initial data analysis is discussed in Section 3.3. Finally, Section 3.4 summarizes the data related aspects of this research.

3.2 Experimental Data

Data collected during the experiment can be divided into two sections. One set, called factor data, quantifies the factors affecting reuse that are under investigation. The second set, called assessment data, is used to evaluate the effect of the factors defined by the first set. All of the raw data values collected are given in Appendix E. The two sets of data are described in detail below.

3.2.1 Factor Data

One factor under investigation is the effect of language paradigm on the software reuse process. As described in the previous chapter, the language paradigm differences are designed into the experimental group breakdown such that the reuse and development efforts can directly evaluate the independent effectiveness of the language used. Therefore, one factor data element is the language, Pascal or C++, used by each subject.

Similarly, one human factor under investigation is the effect of the managerial influence in a developer's working environment. Similar to the language paradigm factor, the managerial influence levels are designed into the subject - group breakdown. Again, the reuse and development efforts are evaluated relative to the empirical framework. Therefore, another data element collected is the level of managerial influence encouraging or discouraging reuse. The levels used to simulate the spectrum of reuse incentives are:

1) No reuse
2) Moderate encouragement to reuse at developer's discretion
3) Strong encouragement to reuse anything which remotely applies

Other human factors were directly measured by questionnaires. For example, questionnaires were used to obtain each subject's amount of experience in a variety of related activities at both a professional and academic level. This information could indicate a relationship between experience in various activities and the ability to reuse. If necessary, these data can also be used to explain differences in groups which was not remedied by random assignment of subjects to groups.

Another human factor measured by questionnaires was the subject's disposition toward reuse. Several questions quantify the subject's trust of his own and other developer's code, the subject's belief that reuse can save time, and whether the subject practices reuse in everyday development. This information could indicate a tendency to reuse based on a fundamental belief in the process.

The inherent cognitive abilities of the subjects are still another group of human factor data. These cognitive abilities were quantified using tests which come from a battery of factor-referenced cognitive tests developed by the Educational Testing Service [EKSR76]. Six tests were used to evaluate three cognitive skills: integrative processes, perceptual speed, and visualization, as discussed in Chapter 1. Two tests measuring each skill were given to each subject and their scores recorded. This battery of tests have been successfully used before in computer-related human evaluations [EWEJ81]. This information could indicate a relationship between an individual's inherent cognitive abilities and the ability to reuse effectively.

3.2.2 Assessment Data

The assessment data is used to evaluate the factors measured by the first set, or to assist the controlled design of the experiment itself. For example, academic ability as measured by computer science grade point average was collected and was used to randomly block across the groups in the experimental design. This was done to reduce extraneous variability in the groups.

Questionnaires were used to get a subjective feedback from the subjects after the experiment was over concerning the level of difficulty of using the tally sheets to record productivity and reuse data. This information is used to determine if there was any bias in the data due to uncomfortable or difficult recording procedures. A subjective evaluation of the relative difficulty of the two tasks was also collected in order to further assess task equality.

The main assessment data collected during the experiment measure the productivity of a subject during the implementation of the target system. Productivity and effort are considered to have an inverse relationship. Therefore, the less effort expended by a subject to satisfy the requirements of one task, the higher the productivity of that subject. The measurements of effort, and therefore of productivity, are divided into two sections as follows:

Main productivity variables

- **Runs** - The number of runs made during system development and testing,
- **RTE** - The number of run time errors discovered during system development and testing, and
- **Time** - The time (in minutes) to fix all run time errors.

Secondary productivity variables

- **Edits** - The number of edits performed during system development and testing, and
- **Syn.** - The number of syntax errors made during system development and testing.

Multiple productivity variables are used to get a complete picture of the development process. The Runs, RTE and Time variables, given their significance to the development process, are considered the main variables of interest. The Edits and Syntax Errors variables are gathered for completeness, but are not given overriding concern.

Since each subject independently implemented the same tasks, a comparison of data across subjects yields a relative measure of the effort used to develop a system task. The comparison is appropriate since each subject's task subsystem was required to meet a minimum set of functional requirements. A subject with a high value for a given indicator is considered less productive than a subject with a low value.

The group means for each productivity variable are depicted graphically in Figures 3 and 4. These charts give a rough indication of how the groups compare although statistical analyses are employed to verify perceived differences.

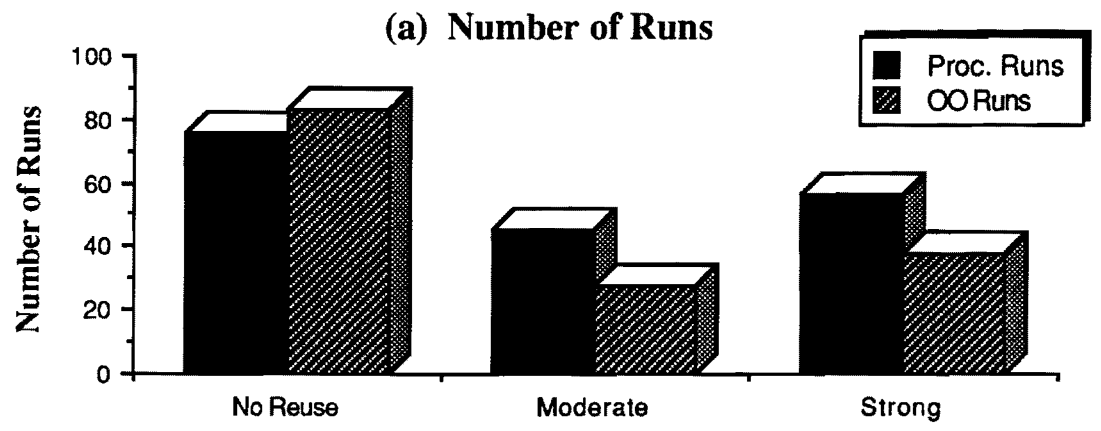
Data pertaining to the reusable components were also collected. First, a list of all possible components was compiled with its intended level of applicability. The intended use of the component (if any) was noted in terms of task and subtask id number.

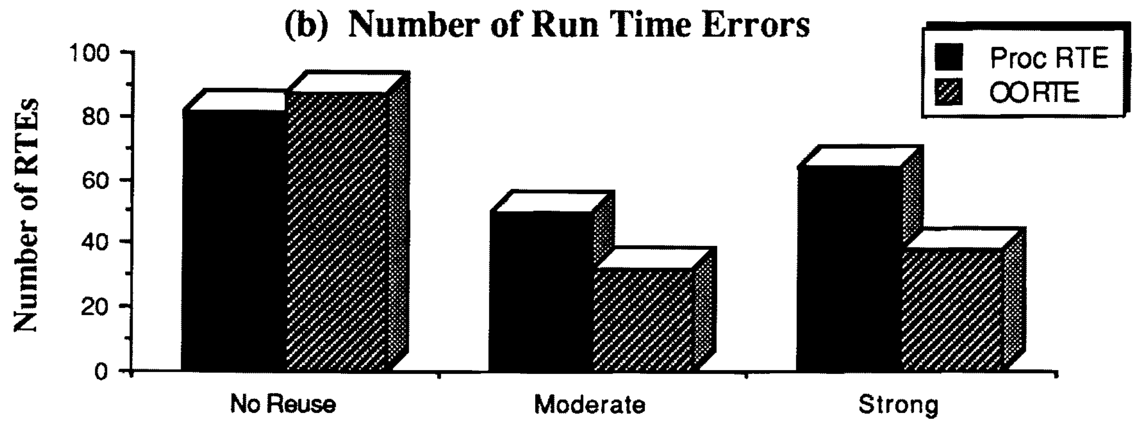
Each time a subject reused a component, the task and subtask was noted, and data were collected to determine the amount of work required to integrate that component into the target system. The percentage of original source code used was collected, along with the number of lines added, deleted and modified in three categories: cosmetic changes, data definition changes, and algorithm changes.

The data collected in reference to the reusable components and their use by the subjects can help determine how appropriate the developer was in choosing and integrating a component.

3.3 Preliminary Analysis

Some preliminary analyses were performed to determine the nature of further analyses and help assess the validity of the data collected. The experiment consists of a fundamental two-way design, which usually lends itself to a straightforward anova (analysis of variance) evaluation of the data to determine differences in group means. However, the research questions are highly specific and often involve combinations of groups to answer questions. Therefore, the tests were performed using linear contrasts of the six groups and are defined specifically for each question.





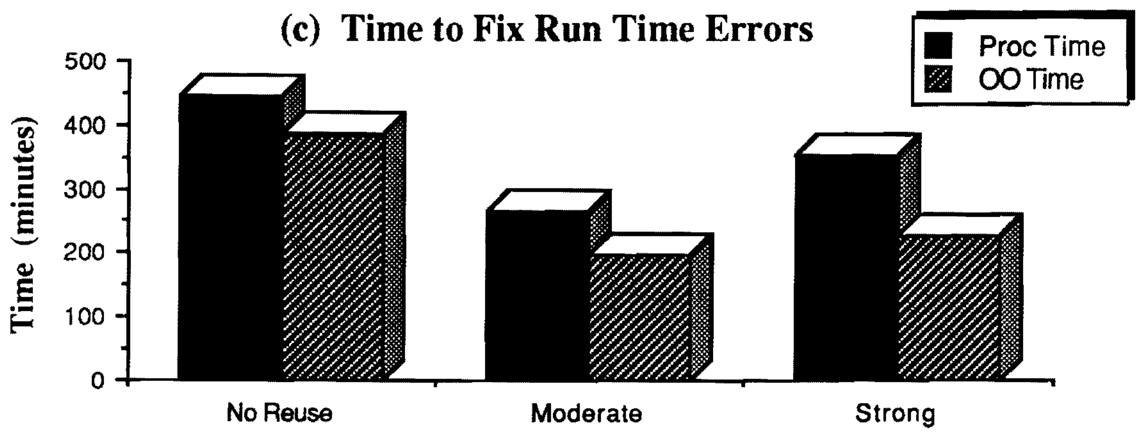
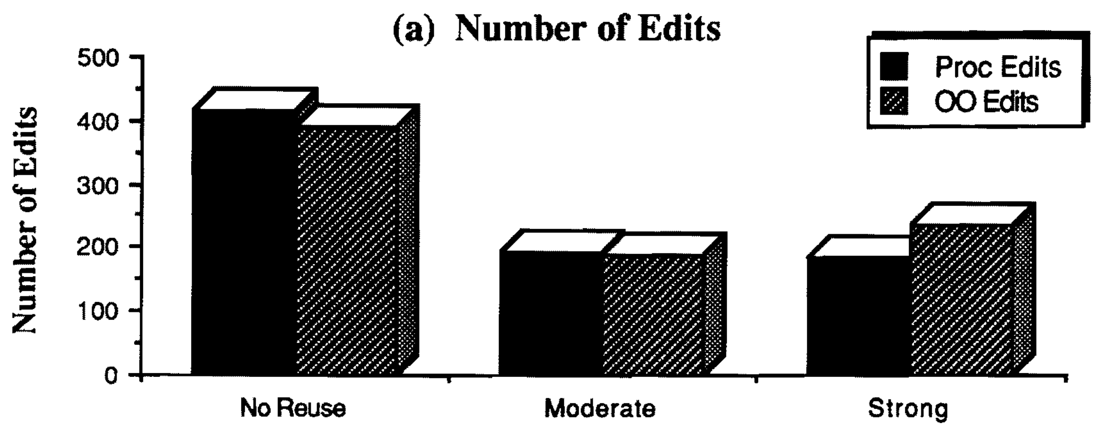


Figure 3. Group means for primary productivity variables.



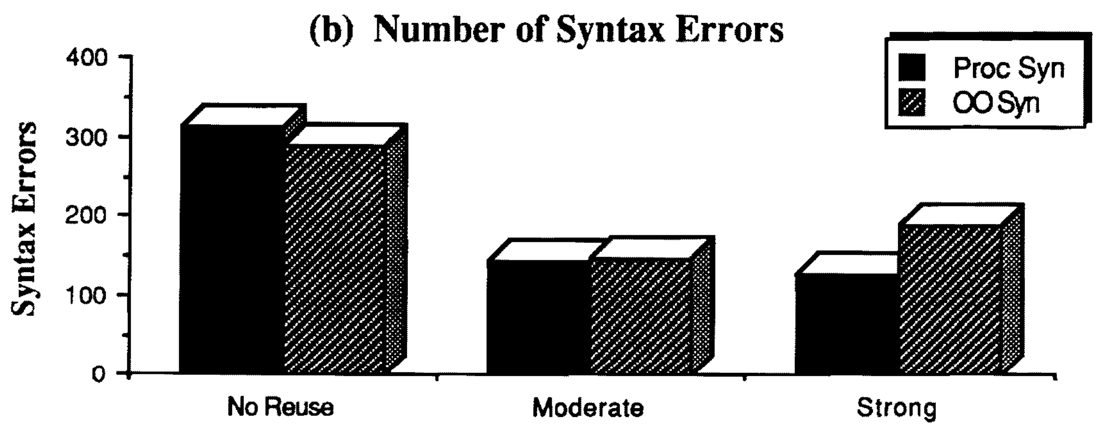


Figure 4. Group means for secondary productivity variables.

Furthermore, all hypotheses predict the direction of difference between means. In other words, all of the research questions are phrased such that a given factor is favored over another. Therefore, all tests were performed in a one-sided manner.

In all statistical tests performed in this research, a result was considered significant if the p-value for the test was less than 5% ($p < 0.05$), which is an accepted norm. The p-value is the probability that the difference could have been obtained by chance, rather than reflecting a true difference in group means. Following conventional criteria, a difference is deemed statistically significant if its p-value is less than 0.05. In such cases, it is extremely unlikely that the difference in means is due to chance.

In a very conservative approach, the 5% chance of error is distributed among all tests, effectively requiring much stronger significance for each individual test. The approach used in this research is more pragmatic, and allows a 5% chance of error on each separate test. Since a few significant results would be expected by chance, we will look for trends of significance in order to draw conclusions.

Many of the research questions are based on improving productivity during system development. The measures of productivity are discussed earlier in this chapter. Prior to the individual tests, a two-way manova (multivariate analysis of variance) test was run against the data for all five productivity variables to determine if fundamental differences exist which deserve further investigation. For the overall effect of language paradigm, the differences were not quite significant ($p < 0.1013$), but were large enough to suggest investigation on an individual basis. The overall effect of managerial influence ($p < 0.0001$) and the overall interaction effect ($p < 0.0414$) were both significant and therefore warrant further investigation. The detailed results of these tests are given in Appendix D.

As described in Chapter 2, the target system specification was divided into two tasks. Initial analysis of the task factor determined that the difference between the two tasks played no role in influencing any of the productivity variables (all p-values for task effects were ≥ 0.2073). In other words, the two tasks were determined to be equally difficult. The lack of difference is attributed to the careful design of task specifications and the blocking of subjects across grade point average. Therefore, all further analyses ignore the task factor, which gives them more statistical power.

Finally, the subjective feedback from the subjects rating the difficulty of the tasks and the questionnaires was examined. The difficulty was rated on a four point scale with 1 being easy and 4 being difficult. The average difficulty rating for task A was 3.14, and for task B was 3.23. This is consistent with the objective task analysis in concluding that the tasks were equally difficult. The average difficulty rating for the development questionnaire was 2.33, and for the reuse questionnaire was 1.86. This indicates that both data collection sheets were fairly simple to use and should not have biased the data recorded.

3.4 Summary

The data collected in this experiment were chosen to answer specific questions posed in Chapter 2. Some data were used to quantify the factors under investigation, while others were used to assess the productivity of the subjects or control the experiment itself.

The data collection methods were designed to assure accurate and complete measurement of the variables of interest. Some data were derived from the experimental design itself, some from objective testing methods, and some from questionnaires and tally sheets designed for ease of use and anonymity.

Initial analysis determined that the factors warrant further investigation which will be determined by the questions themselves. The tests performed are specific and involve multiple group comparisons, therefore linear contrasts were employed when appropriate. Conclusions are therefore based on trends of significance and not on single variables.

The next chapter describes the analysis and conclusions for the language paradigm factor. Chapter 5 concentrates on the managerial influence factor. The other human factors under consideration, namely programmer disposition and cognitive abilities, are examined in Chapter 6. Finally, Chapter 7 summarizes the conclusions of the research.

Chapter 4: Language Paradigm Analyses

4.1 Introduction

This chapter discusses the analyses which address the language paradigm issues under investigation. These issues were listed as the first set of experimental questions in Chapter 2.

For each question, a table is given which summarizes the analysis performed. In each table, the mean values for each productivity variable are listed. These means represent a group or combination of groups from Figure 2 as appropriate for each question. Also listed are the p-values representing the significance of the difference between the means. Remember that a result is considered statistically significant if the corresponding p-value is less than 0.05.

The primary productivity variables, Runs, RTE (Run Time Errors), and Time (in minutes to fix the run time errors), are listed first in the tables. The secondary productivity variables, Edits and Syn (Syntax Errors), are listed below the primary variables. Given their relative importance to the development process, the primary variables will be used to draw conclusions, although the secondary variables will be discussed as well. Keep in mind that we will look for trends of statistical significance in order to draw conclusions, as opposed to basing them on tests of a single variable.

4.2 Empirical Results

The experimental questions posed in Chapter 2 which focus on the language paradigm issues are used as a framework for discussion of the statistical analysis. Each question is addressed separately, giving the results of the appropriate analysis.

1) Does the object-oriented paradigm promote higher productivity than the procedural paradigm?

The third column in Table 1 list the means of the productivity variables calculated from all subjects using the procedural language, including all subjects who reused as well as those who did not. The fourth column shows similar means for subjects in the object-oriented categories. Our hypothesis is that the values for the object-oriented paradigm will be lower than those of the procedural paradigm, indicating a higher productivity for the object-oriented subjects.

Table 1. Language Paradigm Main Effect

			Means	
	Significant?	p-value	Procedural	O-O
Runs	Yes	0.0066	59.27	47.50
RTE	Yes	0.0078	65.00	50.20
Time	Yes	0.0104	354.41	261.70
Edits	No	0.3469	271.55	263.65
Syn.	No	0.8675	183.67	202.40

The three main productivity variables (Runs, RTE and Time) show a significant difference between the means, favoring the object-oriented paradigm. In addition, the object-oriented mean for the Edit variable was also lower than the procedural mean, although not to a significant degree. The means on the Syntax Errors variable did not differ in the predicted direction. Considering the nature of the Edits and Syntax Errors variables, the lack of significance is attributed to the subjects lack of practice using the object-oriented language. The results of the analysis on the main variables indicate that the object-oriented paradigm does promote higher productivity than the procedural paradigm.

2) Does the object-oriented paradigm promote higher productivity than the procedural paradigm when programmers do not reuse?

The means listed in Table 2 are calculated for only those subjects who did not reuse. The third column represents subjects using the procedural language, and the fourth column represents subjects using the object-oriented language. Our hypothesis is that the object-oriented values will be lower than the procedural values. Surprisingly, none of the variables indicate a significant difference.

Table 2. No Reuse (Procedural vs. Object-Oriented)

	Significant?	p-value	Means: Procedural No Reuse	Means: O-O No Reuse
Runs	No	0.8909	75.38	83.17
RTE	No	0.7506	81.25	87.17
Time	No	0.1607	446.38	385.00
Edits	No	0.2360	416.00	392.00
Syn.	No	0.1733	311.00	290.33

Interestingly, the group means do not consistently favor one language or the other. The means for the object-oriented groups are lower for Time, Edits, and Syntax Errors, but the means for the procedural groups are lower for Runs and RTEs. According to this analysis, we must conclude that when reuse is not a factor, the object-oriented paradigm does not promote higher productivity. In other words, when starting from scratch, either language works equally well.

3) Does the object-oriented paradigm promote higher productivity than the procedural paradigm when programmers reuse?

Given the answers to the first two questions, the answer to this question should logically be yes. The results in Table 3 confirm this for the three main productivity variables. The means listed are for subjects who did reuse, including both levels of managerial influence. The third column displays the means for subjects using the procedural paradigm and the fourth column shows the means for subjects using the object-oriented paradigm. Once again, our hypothesis favors the object-oriented paradigm.

Variables Runs, RTE and Time all proved significant with means favoring the object-oriented group, but the Edits and Syntax Errors variables did not differ in the hypothesized direction. Given the importance of the main productivity variables, we can conclude that the object-oriented paradigm does promote higher productivity than the procedural paradigm when reuse in employed.

Table 3. **All Reuse (Procedural vs. Object-Oriented)**

			Means	
	Significant?	p-value	Procedural All Reuse	O - O All Reuse
Runs	Yes	0.0001	50.07	32.21
RTE	Yes	0.0005	55.71	34.36
Time	Yes	0.0153	301.86	208.86
Edits	No	0.8380	189.00	208.64
Syn.	No	0.9767	137.14	164.71

Note that most of the support given to the first question comes from differences between the groups which were encouraged to reuse. However, the development effort when reuse is not an issue is not significantly different, while the reuse potential of the object-oriented paradigm is substantial.

4) Does the object-oriented paradigm promote higher productivity than the procedural paradigm when programmers are moderately encouraged to reuse?

This question and the next one further refine the analysis of the previous question by examining the language paradigm comparison within the context of distinct managerial influence levels. The means listed in Table 4 are for subjects from the moderate encouragement groups only, representing moderate encouragement by management to reuse. The third column represents the subjects using the procedural paradigm and the fourth column represents the subjects using the object-oriented paradigm. Our hypothesis is that the values for the object-oriented group will be less than the procedural group.

The Runs and RTE productivity measures were significantly lower for the object-oriented group. The means favored the object-oriented group for the Time and Edits variables as well, but not to a significant degree. The Syntax Errors variable barely favored the procedural paradigm.

Table 4. Moderate Encouragement (Procedural vs. Object-Oriented)

			Means	
	Significant?	p-value	Procedural Moderate	O-O Moderate
Runs	Yes	0.0023	45.13	27.75
RTE	Yes	0.0178	49.50	32.00
Time	No	0.1179	264.25	196.13
Edits	No	0.4660	192.13	189.50
Syn.	No	0.5688	143.25	146.75

Since two of the three primary measures significantly favored the object-oriented paradigm, and four of the five variables overall showed a tendency in the same direction, we conclude that when subjects are given moderate encouragement to reuse, the object-oriented paradigm does promote higher productivity than the procedural paradigm.

5) Does the object-oriented paradigm promote higher productivity than the procedural paradigm when programmers are strongly encouraged to reuse?

The means listed in Table 5 are for subjects from the strong encouragement groups only. The third column represents the subjects using the procedural paradigm and the fourth column represents the subjects using the object-oriented paradigm. Our hypothesis is that the values for the object-oriented group will be less than the procedural group.

All three primary productivity measures were significantly lower for the object-oriented group. The means favored the object-oriented group for the Edits variable, although not significantly, but not the Syntax Errors variable. We conclude that when subjects are given strong encouragement to reuse, the object-oriented paradigm does promote higher productivity than the procedural paradigm.

Table 5. Strong Encouragement (Procedural vs. Object-Oriented)

	Significant?	p-value	Means: Procedural Strong	Means: O-O Strong
Runs	Yes	0.0043	56.67	38.17
RTE	Yes	0.0035	64.00	37.50
Time	Yes	0.0305	352.00	225.83
Edits	No	0.0854	392.00	234.17
Syn.	No	0.9929	129.00	188.67

6) Does the object-oriented paradigm provide incentives to reuse above those of the procedural paradigm?

As shown in the next chapter, reuse improved productivity over non-reuse for both the procedural and object-oriented paradigms. This question asks whether the *extent* of improvement is comparable for the two language paradigms. Our hypothesis is that the improvement due to reuse will be greater for the subjects using the object-oriented paradigm than those using the procedural paradigm, indicating that the object-oriented paradigm is particularly suited to reuse.

The third column in Table 6 shows for each variable the difference between the mean of the procedural non-reuse group and the mean of the procedural reuse group. This is a measure of the amount of improvement in productivity due to reuse -- the larger the difference, the greater the increase in productivity. The fourth column show comparable mean differences for the object-oriented groups. Therefore, our hypothesis predicts that values in the fourth column should be greater than those in the third column.

On the Runs and RTE variables, the increase in productivity due to reuse was significantly greater for the object-oriented paradigm than for the procedural paradigm. The same pattern occurred on the Time variable, although the difference in means was not large enough to be statistically significant. Once again, contrary to the main productivity variables, the Edits and Syntax Errors variables seem to oppose the hypothesis.

Table 6. Interaction (Extent of Improvement)

			Mean Differences	
			Procedural	O - O
	Significant?	p-value	NR - R	NR - R
Runs	Yes	0.0009	25.31	50.96
RTE	Yes	0.0062	25.54	52.81
Time	No	0.3176	144.52	176.14
Edits	No	0.8753	227.00	183.36
Syn.	No	0.9716	173.86	125.62

Given that two of the three main measures of productivity (Runs and RTE) show significant differences in the hypothesized direction, and that the third main variable (Time) favored the same direction, we conclude that there is a significant difference between the extent of improvement due to reuse across the two language paradigms. In other words, the results show that the object-oriented paradigm demonstrates a particular affinity to the reuse process.

4.3 Summary

This chapter describes the analysis and conclusions for the language paradigm factors. The primary productivity variables were used to answer specific questions based on the productivity of distinct groups or group combinations.

The conclusions formed in this chapter are summarized below:

(1) The object-oriented paradigm substantially improves productivity over the procedural paradigm (question 1),

(2) Language differences are far more important when programmers reuse than when they do not (questions 2 and 3),

(3) Under both moderate and strong encouragement to reuse, the object-oriented paradigm promotes higher productivity than the procedural paradigm (questions 4 and 5), and

(4) The object-oriented paradigm has a particular affinity to the reuse process (question 6).

We did not demonstrate that the object-oriented paradigm promotes productivity when reuse is not a factor. However, the development efforts using either language paradigm were not significantly different. Furthermore, given the reuse potential demonstrated by the object-oriented paradigm, greater benefits can be achieved by using the object-oriented paradigm than by using a procedural approach.

Chapter 5: Managerial Influence Analyses

5.1 Introduction

This chapter discusses the analyses which address the managerial influence issues under investigation. These issues were listed as the second set of experimental questions in Chapter 2.

For questions 7 through 10, tables are given which summarize the analysis performed, as in Chapter 4. Questions 11 and 12 deal with issues that do not lend themselves to straightforward statistical analysis given the available data. These questions are analyzed in a less rigorous, although appropriate, manner.

The questions in this chapter examine the effects of managerial influence on the reuse process. As discussed in Chapter 3, three distinct levels of encouragement to reuse are examined, representing the spectrum of possible managerial support. The three levels are:

1) No reuse allowed
2) Moderate encouragement to reuse at developer's discretion
3) Strong encouragement to reuse anything which remotely applies

Our hypotheses always support the reuse process. That is, assuming fundamental reuse facilities, any level of encouragement to reuse will result in higher productivity than no reuse at all. Furthermore, our hypotheses support moderate encouragement above strong encouragement, given the theory that overzealous reuse can lead to unproductive activities in modifying components that are not applicable to the target system.

5.2 Empirical Results

Again, the experimental questions posed in Chapter 2 are used as a framework for discussion of the statistical analysis. In this chapter, we focus on the set of questions addressing the managerial influence issues. Each question is addressed separately, giving the results of the appropriate analysis.

7) Does reuse promote higher productivity than no reuse?

Given the results in Table 7, the answer to this question is clearly yes. The means in the third column of Table 7 are calculated for all subjects who did not reuse, regardless of the language used. Likewise, the fourth column shows means for all subjects who did reuse, combining the moderate and strong levels of managerial influence.

Our hypothesis is that the means will be lower for the reuse groups, indicating a higher productivity for the subjects who were encouraged to reuse. This hypothesis is strongly supported by all variables.

Table 7. Combined Languages (No Reuse vs. All Reuse)

			Means	
	Significant?	p-value	No Reuse	All Reuse
Runs	Yes	0.0001	78.71	41.14
RTE	Yes	0.0001	83.79	45.04
Time	Yes	0.0001	420.07	255.36
Edits	Yes	0.0001	405.71	198.82
Syn.	Yes	0.0001	302.14	150.92

Table 8. Procedural (No Reuse vs. All Reuse)

			Means	
	Significant?	p-value	Procedural No Reuse	Procedural All Reuse
Runs	Yes	0.0001	75.38	50.07
RTE	Yes	0.0008	81.25	55.71
Time	Yes	0.0047	446.38	301.86
Edits	Yes	0.0001	416.00	189.00
Syn.	Yes	0.0001	311.00	137.14

Table 9. Object-Oriented (No Reuse vs. All Reuse)

			Means	
	Significant?	p-value	O-O No Reuse	O-O All Reuse
Runs	Yes	0.0001	83.17	32.21
RTE	Yes	0.0001	87.17	34.36
Time	Yes	0.0017	385.00	208.86
Edits	Yes	0.0001	392.00	208.64
Syn.	Yes	0.0001	290.33	164.71

This result is further supported by the charts in Tables 8 and 9, which view the data across the reuse factors, but consider each language separately. Table 8 shows the means for the procedural groups with respect to reuse, and Table 9 shows the means for the object-oriented groups with respect to reuse. In both analyses, all variables showed a significant difference in the hypothesized direction.

8) Does management's moderate encouragement to reuse promote higher productivity than no reuse?

Questions 8 through 10 compare the distinct levels of managerial influence without regard to the language used. The means listed in the third column of Table 10 represent the combined groups in the moderate encouragement level of managerial influence. These means are compared to the combined "no reuse" groups, given in column four. Our hypothesis is that the means of the third column will be lower than the means of the fourth column, representing a higher productivity rate for those subjects who were moderately encouraged to reuse.

All five productivity variables supported the hypothesis to a statistically significant degree. We conclude, therefore, that moderate encouragement to reuse promotes higher productivity than no reuse.

Table 10. Combined Languages (Moderate Encouragement vs. No Reuse)

			Means	
	Significant?	p-value	Moderate	No Reuse
Runs	Yes	0.0001	36.44	78.71
RTE	Yes	0.0001	40.75	83.79
Time	Yes	0.0001	230.19	420.07
Edits	Yes	0.0001	190.81	405.71
Syn.	Yes	0.0001	145.00	302.14

9) Does management's strong encouragement to reuse promote higher productivity than no reuse?

This question compares the groups with strong managerial encouragement to reuse, represented by the third column of Table 11, to the groups that did not reuse, whose productivity means are listed in the fourth column. Again favoring reuse over no reuse, our hypothesis is that the strong encouragement means in the third column will be less than the "no reuse" means in the fourth column.

Table 11. Combined Languages (Strong Encouragement vs. No Reuse)

			Means	
	Significant?	p-value	Strong	No Reuse
Runs	Yes	0.0001	47.41	78.71
RTE	Yes	0.0001	50.75	83.79
Time	Yes	0.0037	288.92	420.07
Edits	Yes	0.0001	209.50	405.71
Syn.	Yes	0.0001	158.83	302.14

Once again, all productivity variables support the hypothesis, with all means showing a higher productivity for the groups under strong managerial influence to reuse than the groups that did not reuse.

10) Does management's moderate encouragement to reuse promote higher productivity than management's strong encouragement to reuse?

Table 12 compares the means of the moderate encouragement groups to those of the strong encouragement groups. Because of the possibility of lost time and effort resulting from subject's attempts to reuse components which are not appropriate to the target system, our hypothesis is that the groups with moderate encouragement will have higher productivity than the groups with strong encouragement. That is, our hypothesis is that the means in the third column are less than the means of the fourth column.

Table 12. Combined Languages (Moderate vs. Strong Encouragement)

			Means	
	Significant?	p-value	Moderate	Strong
Runs	Yes	0.0086	36.44	47.41
RTE	No	0.0556	40.75	50.75
Time	No	0.0910	230.19	288.92
Edits	No	0.2144	190.81	209.50
Syn.	No	0.1863	145.00	158.83

The means for the Runs primary variable supported the hypothesis to a significant degree. The p-values for the RTE and Time primary variables were not quite significant, although the means differed greatly in the hypothesized direction. The secondary variables, while differing in the hypothesized direction, were not significant.

Since the differences in means was significant for only one primary variable, we cannot conclude that moderate managerial encouragement promotes higher productivity than strong encouragement. However, the other two primary variables barely missed the significance level, and all five productivity indicators differed in the hypothesized direction. Therefore, the effect of managerial influence certainly warrants further investigation.

11) Does management's strong encouragement to reuse promote improper reuse activities?

The analysis of question 10, while unable to categorically conclude that moderate encouragement promotes higher productivity than strong encouragement, indicates a tendency for less productive results in the strong encouragement category. Question 11 is posed to determine if the cause for lower productivity is inappropriate reuse, as hypothesized.

An analysis of the most commonly reused modules by the moderate and strong groups gives an indication that this hypothesis has merit. The moderate group contained eight subjects and the strong group contained six subjects.

Seven modules were consistently reused by the subjects in the moderate group, and that collection is called the "base set" hereafter. Six subjects used all seven modules, one used five of the seven, and one used only one. The maximum total modules reused by any subject in the moderate group was nine, while the minimum was one. Six distinct modules other than those in the base set of seven were reused, but with little consistency between subjects.

Compare this to the reuse practices in the strong encouragement group. Four subjects used the complete base set, one used six of the seven modules, and one used five of seven. However, the subjects in the strong group reused fifteen distinct modules other than those in the base set. The maximum number of modules reused by subjects in this group was fourteen, while the minimum was six. Once again, little consistency was shown in reusing the modules outside of the base set.

The average percentage of code reused from the base set of modules was 85% by the moderate group, and 81% by the strong group. The average percentage of code reused from all extra modules in the moderate group was 72%, compared to 55% for the strong group.

From question 10, the subjects in the strong group tended to have worse productivity than the moderate group, although not significantly so. Furthermore, there were certain consistencies in the reuse habits of all subjects, and the strong group reused modules that yield less actual code for the target system. Therefore, there is an indication that a cause-effect relationship exists, specifically that the strong encouragement to reuse caused subjects to reuse modules which were more productively written from scratch.

12) Past what point, in terms of the percentage of a component reused for a target system, is it no longer beneficial to reuse?

Given the nature of the data, it is impossible to answer this question with a single percentage value. However, examination of the modules reused other than those in the base set (established in the previous question) yields an indication of the level passed which it is no longer beneficial to reuse.

The percentage in question can be approximated by examining the set of modules used by the strong encouragement group minus the modules in the base set. From question 11, it is this set of modules which caused the lower productivity in the strong group. Excluding the modules in the base set, the maximum percentage reused by subjects in the strong group was 70%, and the minimum reused was 30%. Therefore, the point passed which it is not worth reusing a module is between these limits.

However, this range can be refined. Many subjects reused the base set modules at levels ranging from 50 to 100 percent reuse. Given the higher productivity of the moderate encouragement group, which concentrated on the base set, these reuse practices seem beneficial. The minimum percentage reused of any module in the base set by either reusing group was 50%, indicating that some modules are worth reusing if even only half can be used for the target system.

Therefore, the delimiters for the beneficial cut-off point are between 50 and 30 percent. That is, it is certainly not productive to reuse a module if only 30% can be salvaged. However, some modules should be reused if as little as 50% can be used.

5.3 Summary

This chapter describes the analysis and conclusions for the managerial influence factors. For questions 7 through 10, the primary productivity variables were used to answer specific questions based on the productivity of distinct groups or group combinations. For questions 11 and 12, the subject's reuse activities were examined to determine the cause of lower productivity and to fundamentally quantify it.

The conclusions formed in this chapter are summarized below:

(1) Software reuse promotes higher productivity no matter which language paradigm is used (question 7),

(2) Reuse resulting from both moderate and strong encouragement promote higher productivity than no reuse (questions 8 and 9),

(3) Management's strong encouragement to reuse does tend to promote improper reuse activities (question 11), and

(4) In general, reuse of a module is unproductive if 30% or less is used for the target system, though as much as 50% can be discarded for some modules and still be worth reusing (question 12).

Although we did not demonstrate that management's moderate encouragement to reuse promotes higher productivity than strong encouragement, the results tend to indicate that this phenomenon might exist to some degree. Further research in this area is strongly motivated by this analysis.

Chapter 6: Programmer Attribute Analyses

6.1 Introduction

This chapter discusses the analyses which address the secondary human factors issues of interest, namely programmer cognitive abilities, disposition and experience. These issues were listed as the third set of experimental questions in Chapter 2.

All of the questions in this chapter are addressed by determining if correlations exist between the factors of interest and the productivity variables. The results of question 11 in Chapter 5 demonstrate that high productivity is related to proper reuse practices. Therefore, if a factor of interest correlates with productivity measures, there is an indication that that factor is related to the ability to reuse effectively. No statistical tests are performed on these values due to the nature of the data.

Since the questions deal with the ability to reuse effectively, only the subjects in the four reusing groups were considered. Since distinct factors differentiated the groups in the empirical design, a weighted correlation was calculated (i.e., individual correlation values were determined for each of the four reusing groups, then a combined average was calculated using the number of observations as weights).

Correlation values can range from +1 to -1. A positive correlation indicates that as the value of one variable increases, so does the value of the corresponding variable. A negative correlation indicates that the reverse situation is true; that is, as one increases, the other decreases. A correlation value near zero means no relationship between the two variables exists. For this analysis, a significant relationship is indicated if at least two of the primary variables correlate to a factor at > 0.36 (or < -0.36 for negative correlations), allowing a 5% error possibility.

The first factors of interest in this chapter are the cognitive abilities of individual programmers. Questions 13, 14, and 15 address these issues. As described in Chapter 3, six tests were used to determine the ratings of subjects based on three cognitive abilities (two tests per ability). The cognitive abilities measured in this study were chosen because of their intuitive relationship to the reuse process. The abilities measured were:

• Integration:	the ability of a person to combine objects into a unified whole,
• Perception:	the speed at which a person detects small differences in objects, and
• Visualization:	the ability of a person to visualize changes in objects.

The next area of interest in this chapter is programmer disposition. Question 16 addresses this issue. Questionnaires were used to assess the subject's attitude on several reuse issues. This data generally quantifies the disposition of an individual programmer concerning the reuse process. If these values correlate to the productivity measures, then a subjective influence on the reuse process is indicated.

The last human factor question deals with programmer experience and is addressed in question 17. If an individual simply has more practice at activities which could be related to the reuse process, then a cause-effect relationship might exist. The number of months experience each subject had at several programmer-related activities was recorded and used in the correlation analysis.

6.2 Empirical Results

Once again, the experimental questions posed in Chapter 2 are used as a framework for discussion of the analysis. In this chapter, we focus on the set of questions addressing the secondary human factor issues. Each question is addressed separately, giving the results of the appropriate analysis.

The first three questions in this chapter are addressed by the analysis in Table 13, which shows the relationship between the productivity variables and the scores of the cognitive tests. The column headers are the test designators as taken from the battery of cognitive tests in [EKSR76]. The tests themselves are reprinted in Appendix C.

Table 13. Cognitive Test Correlations

	IP-1	IP-2	P-2	P-3	VZ-1	VZ-2
Runs	0.12	- 0.05	0.21	0.23	- 0.22	- 0.32
RTE	0.12	- 0.26	0.22	- 0.15	- 0.51	- 0.38
Time	0.26	- 0.16	0.09	- 0.13	- 0.61	- 0.22
Edits	0.30	0.19	0.27	0.46	0.01	- 0.12
Syn.	0.21	0.22	0.22	0.49	0.05	- 0.24

13) Does a programmer's inherent ability to combine objects into an integrated whole relate to his ability to reuse effectively?

The first and second columns of Table 13 represent the two tests measuring the integrative processes of the subjects (tests IP-1 and IP-2). Since the scores are correlated against the productivity measures, our hypothesis is that the correlation values will be negative.

All correlation values for the integrative processes tests are within -0.26 to 0.30. Not only are the directions of the correlations inconsistent, but the values are generally close to zero. No relationship between the cognitive ability to integrate objects into a unified whole and the ability to reuse effectively is demonstrated by this data.

14) Does a programmer's inherent ability to quickly detect small differences in objects relate to his ability to reuse effectively?

This question deals with the subject's perception speed. The third and fourth columns of Table 13 show the correlation values for the two tests measuring this ability (tests P-2 and P-3). Our hypothesis is that if any relationship exists, the correlations will be negative, indicating that as perception speed increases, the ability to reuse effectively increases.

Again, as with the previous question, the correlation values are inconsistent and very close to zero. No relationship between the cognitive ability to quickly detect differences in objects and the ability to reuse effectively is demonstrated by this data.

15) Does a programmer's inherent ability to visualize changes in objects relate to his ability to reuse effectively?

The fifth and sixth columns of Table 13 list the correlation values for the tests measuring visualization, or the ability to mentally manipulate objects (tests VZ-1 and VZ-2). Our hypothesis, once again, is that the correlations will be negative.

This relationship is given some support by the data. The fifth column shows two of the main productivity variables, RTE and Time, with fairly large negative correlation values. While the sixth column's values are not as demonstrative, all primary variables for the two tests are consistently negative. Given the nature of the data represented, these results indicate that some relationship exists between the ability to visualize changes in objects and the ability to reuse effectively.

16) Does a programmer's disposition toward reuse relate to his ability to reuse effectively?

As discussed earlier, programmer disposition toward several reuse issues were collected on a four point scale (1=disagree through 4=agree). This data is given in Appendix E. Correlations similar to the previous analysis were attempted using this data. However, for every variable, at least one group showed all subjects with the same level of agreement on the four point scale. Since correlations were calculated per group, and since a constant value cannot correlate to anything, these computations were severely hampered.

Five disposition variables were collected. Four of these variables (reuse of own code, reuse of other's code, trust of other's code, and reuse can save time) were hypothesized to form a negative correlation with the productivity measures. Out of all correlations attempted using these four measures and the main productivity variables (48 total -- 4 disposition variables by 3 productivity variables by 4 reusing groups), 20 were positive, 16 were negative, and 12 could not be calculated due to constant response. The fifth variable (never reuse), was hypothesized to form a positive correlation with productivity measures. For this group of correlations (12 total -- 1 disposition variable by 3 productivity variables by 4 reusing groups), 5 were positive, 4 were negative, and 3 could not be calculated due to constant response.

A meaningful weighted correlation could not be calculated for any of the disposition variables since at least one group per disposition variable gave constant responses. Furthermore, examining the ratio of positive and negative correlations shows that there was no trend favoring one or the other. Therefore, this data cannot support a relationship between programmer disposition and the ability to reuse effectively.

17) Does a programmer's background experience relate to his ability to reuse effectively?

Several variables representing background experience in activities which could relate to the reuse process were measured per subject. These variables were collected separately for academic and professional experience. Several variables, including all variables measured at the professional level, did not show differences between subjects due to lack of experience. Therefore, no correlations were possible for these measures. Table 14 lists the correlation values for the variables which had sufficient merit to analyze.

The workable data recorded the number of months of experience each subject had (1) writing in Pascal, (2) writing in C, (3) creating external documentation for software systems, (4) performing testing activities, and (5) using the Unix operating system. Our hypothesis is that, if a relationship exists between experience and the ability to reuse effectively, the correlation values will be negative. This implies that the more experience a programmer has, the more effective he will be at the activities involved in the reuse process.

Table 14. Background Experience Correlations

	Pascal	C	Doc.	Testing	Unix
Runs	- 0.03	- 0.15	0.03	- 0.31	- 0.26
RTE	- 0.43	- 0.23	0.29	- 0.54	- 0.11
Time	- 0.40	- 0.17	0.37	- 0.59	- 0.01
Edits	0.44	- 0.02	- 0.06	0.10	- 0.24
Syn.	0.47	- 0.20	0.02	0.16	- 0.14

The first and second column show the relationship between writing in a given language and effective reuse. While the primary variables in both columns are consistently negative, the values are not high enough to indicate a relationship. The same conclusion is drawn for the Unix experience variable, as shown in the last column. The documentation variable, shown in the third column, did not even demonstrate a correlation in the hypothesized direction and, like the others, had values too close to zero to indicate a relationship.

Only the amount of experience testing, shown in the fourth column, showed any relationship to effective reuse. All three primary productivity measures formed a negative correlation with the testing variable, and two of them (RTE and Time) had values less than -0.5. These results indicate that a relationship exists between the amount of experience a subject has in testing software products and the ability to reuse effectively.

6.3 Summary

This chapter describes the correlation analysis and conclusions for the secondary human factors. Since these issues were not specifically designed into the experimental framework, rigorous statistical tests were not feasible.

Only two factors seemed to demonstrate a relationship to effective reuse. They are the cognitive ability to visualize changes to objects and the amount of experience performing testing activities. Many factors which were measured did not contain sufficient variability in the data to warrant computation of correlations.

The lack of significant findings in this chapter is probably due to several influences. First, because these factors were not a major focus in the experimental design, the ability to detect the relationships is greatly reduced. Furthermore, the overall reuse process is a complex endeavor, which is why many factors are under investigation. Given many levels and directions of influence, it is unlikely that any single human factor will correlate very highly with reuse success.

Further study is certainly required to assess the full impact of these factors, specifically the two which showed substantial promise. The next chapter discusses future work in this area, and summarizes the conclusions drawn in this experiment.

Chapter 7: Conclusions

7.1 Introduction

This chapter serves to summarize the conclusions drawn in this research. While not all hypotheses under investigation were supported in this study, several of the primary points were demonstrated. The main factors of interest, the object-oriented paradigm and managerial influence, both showed a particular influence in the reuse process.

The individual conclusions were based on the statistical analysis of carefully collected data. Some discussion is given to less rigorous results, but care is taken to differentiate strong conclusions from suggestive ones.

7.2 Summary of Conclusions

A summary of all conclusions reached in this experiment is given below:

(1) The object-oriented paradigm substantially improves productivity over the procedural paradigm (question 1),

(2) Language differences are far more important when programmers reuse than when they do not (questions 2 and 3),

(3) Under both moderate and strong encouragement to reuse, the object-oriented paradigm promotes higher productivity than the procedural paradigm (questions 4 and 5),

(4) The object-oriented paradigm has a particular affinity to the reuse process (question 6).

(5) Software reuse is promotes higher productivity no matter which language paradigm is used (question 7),

(6) Both moderate and strong encouragement to reuse promote higher productivity than no reuse (questions 8 and 9),

(7) Management's strong encouragement to reuse does tend to promote improper reuse activities (question 11), and

(8) In general, reuse of a module is unproductive if 30% or less is used for the target system, though as much as 50% can be discarded for some modules and still be worth reusing (question 12).

Some results, due to lack of rigorous testing, cannot be stated as strong conclusions. For example, some evidence indicates that the cognitive ability to visualize changes to objects and the amount of experience performing testing activities are related to the ability to effectively reuse.

7.3 Future Work

An important facet of the experimental method is that the results are repeatable. Experiments similar to the one described in this dissertation should be conducted to verify the results and clarify any inconclusive situations. In particular, the secondary variables of Edits and Syntax Errors did not always support the analysis of the main variables, even when intuition says that they should have. This tendency deserves further investigation.

Other experiments should be conducted which independently investigate the main elements of this research: software reuse, the object-oriented paradigm and human factors. The factors which affect software reuse are many and varied. Similar experiments can be designed to determine the impact of code characteristics, other language effects, and specific human factors.

This type of experiment should also be conducted in a true commercial setting. While we feel that the similarities between student subjects and professionals make this experiment and others like it worthwhile, completely realistic environmental conditions can only improve the quality of the results. However, given the amount of control necessary to conduct such an experiment, careful attention must be given to the precise experimental design used.

The object-oriented paradigm contains a wealth of possible benefits beyond those of software reuse that have yet to be shown empirically. Claims that associate the object-oriented approach with improved design, less and easier maintenance, and higher reliability when compared to its procedural counterpart demand further investigation.

Experience reports alone are not enough to substantiate the strong assumptions that are associated with many software engineering claims. Experimental research into these areas is necessary to provide a solid base to support the theories that shape state-of-the-art software production.

Bibliography

[BARB87] Barnes, B., Durek, T., Gaffney, J., Pyster, A., "A Framework and Economic Foundation for Software Reuse," Proceedings of the Workshop on Software Reuse, October 1987.

[BIGT87] Biggerstaff, T., Richter, C., "Reusability Framework, Assessment, and Directions," IEEE Software, March 1987, pp. 41-49.

[BOED84] Boehm-Davis, D., Ross, L., "Approaches to Structuring the Software Development Process," International Journal of Man-Machine Systems, (to appear 1991).

[BROF87] Brooks, F.P., "No Silver Bullet: Essence and Accidents of Software Engineering," Computer, April 1987, pp. 10-19.

[BROR80] Brooks, R., "Studying Programmer Behavior Experimentally: The Problems of Proper Methodology," Communications of the ACM, 1980, Volume 23, Number 4, pp. 207-213.

[CARL85] Cardelli, L., Wegner, P., "On Understanding Types, Data Abstraction, and Polymorphism," ACM Computing Surveys, December 1985, pp. 471-522.

[CURB80] Curtis, B., "Measurement and Experimentation in Software Engineering," Proceedings of the IEEE, 1980, Volume 68, Number 9, pp. 1144-1157.

[CURB87] Curtis, B., "Five Paradigms in the Psychology of Programming," MCC Technical Report Number STP-132-87, April 1987.

[CURB89] Curtis, B., "Cognitive Issues in Reusing Software Artifacts," Software Reusability, Volume II, ACM Press, 1989, pp. 269-287.

[EKSR76] Ekstrom, R.B., French, J.W., Harman, H.H., “Manual for Kit of Factor-Referenced Cognitive Tests,” Educational Testing Service, August 1976.

[EVAG89] Evans, G.E., Simkin, M.G., "What Best Predicts Computer Proficiency?," Communications of the ACM, Volume 32, Number 11, 1989, pp. 1322-1327.

[EWEJ81] Ewers, J., Vessey, I., "The Systems Development Dilemma: A Programming Perspective," MIS Quarterly, June 1981, pp. 33-45.

[FISG87] Fischer, G., "Cognitive View of Reuse and Redesign," IEEE Software, July 1987, pp. 60-72.

[FREP83] Freeman, P., "Reusable Software Engineering: Concepts and Research Directions," ITT Proceedings of the Workshop on Reusability in Programming, 1983, pp. 2-16.

[FREP87] Freeman, P., "A Perspective on Reusability," Software Reusability, Computer Society Press of the IEEE, 1987, pp. 2-8.

[GOLA83] Goldberg, A., Robson, D., "Smalltalk-80: The Language and its Implementation," Addison-Wesley, Reading, Mass., 1983.

[HENS90] Henry, S.M., Humphrey, M., Lewis, J.A., "Evaluation of the Maintainability of Object-Oriented Software," Proceedings of the Conference on Computer and Communication Systems, Volume 1, Hong Kong, September 1990, pp. 404-409.

[HORE84] Horowitz, E., Munson, J.B., "An Expansive View of Reusable Software," IEEE Transactions on Software Engineering, September 1984, pp. 477-487.

[JONT84] Jones, T.C., "Reusability in Programming: A Survey of the State of the Art," IEEE Transactions on Software Engineering, September 1984, pp. 488-494.

[KERB84] Kernighan, B.W., "The Unix System and Software Reusability," IEEE Transactions on Software Engineering, September 1984, pp. 513-518.

[LANR84] Lanergan, R.G., Grasso, C.A., "Software Engineering with Reusable Designs and Code, IEEE Transactions on Software Engineering, September 1984, pp. 498-501.

[LENM87] Lenz, M., Schmid, H.A., Wolf, P.F., "Software Reuse through Building Blocks," IEEE Software, July 1987, pp. 34-42.

[LUBM88] Lubars, M.D., "Code Reusability in the Large versus Code Reusability in the Small," Software Reuse: Emerging Technology, Computer Society Press of the IEEE, 1988, pp. 68-76.

[MATY87] Matsumoto, Y., "A Software Factory: An Overall Approach to Software Production," Software Reusability, Computer Society Press of the IEEE, 1987, pp. 155-178.

[MEYB87] Meyer, B., "Reusability: The Case for Object-Oriented Design," IEEE Software, March 1987, pp. 50-64.

[NEIJ84] Neighbors, J.M. "The Draco Approach to Constructing Software from Reusable Components," IEEE Transactions on Software Engineering, September 1984, pp. 564-574.

[PRIR87] Prieto-Diaz, R., Freeman, P., "Classifying Software for Reusability," IEEE Software, January 1987, pp. 6-16.

[STRB88] Stroustrup, B., "What is Object-Oriented Programming?," IEEE Software, July 1988, pp. 10-20.

[TRAW87a] Tracz, W., "Ada Reusability Efforts: A Survey of the State of the Practice," Proceedings of the Fifth Annual Joint Conference on Ada Technology and Washington Ada Symposium, 1987, pp. 35-44.

[TRAW87b] Tracz, W., "Software Reuse: Motivators and Inhibitors," Proceedings of COMPCON S'87, 1987, pp. 358-363.

[TRAW87c] Tracz, W., "Reusability Comes of Age," IEEE Software, July 1987, pp. 6-8.

[TRAW88] Tracz, W., "Software Reuse Myths," ACM SIGSOFT Software Engineering Notes, January 1988, pp. 17-21.

[WEGP83] Wegner, P., "Varieties of Reusability," ITT Proceedings of the Workshop on Reusability in Programming, 1983, pp. 30-44.

[WOOS87] Woodfield, S.N., Embley, D.W., Scott, D.T., "Can Programmers Reuse Software?," IEEE Software, July 1987, pp. 52-59.

Appendix A

Target System Specifications

The following pages contain the specifications for the tasks performed in the experiment. The specifications are divided into the following three sections:

- Introductory material
- Part A: Employee Management
- Part B: Business Management

Specifications

Techtronics Office Automation System

(TOAST)

Introductory Material

Techtronics Corporation

Willmington, Maine

I. Introduction

This document serves as a introduction to the Techtronics Corporation and the Techtronics Office Automation System, or TOAST. The following sections provide a brief description of the company assuming no knowledge of the purpose of Techtronics or of our practices. However, the description is purposely lacking detail in areas that are considered irrelevant to the construction of the TOAST system due to the proprietary nature of the work performed at Techtronics.

The specifications for the TOAST system are divided into two major sections: (A) Employee Management and (B) Widget Management. These specifications are included as separate documents. Included in this document are the specifications of the TOAST system that pertain to both of the major sections of the system. It is very important that the information presented here is well understood before examining any other specifications.

The next section provides a short description of the Techtronics Corporation. Section III discusses the user interface to be maintained throughout TOAST. Finally, Section IV explains security matters in terms of employee access to various functions of TOAST.

II. Background

The Techtronics Corporation was founded by Harold J. Turnbuckle in 1849 as a small manufacturer of the tassels used on graduation caps. From this humble beginning Techtronics has grown to become the largest manufacturer of widgets on the eastern seaboard. Widget use, of course, ranges from the creation of bookends to the accurate construction of the funny things that float on swimming pools to divide the deep end from the shallow end to the internal complexities of nuclear reactors for naval submarines. As a matter of fact, widgets have been known to be used in almost all manufacturing and high technology endeavors, except of course the construction and use of personal computers.

Techtronics employs over 14,000 people and manufactures over 30,000 widgets of varying sizes every day. Both our management offices and manufacturing shop floor are efficiently located in a small converted brownstone in Willmington, Maine. Computer terminals have been recently placed in all offices and across the shop floor and are the basis of the TOAST package being proposed.

TOAST has been envisioned as an verbose software package which can be accessed by any employee at any terminal, who could perform any function the system provides assuming he has the proper clearance. The functions range from payroll management to supply acquisition to widget testing to warehouse storage. Employees are even allowed to take a break and play simple games to relieve stress. We at Techtronics believe a happy employee is a productive employee.

The following section describes the user interface to be used throughout the TOAST system.

III. User Interface

TOAST is designed to be a menu-driven, windowed package at all levels. The hierarchy of menus and positioning of windows is left to the designers discretion but should conform to a few simple guidelines. First, menu information should be kept fairly simple and the number of choices should not exceed 9 per menu. If more choices are needed, this should be reflected in the menu hierarchy. Also, windows not in use should be either be (a) removed if not important to the current application or (b) positioned so that the information can be seen in addition to the current window being used. Windows may overlap if necessary.

Each menu should be contained within its own window. Choices should be made by a selection of the digits 1 through n where n is the number of choices that can be made on that menu. Depending on the choice selected, a new window should appear with either another menu or the application of choice. Only one window should be active at one time.

Non-menu interfaces should be appropriate for the task at hand. Prompts should be clear and precise. The user should always be directed toward what is required of him at any given time.

IV. Access Control

All users should be allowed to attempt to perform any function on the TOAST system. However, each function has associated with it an access level designation. If the employee attempting to use the function does not have the appropriate access level, then that user is not allowed to perform that operation.

See Section V (Common Data) for a description of the employee code which contains the access level designation for a given employee and the operation information to be maintained for each function in the TOAST system.

The access level determines what operations can be performed by which employees. The levels are hierarchical, with level D02 having the highest (most secure) access and A01 having the lowest. If an operation requires a B02 access, then only employees with a B02-D02 access level are allowed to use that function (i.e. employees with A01 through B01 are denied access).

Each operation keeps a log of access attempts for monitoring purposes.

V. Common Data

This section describes data important to both major sections of the TOAST specifications. All data must be maintained in exactly the manner described here.

Employee Code

Each Techtronics employee has an employee code which contains various information. The employee code is a thirteen character string with the following breakdown:

Characters	Information Stored
1-3	Employee level
4-5	Division designation
6-8	Team designation

9-13 Employee code number

The division designation is a two character string indicating in which division the employee is located. The team designation is a three character string indicating the team within the division to which that employee currently belongs. The employee number is a unique five digit string specifying that individual employee.

The first three characters of an employee code indicate the access level at which that employee operates. The employee level can be any of the following:

A01 through A12
B01 through B07
C01 through C04
D01 through D02

For example, an employee code C03MFTST03378 defines an employee with C03 access level, in division MF, group TST with individual id code 03378.

Operation Information

Each operation in the TOAST system has associated with it an operation name, the level which an employee must have to use that operation, and a log storing the date, time and employee id for each attempted access (successful or not). The operation name is a 30 character string, the date/time is a twelve character string, and the employee id is a twelve character string (see Employee Code, above).

The operation level is consistent with the employee level in the employee code and determines what employees can perform that operation (see Section IV).

Specifications

Techtronics Office Automation System

(TOAST)

Part A: Employee Management

Techtronics Corporation

Willmington, Maine

I. Introduction

This document describes the specifications for Part A of the Techtronics Office Automation System (TOAST). In particular, these specifications deal with the internal business management of the Techtronics Corporation. These factors include employee management, payroll, cost center accounting, meeting planners, and even entertainment.

As one of the largest employers on the east coast, the Techtronics Corporation holds in high regard the efficient management of employee information. Currently all data is handled using white 3x5 cards and indelible blue felt markers. The employee code is the fundamental piece of information concerning any process. This code is described in the introductory specification package for TOAST.

Payroll checks are issued every two weeks and involve careful processing. Accurate management of check calculation and tax information is essential.

Meetings are an everyday occurrence in the Techtronics Corporation. TOAST should provide services that will schedule meetings between employees without unnecessary communication overhead.

Each division within Techtronics is sectioned into cost centers. This partitioning governs the internal budgeting for our divisions. Each cost center is given a specific amount of money in their account at the beginning of each fiscal year. It is the responsibility of division management that this money is used effectively throughout the year and that costs are not overrun. All financial expenditures are charged to a cost center and some may require special privilege. One example of cost center use is the company store, which maintains most supplies necessary for office management. Each time an item is "purchased" from the store by charging its price to a particular cost center.

We at Techtronics also put great stock in the happiness of our employees. In that light, TOAST should include specific "games" which will provide essential diversions for the proper relief of stress. We are confident that our dedicated employees will use this opportunity in the manner in which it is intended. Besides, our operation tracking system will allow us to keep tabs on who uses it and we'll fire anyone who abuses our trust without a second thought.

The following section describes in succinct detail the information which must be maintained for the various aspects of the Techtronics Corporation handled in this specification. This data has been used to maintain our processes off line for many years and are considered tried and true. Section III describes the precise functionality that TOAST should perform.

II. Related Data

Employee Data

Employee Code (see introductory specification)
Payroll information (see below)
Number of sick and vacation days allowed per year.
Number of sick and vacation days available.

Payroll Data

Yearly salary.
State and federal tax rate percentages.
Salary paid to date.
State and federal taxes paid to date.

Operation Data

Operation code: a three character string broken down as follows:

Characters	Information Stored
1-2	Operation Category (numeric)
3	Operation indicator (alpha)

The operation code is consistent with this specification. For example, the operation to modify the tax rates has the code 03C.

Operation level - needed by an employee to access that operation.

Operation log - consisting of the following information for each access:

Date/time of access. (see below)

Employee code of person attempting access.

Date/Time Data

Date string: an eleven character string broken down as follows:

Characters	Information Stored
1-2	Last two digits of year (numeric)
3	Dash (-)
4-5	Month (numeric, zero filled)
6	Dash
7-8	Day (numeric, zero filled)
9	Dash
10-11	Hour (numeric, 24 hour clock, zero filled)

For example, the date string 89-09-10-14 represents 2 PM on September 10, 1989. All dates are represented by strings in this form. the day and hour can be left off for a more general date. However, if the hour designator is present, then the day designator must be.

Cost Center Data

Cost center code - a three character string broken down as follows:

Characters	Information Stored
1-2	Operation Category (numeric)
3	Operation indicator (alpha)

Available funds.

Year to date costs.

Charge history containing the following information per charge:

Amount charged.
Date/time.
Employee code of person charging.
Charge type indicator - a single character from the following list:

O, S, T, E, L, or X

Inventory Data

Stock id - a four character string broken down as follows:

Characters	Information Stored
1-2	Item type (one of TD, SY, II, SL, VN)
3-4	Item number (numeric)

Item name. (alphanumeric)
Item price. (numeric)
Current quantity. (numeric)

III. Functionality

General functional categories are listed below. Specific operations are listed for each category. The code in parenthesis () is the level required by an employee to perform the operation.

(1) Employee Management

Maintain an employee list with all appropriate information. All references are made by employee code.

a) Add a new employee. (C01)
b) Remove an employee. (C01)
b) Display basic information about a specific employee. (C01)
c) Modify an employee code for a specific employee. (C02)

d) Charge a sick or vacation day to an employee. (C01)

(2) Operation Access

Monitor the access of operations using the operation log. See the introductory specification material for more information on operation access. All operation references are made by operation code.

a) List all employees who used a particular operation on a given day. (C01)

(3) Payroll

Perform all payroll related operations to employee data.

a) Begin a new fiscal year. (D01)
b) Display financial information about a specific employee. (C02)
c) Update tax rates. (C02)
d) Compute a two-week paycheck and maintain tax information. (C02)

(4) Calendar

Maintain a personal calendar for a given employee.

a) Block a given hour of a given day. (B01)
b) Free a given hour of a given day. (B01)
c) Schedule a meeting with a specific employee for a given day. (B01)

(5) Cost Center Management

Maintain cost center information about the entire corporation. All references to cost centers are made by cost center code.

a) Add a new cost center. (C01)
b) Delete a cost center. (C01)
c) Display basic information about all cost centers. (C02)

d) Display charges by a specific employee within a cost center. (C02)

(6) Company Store

Maintain an inventory system for the company store. All references are made to a stock item by stock id.

a) Create a new stock item. (B05)
b) "Purchase" a stock item. (B05)
c) Display all information about a stock item. (B05)

(7) Entertainment

Create the following simple games for stress management purposes. Explanations of the games are as follows:

HI-LOW: Accept guesses for a random number between 1 and 1000. Return a message stating whether the guess was hi or low. Allow the player to quit at any time. Play ends when the guess is correct.

RIDDLE: Ask the user for the solution to a random riddle. Give the user up to three guesses if necessary before telling him the correct answer.

a) Play the Hi-Low game. (A01)
b) Play the Riddle game. (A01)

Specifications

Techtronics Office Automation System

(TOAST)

Part B: Business Management

Techtronics Corporation

Willmington, Maine

I. Introduction

This document describes the specifications for Part B of the Techtronics Office Automation System (TOAST). In particular, these specifications deal with the aspects of the Techtronics Corporation that make it a profitable business. These factors include widget construction, tracking, defect repairs, storage, quality control and customer management.

As described in the introductory specifications for TOAST, the Techtronics Corporation is the largest manufacturer of widgets on the east coast. This productivity is largely due to our efficient shop floor control system which moves widgets from station to station through the various stages of construction. These stations, and of course the widgets which travel through them, need to be governed in a manner which allows flexibility, enhancement, and which gives firm control over the processes involved.

Quality control processes at Techtronics involve various types of destructive testing and defect repair management. Our high standards have kept us the leader in widget development over many years. The testing processes are highly scientific and need accurate control to ensure safety and quality.

Widget storage is also a major concern of the shop floor. At various points in the widget development life cycle, widgets in intermediate levels of construction are stored in our large warehouse prior to further processing. Also, completed widgets are stored temporarily before shipment across the country and the world in our underrated fleet of VW microbuses.

Customers who receive our widgets come from all kinds of business orientations: government, scientific, financial, hygienic, ect. Quotes on the price considerations of our widgets are requested daily and therefore new customers are obtained constantly. This process of our business aspects also needs careful attention.

The following section describes in succinct detail the information which must be maintained for the various aspects of the Techtronics Corporation handled in this specification. This data has been used to maintain our processes off line for many years and are considered tried and true. Section III describes the precise functionality that TOAST should perform.

II. Related Data

Widget Data

Widget id - a twelve character string broken down as follows:

Characters	Information Stored
1-4	Widget type (BLAG,BLOG,HORT,NNEL,MILE,ZORN)
5	New or repair indicator (N or R)
6-9	Current location
10-12	Widget id number (numeric)

The current location of a widget refers to the station it is currently at. The first character of the location is A-D, followed by a numeric string under 400, zero filled. The id number is unique and identifies that widget specifically.

Creation date of the widget.
Total number of corrosion tests performed to date.
Total number of x-ray tests performed to date.
Number of unsuccessful corrosion tests.
Number of unsuccessful x-ray tests.

Date/Time Data

Date string: an eleven character string broken down as follows:

Characters	Information Stored
1-2	Last two digits of year (numeric)
3	Dash (-)
4-5	Month (numeric, zero filled)
6	Dash
7-8	Day (numeric, zero filled)
9	Dash

10-11	Hour (numeric, 24 hour clock, zero filled)

For example, the date string 89-09-10-14 represents 2 PM on September 10, 1989. All dates are represented by strings in this form. the day and hour can be left off for a more general date. However, if the hour designator is present, then the day designator must be.

<u>Station Data</u>

Bay the station is in. (A, B, or C)
Station number. (numeric under 400)
Station managers name. (alpha)
Station manager's employee code. (see introductory specifications)
Operation description. (alpha)
Components queued at station.

<u>Test Log Data</u>

Widget id tested.
Date tested.
Test number performed.
Success or Failure result. (S or F)
Acid used (hydrochloric or sulfuric - corrosion tests only)

<u>Customer Data</u>

Customer code - a six character string broken down as follows:

Characters	Information Stored
1	Customer type (D, R, G, or X)
2-6	Customer id number (numeric)

Full Address. (street, state, zip code)
Telephone number.
Name of a contact person. (alpha)

III. Functionality

General functional categories are listed below. Specific operations are listed for each category. The code in parenthesis () is the level required by an employee to perform the operation.

(1) Widget Tracking

Control the processing of widgets across the shop floor. All references to widgets are made by the widget id.

a) Add a new widget to the system. (B01)
b) Remove a widget from the system. (B04)
c) Move a widget from one station to another. (B01)
d) Increment the number of successful/unsuccessful corrosion tests. (B02)
e) Increment the number of successful/unsuccessful x-ray tests. (B02)

(2) Station Management

Maintain current information about the configuration of the stations across the shop floor.

a) Create a new station in a certain bay. (C01)
b) Remove a station from a bay. (C01)
c) Change the manager of a station. (C02)
d) Display basic information for all widgets at a given station. (B01)

(3) Corrosion Tests

Perform corrosion testing on widgets and log results. The testing process involves measuring damage done to the outer shells of widgets by various acid types. Two vectors of corrosion data is collected per test, both with the same number of values. For six widget types, there are three different tests, as follows:

Widget BLAG, BLOG:	Vectors: 5 values each	
	Test Fails:	1 value greater than 15
		2 corresponding values sum > 25
Widget HORT, NNEL:	Vectors: 14 values each	
	Test Fails:	1 value greater than 17
		2 corresponding values sum > 30
Widget MILE, ZORN:	Vectors: 22 values each	
	Test Fails:	1 value greater than 20
		2 corresponding values sum > 35

a) Perform corrosion test on a widget of any type. (B04)
b) Display all tests performed on a given date. (B01)
c) Display all tests performed using a certain acid on a given date. (B01)
d) Display all tests performed using a certain test on a given date. (B01)

(4) X-ray Tests

Perform x-ray testing on widgets and log results. The testing process involves measuring internal dimensions of widgets after construction. One vector of x-ray data is collected per test. For six widget types, there are two different tests, as follows:

Widget BLAG, BLOG, MILE:	Vector: 9 values	
	Test Fails:	1 value greater than 15
		2 adjacent values sum > 20
Widget HORT, NNEL, ZORN:	Vector: 10 values	
	Test Fails:	1 value greater than 20
		2 adjacent values sum > 35
		3 adjacent values sum > 45

a) Perform x-ray test on a widget of any type. (B04)
b) Display all tests performed on a given date. (B01)
c) Display all tests performed using a certain test on a given date. (B01)

(5) Warehouse Management

Control the warehouse storage management. Components are stored in bay C, station 399. A maximum of 1000 "units" can be stored in the warehouse, and the following describe how many "units" each widget type takes up: BLAG, BLOG: 10 units; HORT, NNEL: 15 units; MILE: 17 units; ZORN: 25 units.

a) Store a new component. (B02)
b) Return the location of a stored widget. (B02)
c) Remove a component from storage. (B02)

(6) Customer Tracking

Manage the customer identification scheme. All references to a customer are made by the customer code.

a) Add a new customer. (B01)
b) Remove a customer. (B02)
c) Display a customer's information. (B02)

(7) Quote Management

Mange the process of creating price quotes for interested customers. Quote requests are immediately processed and displayed. Use the following chart to compute costs:

Widget Type	Material Cost	Labor Hours	Repair Hours
BLAG, BLOG	22.78	70	20
MILE, HORT, NNEL	88.09	120	25
ZORN	217.69	400	35

Charge 20.18 per labor hour and 22.13 per repair hour. Total labor costs equal the labor hours plus the repair hours per unit. Overhead costs are always 5.50 per widget requested.

a) Process a quote request. (A05)

Appendix B

Data Collection

The following pages list the questions and forms used to collect various disposition, experience, reuse and productivity data during the experiment.

Initial Confidence and Disposition

	Disagree			Agree
I know the Pascal programming language.	1	2	3	4
I know the 'C' programming language.	1	2	3	4
I know the C++ programming language.	1	2	3	4
I know the object-oriented paradigm.	1	2	3	4
I know software engineering concepts in general.	1	2	3	4
I reuse my own software whenever appropriate.	1	2	3	4
I will reuse other people's software if appropriate.	1	2	3	4
I trust other people's code as well as my own.	1	2	3	4
Reuse of code can save time.	1	2	3	4
I never reuse software.	1	2	3	4

Overall QCA: _______________

Major QCA: _______________

Background Experience

Indicate the number of months experience you have had in the following areas:

	Academic and personal	Professional
Programming in:		
Pascal	________	________
C	________	________
C++	________	________
Smalltalk	________	________
Other OO Language	________	________
Requirements generation	________	________
Specification generation	________	________
Integration of new features into someone else's code	________	________
Writing external documentation	________	________
Testing your own source	________	________
Testing someone else's source	________	________
Using Unix	________	________

List all other object-oriented languages that you are familiar with (Smalltalk, Objective C, etc.).

____

List all computer science classes taken at Virginia Tech past CS 3204 (data structures).

____

____

List your favorite or most knowledgeable areas of computer science (i.e. Operating Systems, Database, Compilers, Data Structures, etc.).

____

____

Reuse Questionnaire

Fill out the following information for each module used from the reusable collection:

Developing Subsystem: A) Employee Management B) Business Management

Reusable Module Name: ______________________

Name of routine, class, etc. being reused: ______________________

Part of new system for which this module is being used: ______________________

Answer the following questions as accurately as possible.

Estimate the percentage of source used:

0 10 20 30 40 50 60 70 80 90 100

Categorize the types of changes performed (circle all that apply):

a) Cosmetic (output format, documentation, etc.)

Number of lines: _______ added _______ deleted _______ modified

b) Data definition(s) modified

Number of lines: _______ added _______ deleted _______ modified

c) Algorithm(s) modified

Number of lines: _______ added _______ deleted _______ modified

d) No changes (used code as is)

e) No changes (used concepts only)

Specifically describe the changes you performed:

____

____

____

Time impact of using this module: ____________

Development Questionnaire

System (circle one): a) Employee Management b) Business Management

Module	# Edits	# Lines			"Useless"	Errors	
		Added	Deleted	Modified	Edits	Syntax	Run Time

# Runs	Run Time Errors				Time Impact
	Logical	Format	Inf. Loop	Abend	

Experimental Difficulty

Evaluate the difficulty of the following:

		Easy			Difficult	
1)	Introductory C++ task.	1	2	3	4	
2)	Task A.	1	2	3	4	
3)	Task B.	1	2	3	4	
4)	Development questionnaires.	1	2	3	4	
5)	Reuse questionnaires.	1	2	3	4	NA

Appendix C

Cognitive Ability Tests

The following pages contain the cognitive ability tests used to rank the subjects according to three cognitive skills. Two tests were used per skill, as follows:

- Integrative Processes
  - Test IP-1: Calendar test
  - Test IP-2: Following directions
- Perceptual Speed
  - Test P-2: Number Comparison Test
  - Test P-3: Identical Pictures Test
- Visualization
  - Test VZ-1: Form Board Test
  - Test VZ-2: Paper Folding Test

Name ____

CALENDAR TEST -- IP-1

This is a test of your accuracy in following directions. Each direction will ask you to find a date on a calendar which is printed on the last page of this booklet.

In this calendar you are to remember that:

1. A circled number is a holiday
2. Saturdays and Sundays are weekend days
3. All days except holidays and weekends are work days
4. The first day of Spring is March 21
5. The first day of Summer is June 21
6. The first day of Fall is September 21
7. The first day of Winter is December 21

Look at the sample items below. Put an x on the letter in front of the correct answer.

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	⑪	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

I. What is the third Tuesday of the month?

a. 15th b. 17th c. 22nd d. 24th e. Not given

II. What is the third working day after the holiday?

a. 13th b. 14th c. 15th d. 16th e. Not given

III. What is the seventh working day after the third Monday of the month?

a. 9th b. 27th c. 29th d. 30th e. Not given

The answers are I,a; II,d; III,d.

Your score will be the number of dates you mark correctly <u>minus</u> a fraction of those marked incorrectly. Therefore, it will <u>not</u> be to your advantage to guess unless you have some idea about which date is correct.

This test has two parts. Each part has 10 dates for you to select. You will have <u>7 minutes</u> for each part. When you have finished Part 1, STOP. Please do not go on to Part 2 until asked to do so.

Tear off the last page of this booklet now so you will be able to refer to that calendar easily as you take the rest of this test.

DO NOT TURN THIS PAGE UNTIL ASKED TO DO SO

Part 1 (7 minutes)

Mark an x on the letter in front of the correct answer. Use the calendar that was the last page of this booklet.

1. What is the sixth day after November 29th?

   a. Dec.5 b. Dec.6 c. Dec.10 d. Dec.11 e. Not given

2. What month has the only Tuesday the 12th?

   a. April b. June c. Sept. d. Nov. e. Not given

3. What is the fourth working day after the third weekend in the month having the shortest name?

   a. 1st b. 4th c. 17th d. 24th e. Not given

4. In the first month having 30 days, what is the first working day after the 8th of the month?

   a. Jan. 9 b. March 9 c. April 9 d. June 11 e. Not given

5. The Sunday after the first holiday that falls on a Thursday is:

   a. Feb. 25 b. June 17 c. Aug.19 d. Nov. 25 e. Not given

6. What is the sixth working day in the month which has a holiday that falls on a Friday?

   a. 8th b. 9th c. 10th d. 19th e. Not given

7. What is the eighth working day of the first month in which the 5th of the month falls on a weekend and in which there is only one holiday?

   a. May 8 b. May 10 c. Aug.8 d. Aug.10 e. Not given

8. In the month whose 12th is on a Tuesday, what is the fifteenth working day?

   a. 22nd b. 23rd c. 24th d. 27th e. Not given

9. What is the third Wednesday in the month that follows the second month having exactly 9 weekend days?

   a. 15th b. 16th c. 17th d. 18th e. Not given

10. What is the last weekend day of the last month of the year for which the 5th is on a Tuesday or Thursday and during which there is no change of seasons?

    a. 28th b. 29th c. 30th d. 31st e. Not given

DO NOT GO ON TO THE NEXT PAGE UNTIL ASKED TO DO SO

STOP.

Part 2 (7 minutes)

Mark an x on the letter in front of the correct answer. Use the calendar that was the last page of this booklet.

11. What is the first Wednesday of the 5th month of the year?

    a. Jan. 3 b. Jan. 5 c. May 2 d. May 5 e. Not given

12. What is the first working day following the third holiday of the year?

    a. Feb. 15 b. Feb. 16 c. Feb. 20 d. Feb.22 e. Not given

13. What is the second weekend day in the first month which does not have an "R" in its name?

    a. 2nd b. 3rd c. 4th d. 5th e. Not given

14. What is the first Tuesday of summer?

    a. June 19 b. June 21 c. June 26 d. July 3 e. Not given

15. What is the fifth day following the 3rd of the month before the third change of seasons?

    a. 8th b. 10th c. 11th d. 12th e. Not given

16. One month both begins and ends on a weekend. In that month, on what day of the week is the first day that contains a 3 but does not fall on a Monday or a Thursday?

    a. Tue. b. Wed. c. Fri. d. Sat. e. Not given

17. What is the second summer Friday that is not among the first four days of the month?

    a. 6th b. 10th c. 13th d. 20th e. Not given

18. What is the first Thursday in the second month after the last month which has a "U" in its name?

    a. 4th b. 6th c. 8th d. 11th e. Not given

19. In the month latest in the year that both starts and ends on a working day, what is the first working day that occurs after the 15th of the month and that is not on the same day of the week that begins or ends the month?

    a. 16th b. 17th c. 18th d. 19th e. Not given

20. What day follows the first nine working days in the month containing the third change of seasons?

    a. 10th b. 11th c. 15th d. 16th e. Not given

DO NOT GO BACK TO PART 1 AND
DO NOT GO ON TO ANY OTHER TEST UNTIL ASKED TO DO SO

STOP.

1. A circled number is a holiday.
2. Saturdays and Sundays are weekend days.
3. Other days except holidays are working days.
4. The first day of Spring is March 21.
5. The first day of Summer is June 21.
6. The first day of Fall is September 21.
7. The first day of Winter is December 21.

1973 JANUARY 1973

S	M	T	W	T	F	S
	[✓ 1]	2	3	4	5	6
7	8	9	10	11	12	13
14	[✓ 15]	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

1973 JULY 1973

S	M	T	W	T	F	S
1	2	3	[✓ 4]	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

1973 FEBRUARY 1973

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	[✓ 12]	13	[✓ 14]	15	16	17
18	[✓ 19]	20	21	[✓ 22]	23	24
25	26	27	28			

1973 AUGUST 1973

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	[✓ 16]	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

1973 MARCH 1973

S	M	T	W	T	F	S
				1	2	3
4	5	6	[✓ 7]	8	9	10
11	12	13	14	15	16	[✓ 17]
18	19	20	21	22	23	24
25	26	27	28	29	30	31

1973 SEPTEMBER 1973

S	M	T	W	T	F	S
						1
2	[✓ 3]	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23/30	24	25	26	27	28	29

1973 APRIL 1973

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
[✓ 15]	16	[✓ 17]	18	19	[✓ 20]	21
[✓ 22]	23	24	25	26	27	28
29	30					

1973 OCTOBER 1973

S	M	T	W	T	F	S
	1	2	3	4	5	[✓ 6]
7	[✓ 8]	9	10	11	12	13
14	15	16	17	18	19	20
21	[✓ 22]	23	24	25	26	27
28	29	30	31			

1973 MAY 1973

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	[✓ 28]	29	30	31		

1973 NOVEMBER 1973

S	M	T	W	T	F	S
				1	2	3
4	5	[✓ 6]	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	[✓ 22]	23	24
25	26	27	28	29	30	

1973 JUNE 1973

S	M	T	W	T	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	[✓ 14]	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

1973 DECEMBER 1973

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23/30	24/31	[✓ 25]	26	27	28	29

Name ____

FOLLOWING DIRECTIONS — IP-2

This is a test of your ability to follow a set of directions. You will be given a pattern of letters to look at and will be asked questions about how certain directions will change that pattern. The answer to each question will be one of the letters in the pattern. You are to decide which letter is correct and put an x on that letter.

Look at this example:

			Column		
	1	2	3	4	5
Row 1	A	B	C	D	E
Row 2	E	A	B	C	D
Row 3	D	E	A	B	C
Row 4	C	D	E	A	B
Row 5	B	C	D	E	A

I. Which is the only letter that appears directly above the letter A?

A B C D E

II. If one letter occurs more frequently than another, the answer is the most frequently occuring letter; if no letter occurs most frequently, the answer is the letter in the upper left to lower right diagonal.

A B C D E

You should have marked B for I and A for II.

Your score on this test will be the number of letters which you mark correctly minus a fraction of those incorrect. Therefore it will <u>not</u> be to your advantage to guess unless you are fairly certain of your answer.

This test has two parts. Each part has 10 items on one page. You will have <u>7 minutes</u> for each part. When you have finished Part 1, STOP. Please do not go on to Part 2 until you are asked to do so.

DO NOT TURN THIS PAGE UNTIL ASKED TO DO SO

Part 1 (7 minutes)

All of the directions and questions in Part I will be asked about the pattern of letters below: Mark an x on the correct letter for each question.

	Column 1	2	3	4	5
Row 1	A	B	C	D	E
Row 2	B	D	E	A	C
Row 3	C	E	D	A	B
Row 4	B	A	C	E	D
Row 5	A	C	E	B	D

1. In row 2, what letter appears between D and A?

   A B C D E

2. What letter comes between E and A in Row 3?

   A B C D E

3. What letter would come above the C in Row 5 if the letters in Row 4 were reversed?

   A B C D E

4. The answer to this question is the letter immediately to the right of the middle letter of Row 4.

   A B C D E

5. What is the letter that appears just to the right of the letter that is immediately over the one just to the left of the A in Row 4?

   A B C D E

6. If E is closer to the left end of Row 2 than B, the answer is B; if not, the answer is the letter most frequently coming immediately to the right of A in the pattern.

   A B C D E

7. Suppose Rows 1, 3, and 5 were all written backwards, while Rows 2 and 4 remain as they are. What letter will then occur two times in Column 4?

   A B C D E

8. Start in the lower left hand corner and follow the letters up Column 1, down Column 2, up Column 3, and so on until you reach the upper right hand corner. What is the first letter to appear four times?

   A B C D E

9. Start in the upper left hand corner of the pattern and follow the letters around the outside of the pattern in a clockwise (to the right) direction. What is the second letter following the letter between the second C and the second D you come to?

   A B C D E

10. If A appears in one diagonal (lower left to upper right) more often than D appears in the other diagonal (upper left to lower right), the answer is A; if not, the answer is the letter that does not appear in either diagonal which comes latest in the alphabet.

   A B C D E

DO NOT GO ON TO THE NEXT PAGE UNTIL ASKED TO DO SO.

STOP

Part 2 (7 minutes)

All of the directions and questions in Part 2 will be asked about the pattern of letters below: Mark an x on the correct letter for each question.

	Column 1	2	3	4	5
Row 1	E	D	C	B	A
Row 2	C	A	E	D	B
Row 3	B	A	D	E	C
Row 4	D	E	C	A	B
Row 5	D	B	E	C	A

11. What letter is just above the E in Row 4?
A B C D E

12. Suppose Row 2 were written backwards. What letter would come under B in Row 1?
A B C D E

13. In Row 3, if B appears anywhere to the right of C, the answer is the first letter of Row 1; if not, the answer is the last letter of Row 2.
A B C D E

14. There are two letters that never appear at the right end of a row. The answer is the one that comes later in the alphabet.
A B C D E

15. If A is ever immediately under B, the answer is the letter in Row 1 which that A is under, if not, the answer is the letter in Row 4 which has A over it.
A B C D E

16. Suppose Rows 2 and 4 were written backwards; while Rows 1, 3, and 5 remain as they are. What letter will occur only once in Column 2?
A B C D E

17. The answer to this problem is the letter in Row 4 which is over the letter of Row 5 that never appears at the left end of a row.
A B C D E

18. Start in the upper left hand corner. Read down Column 1, then read down Column 2, and so on. If the first A you come to is followed immediately by an E, the answer is the letter in front of the next D. However, if the first A is not followed immediately by an E, the answer is the letter after whatever one does follow the first A.
A B C D E

19. Start in the upper right hand corner and follow the letters around the outside of the pattern in a counterclockwise (to the left) direction. What is the second letter following the letter between the third B and the third C that you come to?
A B C D E

20. If you draw a line through the two diagonals (upper left to lower right and upper right to lower left) and erase those letters, you will have four triangles of four letters each remaining. If none of these four triangles has two letters alike in them, the answer is the letter that appears least often in the triangles; if not, the answer is the letter that appears where the two diagonals cross.
A B C D E

DO NOT GO BACK TO PART 1 AND
DO NOT GO ON TO ANY OTHER TEST UNTIL ASKED TO DO SO

STOP

Name ______________________________

NUMBER COMPARISON TEST — P-2

This is a test to find out how quickly you can compare two numbers and decide whether or not they are the same. If the numbers are the same, go on to the next pair, making no mark on the page. If the numbers are not the same, put an X on the line between them. Several examples are given below with the first few marked correctly. Practice for speed on the others.

659	____	659	7343801	____	7343801
73845	X	73855	18824	____	18824
1624	____	1624	705216831	____	795216831
435	X	436	971	____	971
4821459	____	4814259	446014721	____	446014721
658351	____	656351	5173869	____	5172869
11655	____	11652	6430017	____	6430017
617439423	____	617439428	512196045	____	512168045
1560459	____	1660459	55179	____	55097
90776105	____	90716105	65216067	____	65216057

Your score will be the number marked correctly minus the number marked incorrectly. Therefore, it will not be to your advantage to guess unless you have some idea whether or not the numbers are the same.

You will have 1 1/2 minutes for each of the two parts of this test. Each part has one page. When you have finished Part 1, STOP. Please do not go on to Part 2 until you are asked to do so.

DO NOT TURN THIS PAGE UNTIL ASKED TO DO SO.

Part 1 (1 1/2 minutes)

Make an X on the line between the numbers that are not the same.

639	_____	639	414982	_____	415982
4714306	_____	4715306	60971	_____	60971
65382	_____	65372	16253948	_____	16253948
710	_____	710	42018591760	_____	43018591760
43210573	_____	43210573	647107569	_____	647107569
6182653905221	_____	6182653905221	721532992551	_____	721582992551
43270105338	_____	43276105338	341798301	_____	341798701
27109816845	_____	27109816855	80537051248	_____	80537051248
519605	_____	519605	5911306581-91	_____	5911306581-91
923-52170687	_____	923452170687	8361-081	_____	3361-081
3705-3141	_____	3105-3141	49-71307	_____	47-71307
2570665292	_____	2570665292	6082649875	_____	6082647375
32016591670	_____	32018691670	5930582136	_____	5730582136
5472075695	_____	5472075683	25603179-137	_____	25603129-137
621532992551	_____	621582992551	805731195	_____	805131195
24179830	_____	24179830	48210-35512	_____	48210435612
70537051248	_____	70537057248	4051768-1309	_____	4051768-1309
7361408	_____	7361708	801-57-9786	_____	801-57-9796
39471307	_____	39471507	53210573	_____	53210573
508254987505	_____	508264957505	718265390521	_____	718265390521
4930582136	_____	4930582136	5527010538	_____	5527010573
136031794157	_____	136031794157	571098168-3	_____	571898166-5
705731195	_____	705736195	619605	_____	617605
38210435512	_____	38210535512	123-52170687	_____	123-52151687

DO NOT GO ON TO THE NEXT PAGE UNTIL ASKED TO DO SO.

<u>STOP</u>.

Part 2 (1 1/2 minutes)

Make an X on the line between the numbers that are not the same.

7573	____	7573	289414	____	289414
347520	____	349520	17906	____	17906
4951	____	4951	16719581024	____	16719581024
4573043	____	4571043	16719581024	____	16719561024
37501243	____	37501243	3965701746	____	3665701746
125093562816	____	125093562816	135299235127	____	135299235127
8350107234	____	8350107234	13697143	____	13897145
54861590172	____	5486170172	84215073508	____	84216073508
506915	____	596915	941356031195	____	941356431195
786071254329	____	786071255329	8041638	____	8041458
41345073	____	41345073	70317494	____	70317494
925660752	____	925660752	35789462806	____	35789562806
16719581023	____	16717581023	6312350395	____	6312350795
3965701745	____	3965701745	731497130632	____	731497130632
135299235126	____	135399235130	591137508	____	591167508
13897142	____	13897142	21553401234	____	21553401284
84215073506	____	84215073507	1251373807	____	1251373507
941356031194	____	941346031194	903148671504	____	903148871504
8041637	____	8071637	68794355108	____	68754354108
70317495	____	70317495	37501255	____	37501255
35789462805	____	35789462805	125093562817	____	125093562817
6312350394	____	6312350394	8350107255	____	8350107255
731497130631	____	731497130681	54861590173	____	5486140173
591137507	____	591127507	506916	____	506916

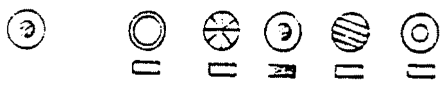
DO NOT GO BACK TO PART 1 AND DO NOT GO ON
TO ANY OTHER TEST UNTIL ASKED TO DO SO.

STOP

Name ______________________________

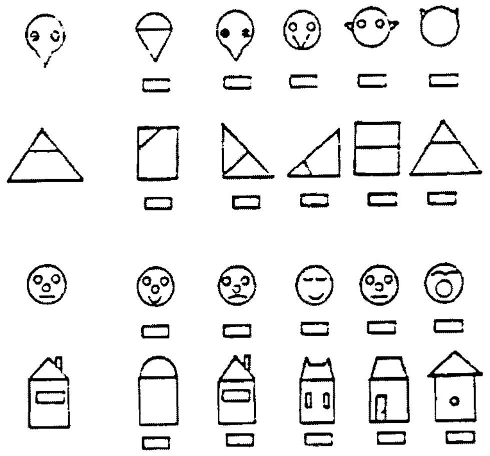
IDENTICAL PICTURES TEST -- P-3

How fast can you match a given object? This is a test of your ability to pick the correct object quickly. At the left of each row is an object. To the right are five test objects, one of which matches the object at the left. Look at the example below:



The third test object has been marked by blackening the space below it, because it is the same as the object at the left.

Now practice on the problems below. Mark them as fast as you can:

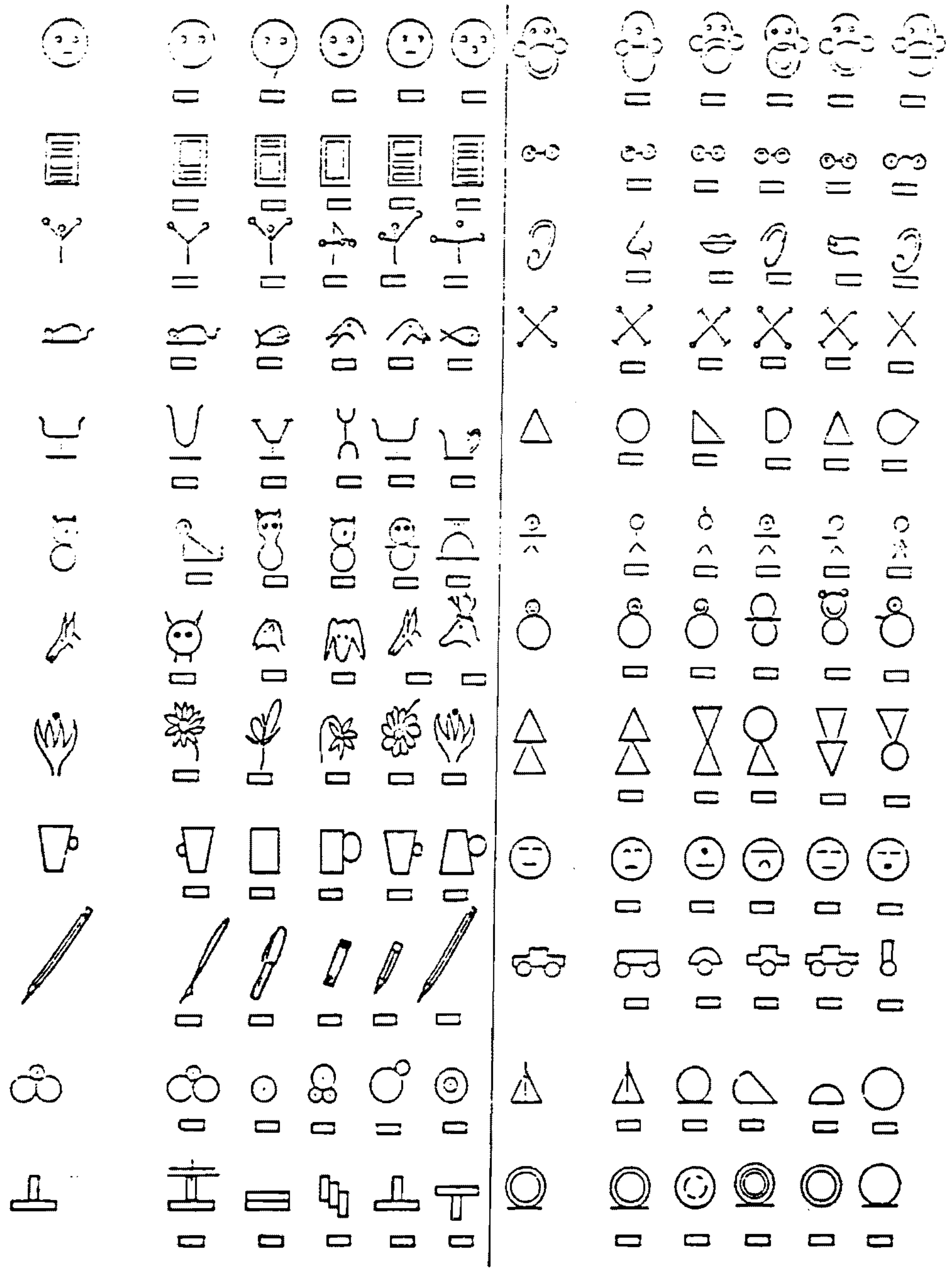


Your score on this test will be the number of objects marked correctly minus a fraction of the number marked incorrectly. Work as quickly as you can without sacrificing accuracy.

You will have <u>1 1/2 minutes</u> for each of the two parts of this test. Each part has two pages. Be sure to do both pages if you have time. When you have finished Part 1, STOP. Please do not go on to Part 2 until you are asked to do so.

DO NOT TURN THIS PAGE UNTIL ASKED TO DO SO.

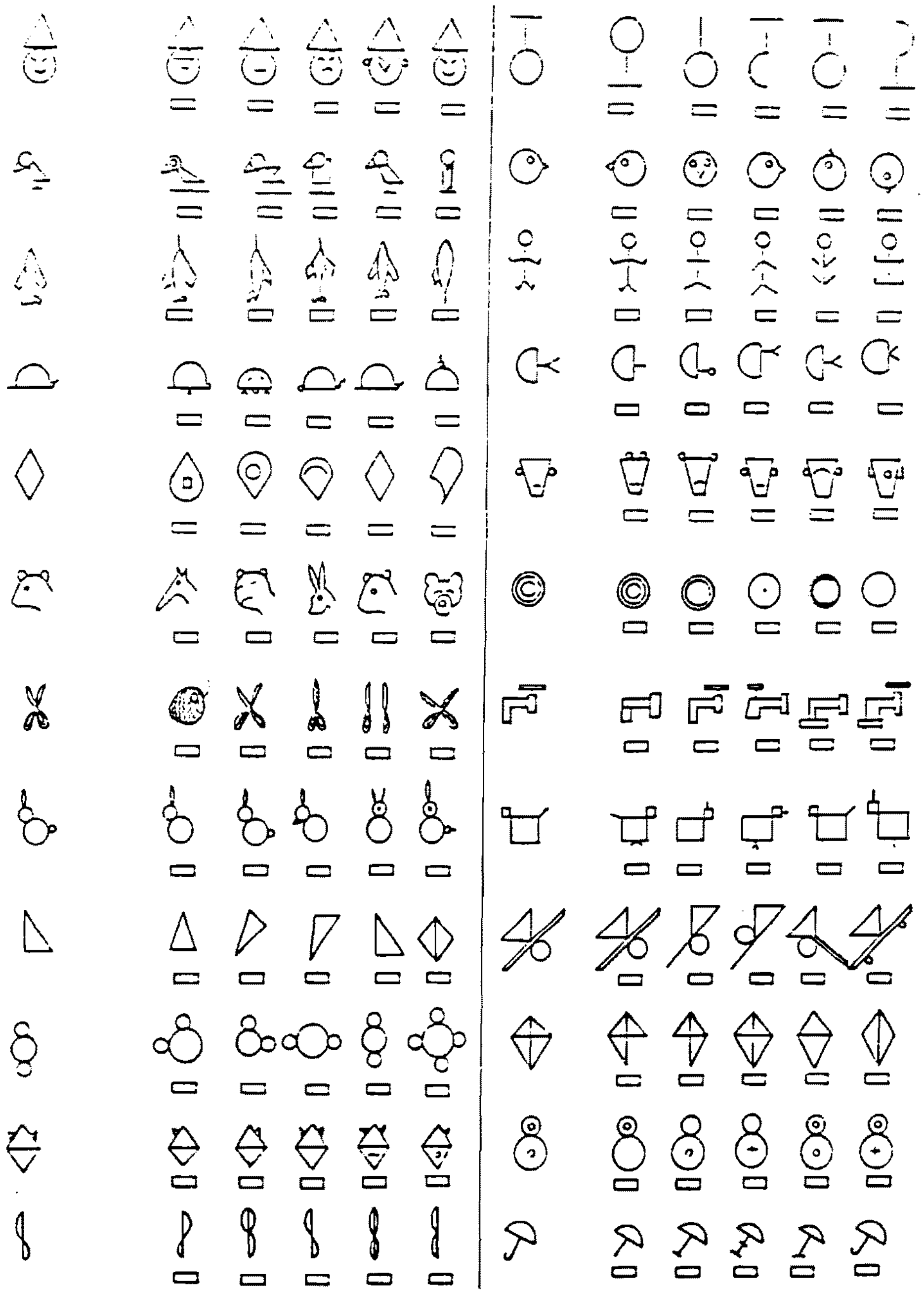
Part 1 (1 1/2 minutes)



GO TO THE NEXT COLUMN.

GO ON TO THE NEXT PAGE.

Part 1 (continued)



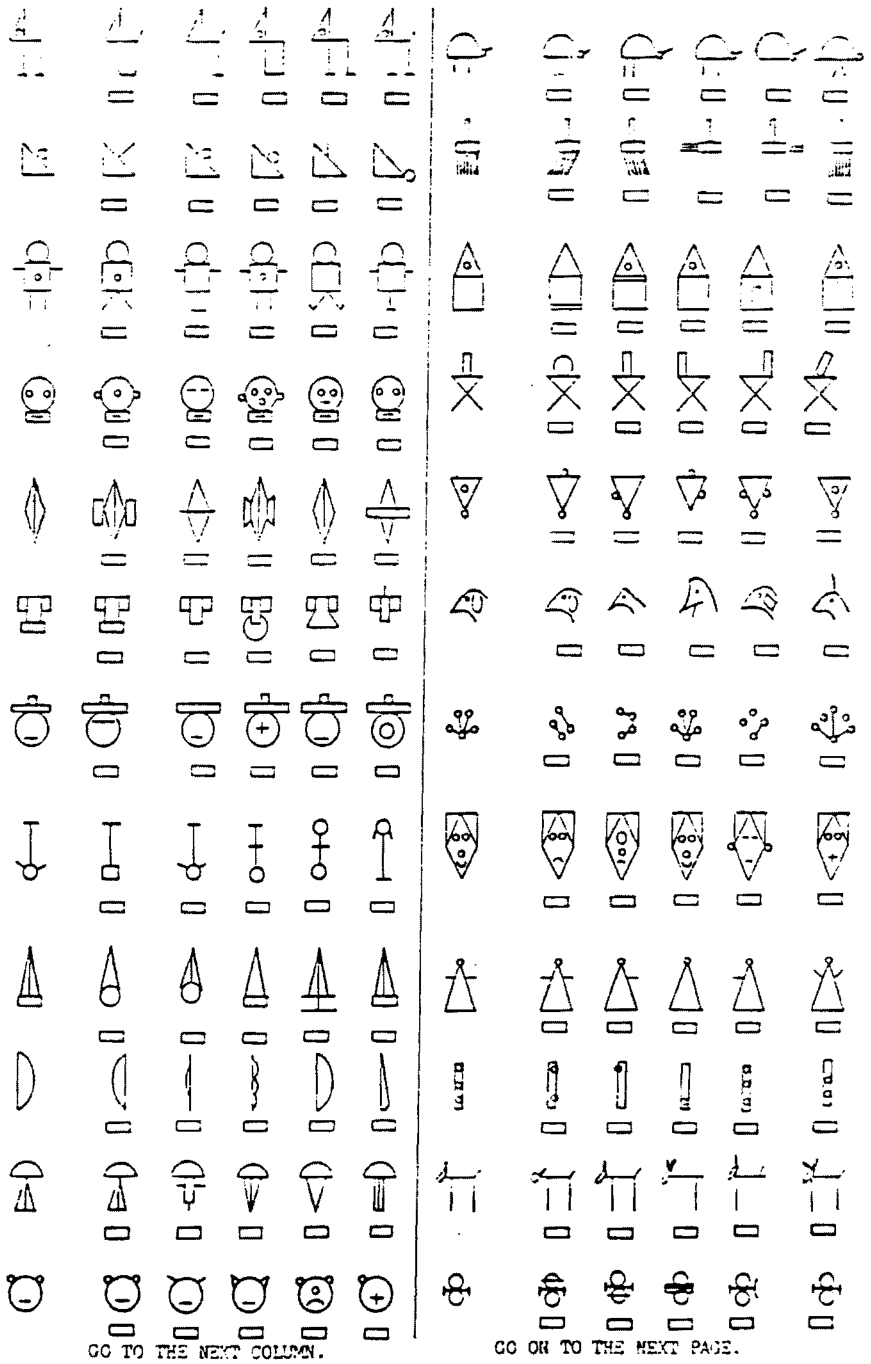
GO ON TO THE NEXT COLUMN.

STOP HERE.

DO NOT TURN THIS PAGE UNTIL ASKED TO DO SO.

STOP.

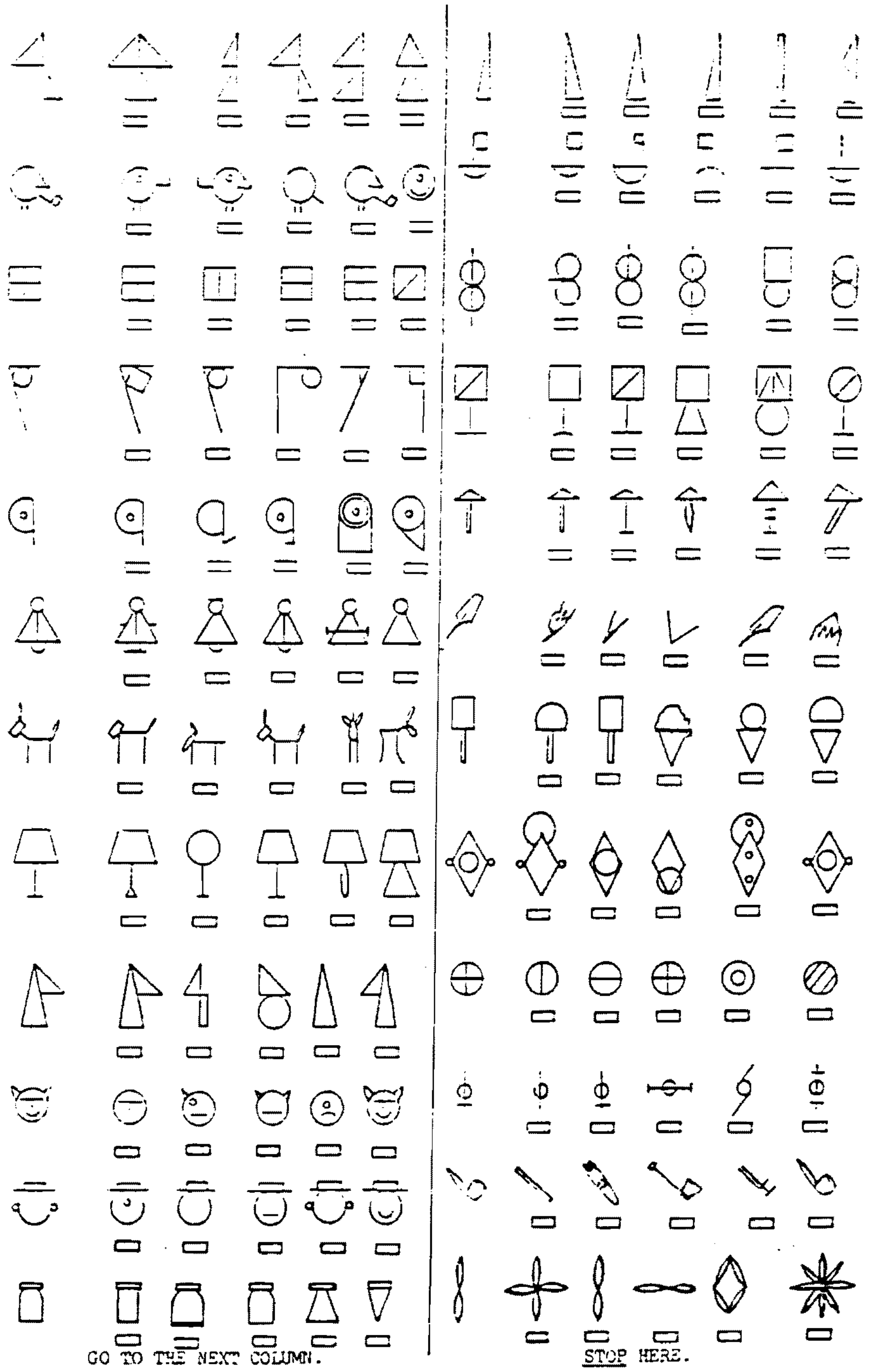
Part 2 (1 1/2 minutes)



GO TO THE NEXT COLUMN.

GO ON TO THE NEXT PAGE.

Part 2 (continued)



GO TO THE NEXT COLUMN. STOP HERE.

DO NOT GO BACK TO PART 1 AND

DO NOT GO ON TO ANY OTHER TEST UNTIL ASKED TO DO SO. STOP.

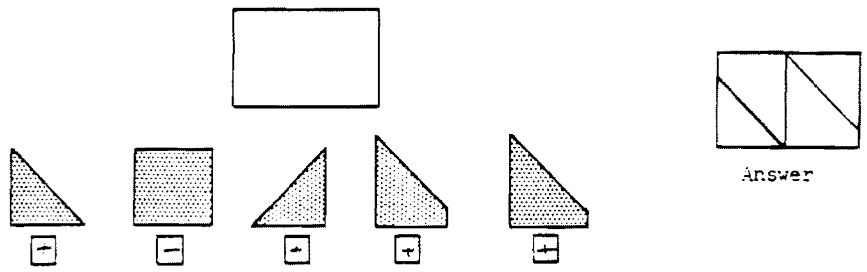
Name ____

FORM BOARD TEST — VZ-1

This is a test of your ability to tell what pieces can be put together to make a certain figure. Each test page is divided into two columns. At the top of each column is a geometrical figure. Beneath each figure are several problems. Each problem consists of a row of five shaded pieces. Your task is to decide which of the five shaded pieces will make the complete figure when put together. Any number of shaded pieces, from two to five, may be used to make the complete figure. Each piece may be turned around to any position but it <u>cannot be turned over</u>. It may help you to sketch the way the pieces fit together. You may use any blank space for doing this. When you know which pieces make the complete figure, mark a <u>plus</u> (+) in the box under ones that are used and a <u>minus</u> (-) in the box under ones that are not used.

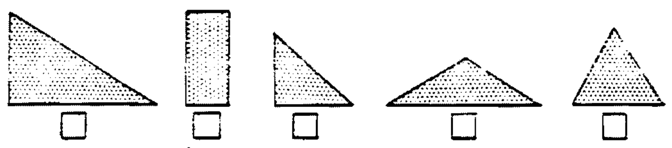
In Example A, below, the rectangle can be made from the first, third, fourth, and fifth pieces. A <u>plus</u> has been marked in the box under these places. The second piece is not needed to make the rectangle. A <u>minus</u> has been marked in the box under it. The rectangle drawn to the right of the problem shows one way in which the four pieces could be put together.

A.

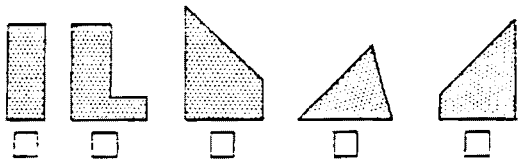


Now try to decide which pieces in Examples B and C will make the rectangle.

B.



C.



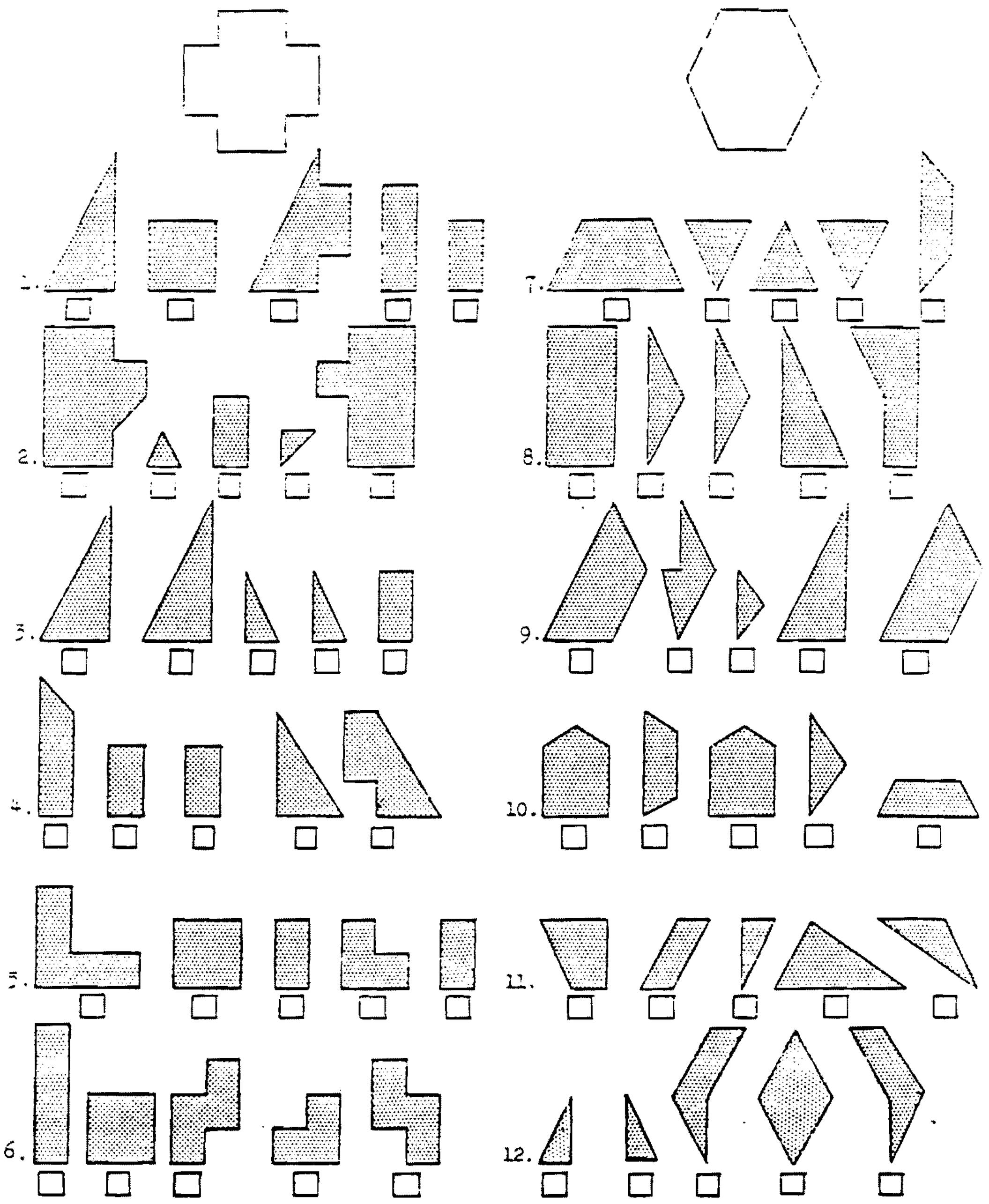
In Example B, the first, fourth, and fifth pieces are needed. You should have marked a <u>plus</u> under these three pieces and a <u>minus</u> under the other two pieces. In Example C, the second, third, and fifth pieces should be marked with a <u>plus</u> and the first and fourth with a <u>minus</u>.

Your score on this test will be the number marked correctly minus the number marked incorrectly. Therefore, it will <u>not</u> be to your advantage to guess unless you have some idea whether or not the piece is correct.

You will have <u>8 minutes</u> for each of the two parts of this test. Each part has 2 pages. When you have finished Part 1 (pages 2 and 3), STOP. Please do not go on to Part 2 until you are asked to do so.

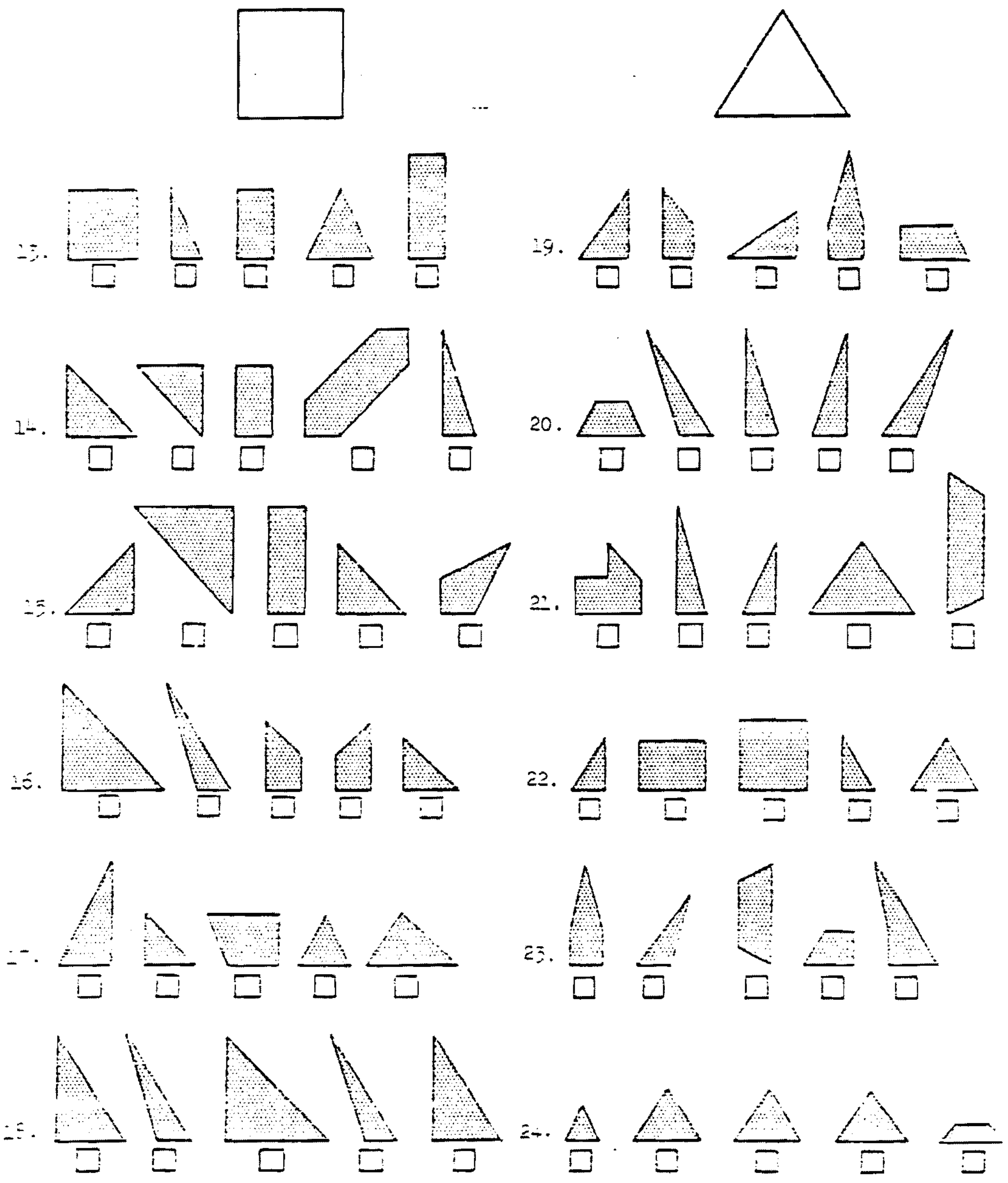
DO NOT TURN THIS PAGE UNTIL ASKED TO DO SO.

Part 1 (8 minutes)



GO ON TO THE NEXT PAGE.

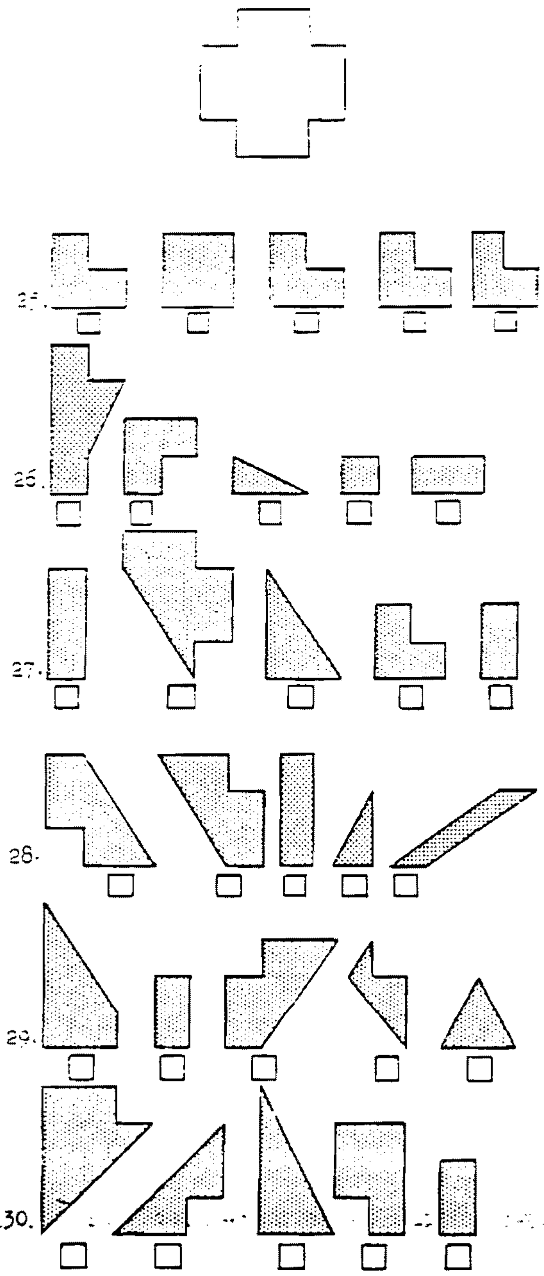
Part 1 (continued)

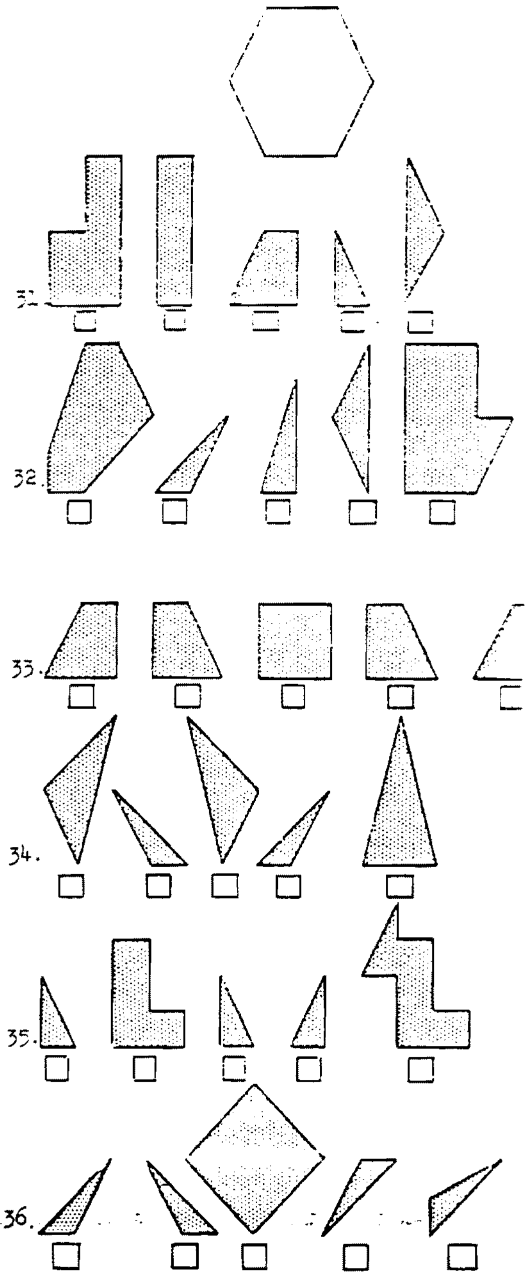


DO NOT TURN THIS PAGE UNTIL ASKED TO DO SO.

STOP.

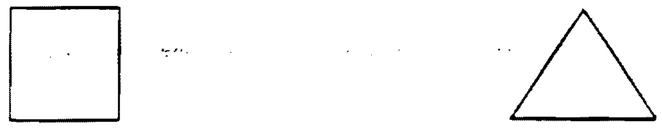
Part 2 (8 minutes)

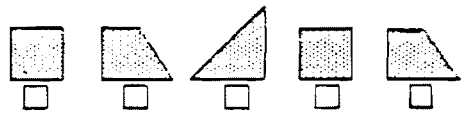


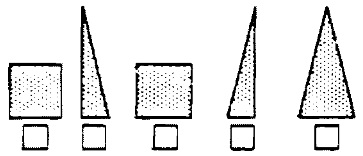


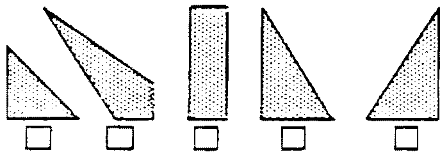
GO ON TO THE NEXT PAGE.

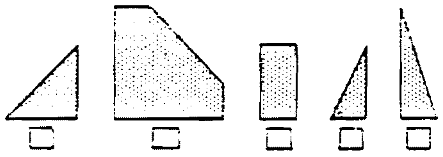
Part 2 (continued)

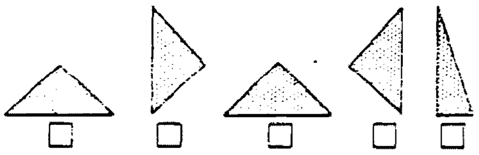


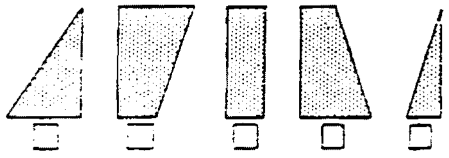
37. 

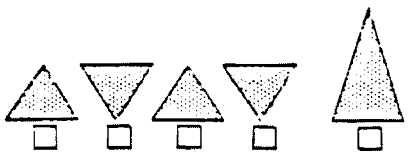
38. 

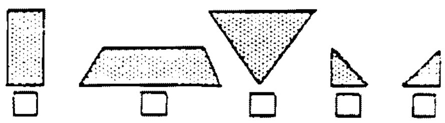
39. 

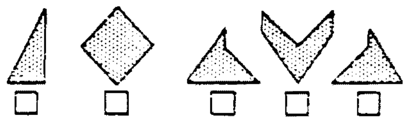
40. 

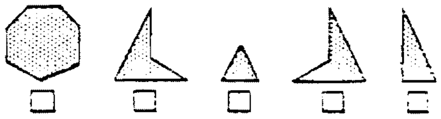
41. 

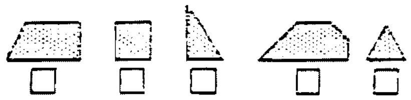
42. 

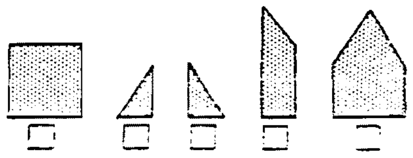
43. 

44. 

45. 

46. 

47. 

48. 

DO NOT GO BACK TO PART 1, AND

DO NOT GO ON TO ANY OTHER TEST UNTIL ASKED TO DO SO.

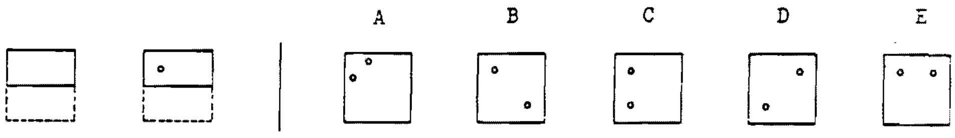
STOP.

Name ____

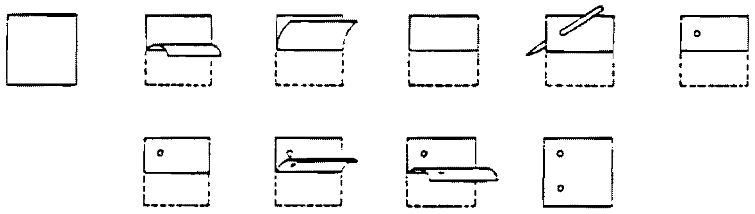
PAPER FOLDING TEST — VZ-2

In this test you are to imagine the folding and unfolding of pieces of paper. In each problem in the test there are some figures drawn at the left of a vertical line and there are others drawn at the right of the line. The figures at the left represent a square piece of paper being folded, and the last of these figures has one or two small circles drawn on it to show where the paper has been punched. Each hole is punched through all the thicknesses of paper at that point. One of the five figures at the right of the vertical line shows where the holes will be when the paper is completely unfolded. You are to decide which one of these figures is correct and draw an X through that figure.

Now try the sample problem below. (In this problem only one hole was punched in the folded paper.)



The correct answer to the sample problem above is C and so it should have been marked with an X. The figures below show how the paper was folded and why C is the correct answer.



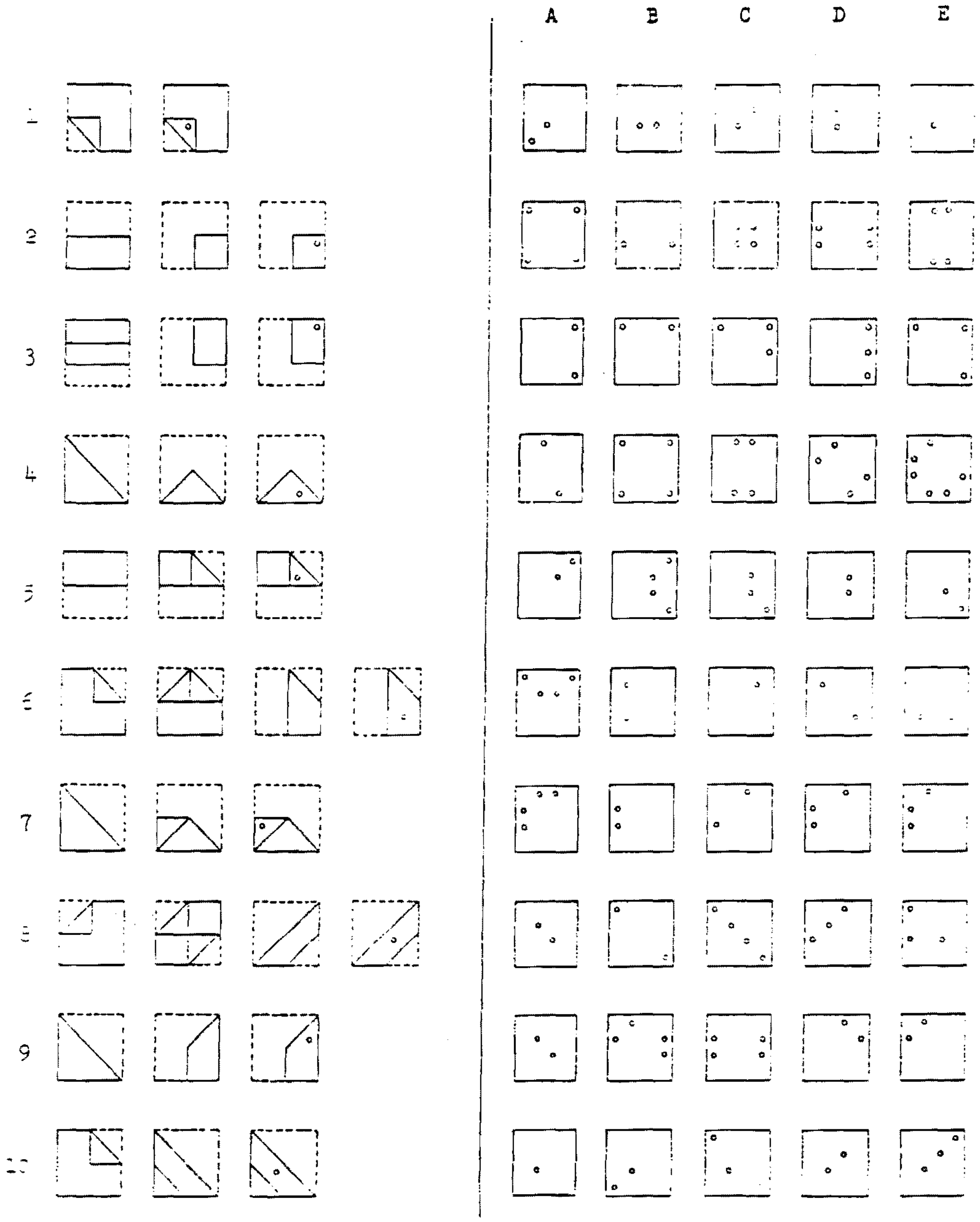
In these problems all of the folds that are made are shown in the figures at the left of the line, and the paper is not turned or moved in any way except to make the folds shown in the figures. Remember, the answer is the figure that shows the positions of the holes when the paper is completely unfolded.

Your score on this test will be the number marked correctly minus a fraction of the number marked incorrectly. Therefore, it will <u>not</u> be to your advantage to guess unless you are able to eliminate one or more of the answer choices as wrong.

You will have <u>3 minutes</u> for each of the two parts of this test. Each part has 1 page. When you have finished Part 1, <u>STOP</u>. Please do not go on to Part 2 until you are asked to do so.

DO NOT TURN THIS PAGE UNTIL ASKED TO DO SO.

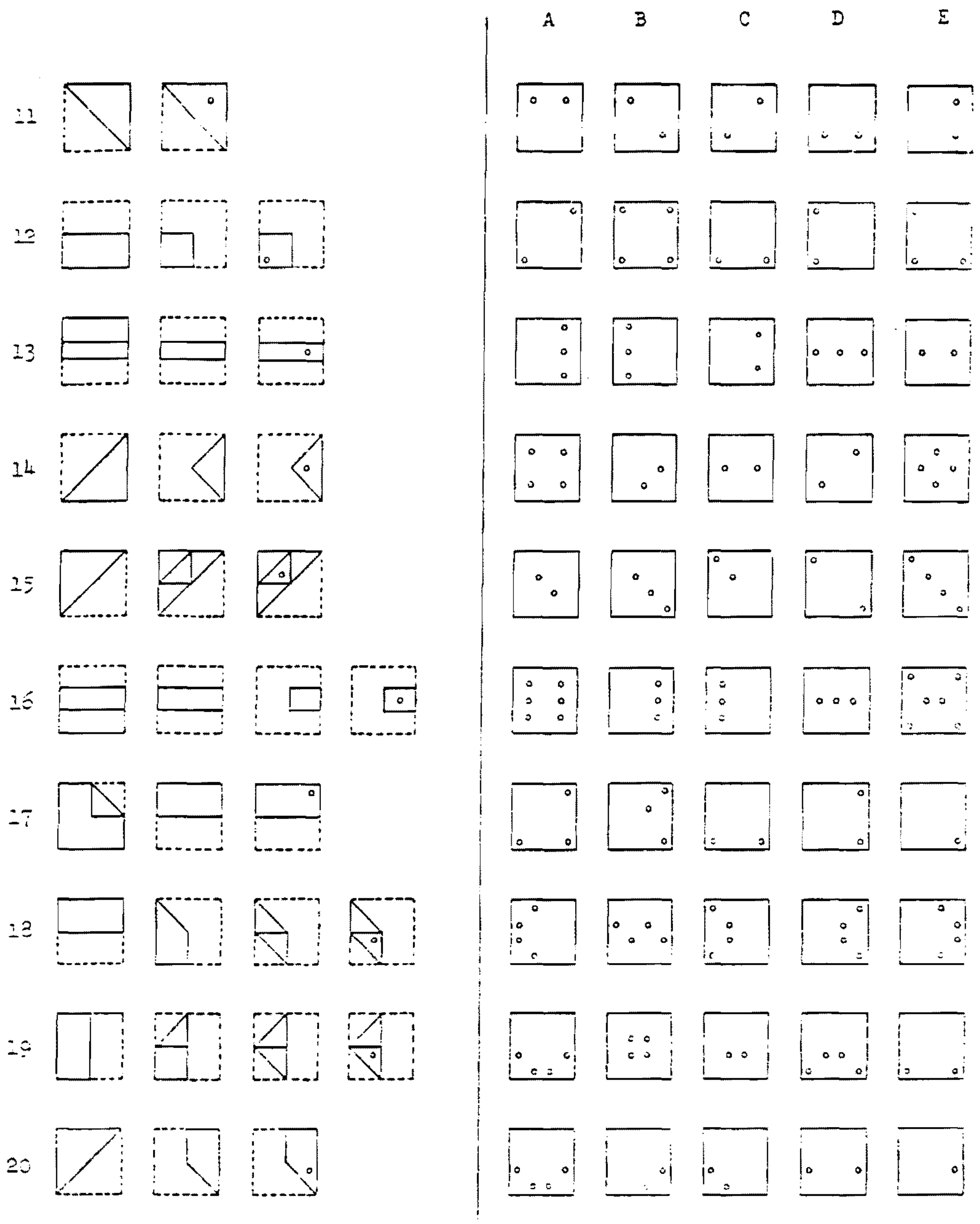
Part 1 (3 minutes)



DO NOT GO ON TO THE NEXT PAGE UNTIL ASKED TO DO SO.

[illegible].

Part 2 (3 minutes)



DO NOT GO BACK TO PART 1 AND

DO NOT GO ON TO ANY OTHER TEST UNTIL ASKED TO DO SO.

STOP.

Appendix D

Preliminary Analyses

Table 15. Manova Wilks' Lambda Statistic

Source	Value	F	Num DF	Den DF	p-value
Language	0.7954	2.0272	5	32	0.1013
Man. Infl.	0.1337	11.1024	10	64	0.0001
Interaction	0.5726	2.0575	10	64	0.0414

Table 16. Anova Test using Runs

Source	DF	Type III SS	Mean Square	F	p-value
Language	1	901.34	901.34	6.80	0.0132
Man. Infl.	2	14208.97	7104.49	53.63	0.0001
Interaction	2	1516.15	758.08	5.72	0.0070

Table 17. Anova Test using RTE

Source	DF	Type III SS	Mean Square	F	p-value
Language	1	1657.53	1657.53	6.45	0.0156
Man. Infl.	2	14811.47	7405.73	28.81	0.0001
Interaction	2	1850.76	925.38	3.60	0.0376

Table 18. Anova Test using Time

Source	DF	Type III SS	Mean Square	F	p-value
Language	1	74703.37	74703.37	5.85	0.0208
Man. Infl.	2	260285.50	130142.75	10.19	0.0003
Interaction	2	8076.84	4038.42	0.32	0.7309

Table 19. Anova Test using Edits

Source	DF	Type III SS	Mean Square	F	p-value
Language	1	589.34	589.34	0.16	0.6937
Man. Infl.	2	389929.21	194964.60	52.15	0.0001
Interaction	2	9011.42	4505.71	1.21	0.3114

Table 20. Anova Test using Syntax Errors

Source	DF	Type III SS	Mean Square	F	p-value
Language	1	2064.29	2064.29	1.28	0.2649
Man. Infl.	2	207712.82	103856.41	64.54	0.0001
Interaction	2	10763.14	5381.57	3.34	0.0465

Appendix E

Raw Data

Table 21 Description: General Information

Column	Description
1	Subject Id Number
2	Language Used (1 = Pascal, 2 = C++)
3	Managerial Influence Level (1 = Cannot Reuse, 2 = Moderate Encouragement, 3 = Strong Encouragement)
4	First Task Performed

Table 21: General Information

Subject Id	Language	MI Level	First Task
1	2	1	A
2	1	2	A
3	1	3	A
4	1	1	A
5	1	1	A
6	2	2	A
7	1	3	A
8	2	2	A
9	2	1	A
10	2	3	A
11	1	1	B
12	1	2	A
13	1	1	B
14	1	2	B
15	1	3	B
16	2	3	B
17	2	1	B
18	2	2	B
19	2	3	B
20	1	2	B
21	2	2	B

Table 22 Description: Task A Functionality Information

Column	Description
1	Task A Subtask Id Number
2	Task A Subtask Name
3	Task A Subtask Description

Table 22: Task A Functionality Information

Subtask Id	Subtask Name	Subtask Description
1	Employee Manage.	Maintain employee database.
2	Operation Access	Monitor acess of all on-line operations.
3	Payroll	Perform all payroll related operations.
4	Calendar	Maintain interacting employee appointment calendars.
5	Cost Center Manage.	Control departmental cost center accounting.
6	Company Store	Maintain an inventory system for the company store.
7	Entertainment	Play the hi-low and riddle games.

Table 23 Description: Task B Functionality Information

Column	Description
1	Task B Subtask Id Number
2	Task B Subtask Name
3	Task B Subtask Description

Table 23: Task B Functionality Information

Subtask Id	Subtask Name	Subtask Description
1	Widget Tracking	Control widget processing (station to station) on shop floor.
2	Station Manag.	Maintain station configuration and management information.
3	Corrosion Tests	Perform specific corrosion testing on widget data.
4	X-ray Tests	Perform specific x-ray testing on widget data.
5	Warehouse Manag.	Control storage and retrieval operations in the warehouse.
6	Customer Tracking	Maintain a customer data base.
7	Quote Manag.	Create price quotes for customers based on widget data.

Table 24 Description: Development Information

Column	Description
1	Subject Id Number
2	Task
3	Number of Runs
4	Number of Run Time Errors
5	Time Impact of Run Time Errors (minutes)
6	Number of Edits
7	Number of Syntax Errors

Table 24: Development Information

Subject Id	Task	Runs	RTE	Time Impact	Edits	Syn. Err
1	A	92	86	497	353	245
	B	75	66	288	336	277
2	A	38	45	312	194	141
	B	50	62	348	180	111
3	A	62	47	226	290	174
	B	45	38	277	154	120
4	A	69	85	515	318	261
	B	87	92	345	549	351
5	A	65	61	383	477	350
	B	50	57	366	323	273
6	A	22	23	173	203	147
	B	34	39	114	160	128
7	A	70	85	423	107	104
	B	48	69	268	194	136
8	A	37	24	68	201	174
	B	23	30	178	235	198
9	A	90	96	130	404	277
	B	99	122	622	479	306
10	A	44	55	376	267	237
	B	51	50	292	263	202
11	A	91	75	491	532	418
	B	70	92	471	352	269
12	A	33	42	223	202	146
	B	49	35	117	237	154
13	A	69	71	404	330	239
	B	102	117	596	447	327
14	A	51	66	424	139	123
	B	30	37	184	170	153
15	A	47	68	413	179	111
	B	68	77	505	185	129
16	A	28	27	170	214	167
	B	34	29	165	255	210
17	A	68	63	383	478	390
	B	75	90	390	302	247
18	A	28	27	173	206	155
	B	31	43	320	178	132
19	A	28	20	85	243	165
	B	44	44	267	163	151
20	A	54	66	317	250	177
	B	56	43	189	165	141
21	A	29	28	381	148	98
	B	18	42	162	185	142

Table 25 Description: Reuse Information

Column	Description
1	Subject Id Number
2	Component Name (percentage used for target system)

Table 25: Reuse Information

Subject Id	Comp. Name (percentage used)
2	alarm (100), bptree (70), cust (100), dates (100), empcode (70), empid (70), payroll (80), schedule (60), utils (50)
3	alarm (90), cust (80), dates (100), empcode (80), empid (80), error (30), payroll (60), queue (50), quote (60), schedule (50)
6	dates (90)
7	alarm (100), compid (70), ctrack (60), cust (100), dates (100), empcode (80), empid (80), idscheme (40), payroll (70), quote (60), schedule (60), strack (60)
8	alarm (100), corrosion (80), cust (80), dates (90), empcode (80), empid (80), payroll (90), schedule (70), stress (80)
10	cust (90), dates (90), empcode (90), empid (80), opcode (70), payroll (60)
12	alarm (90), cust (90), dates (100), empcode (80), empid (90), payroll (80), schedule (60), stress (70)
14	alarm (100), cust (80), dates (100), empcode (70), empid (80), payroll (100), schedule (70)
15	alarm (100), compid (60), corrosion (70), cust (100), dates (100), empcode (90), emphead (60), empid (80), payroll (80), quote (60), quote2 (70), schedule (70), shop (30), warehouse (50)
16	alarm (90), bptree (50), dates (80), empcode (80), empid (80), payroll (60), schedule (70)
18	alarm (100), dates (90), empcode (100), empid (90), list (80), schedule (70)
19	alarm (90), cust (90), dates (90), empcode (80), empid (80), list (50), payroll (60), schedule (80)
20	alarm (100), cust (90), dates (90), empcode (90), empid (90), payroll (70), quote (70), schedule (70), utils (60)
21	alarm (100), cust (80), dates (90), empcode (80), empid (90), list (80), payroll (70), schedule (70)

Table 26 Description: Previous Experience Information

Column	Description
1	Subject Id Number
	Number of Months (Academic Experience):
2	Programming in Pascal
3	Programming in C
4	Programming in C++
5	Programming in Smalltalk
6	Programming in Another Object-Oriented Language
7	Requirements Generation
8	Specification Generation
9	Integration of New Features Into Anothers Code
10	Writing External Documentation
11	Testing Own Source Code
12	Testing Anothers Source Code
13	Using UNIX
	Number of Months (Professional Experience):
14	Programming in Pascal
15	Programming in C
16	Programming in C++
17	Programming in Smalltalk
18	Programming in Another Object-Oriented Language
19	Requirements Generation
20	Specification Generation
21	Integration of New Features Into Anothers Code
22	Writing External Documentation
23	Testing Own Source Code
24	Testing Anothers Source Code
25	Using UNIX

Table 26: Previous Experience Information

	Academic Experience												Professional Experience											
Sub. Id	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	33	9	0	0	0	0	0	0	2	33	1	20	0	3	0	0	0	0	0	0	0	3	0	0
2	24	12	0	0	0	0	0	0	0	15	0	12	0	0	0	0	0	0	0	0	0	0	0	0
3	40	10	0	0	0	0	0	0	1	40	0	15	0	0	0	0	0	3	3	1	3	6	0	0
4	66	36	3	0	6	0	0	0	6	36	6	36	0	0	0	0	0	0	0	0	0	0	0	0
5	65	3	0	0	0	0	0	6	40	60	0	60	0	0	0	0	0	0	0	0	0	0	0	0
6	50	0	0	0	0	0	0	2	0	50	0	15	1	3	0	0	0	0	2	2	1	3	2	3
7	24	6	0	0	0	1	1	0	10	18	0	24	0	1	0	0	0	4	0	6	3	6	6	0
8	60	0	3	0	0	0	0	2	7	90	0	42	3	0	0	0	0	0	0	0	3	3	0	0
9	60	9	0	0	0	0	0	48	48	60	48	36	0	0	0	0	0	0	0	0	0	0	0	0
10	24	2	2	0	0	0	0	0	3	24	0	3	0	0	0	0	0	0	0	0	0	0	0	0
11	48	6	0	0	0	0	0	0	6	12	0	48	0	0	0	0	0	0	0	0	0	3	0	0
12	36	6	0	0	0	36	36	3	5	36	6	20	0	0	0	0	0	0	0	0	0	0	0	0
13	36	0	0	0	0	0	0	3	3	36	0	36	0	0	0	0	0	0	0	0	0	0	0	0
14	25	3	0	0	0	0	0	0	30	35	2	25	0	2	0	0	0	3	3	1	6	15	2	6
15	36	3	1	0	0	3	2	6	16	36	0	48	0	0	0	0	0	0	0	0	0	0	0	0
16	48	18	3	0	3	0	0	3	3	48	0	48	0	9	0	0	0	0	0	0	0	0	0	9
17	60	24	0	0	0	0	0	0	24	84	0	48	0	4	0	0	0	0	0	0	2	36	0	0
18	42	6	0	0	0	0	0	0	36	18	0	18	0	0	0	0	0	0	0	0	0	0	0	0
19	18	20	3	0	0	1	1	1	0	40	2	45	0	10	0	0	0	2	2	2	2	15	2	15
20	36	6	0	0	0	0	0	0	0	18	0	18	0	0	0	0	0	0	0	0	0	0	0	0
21	48	1	0	0	0	0	0	2	2	48	0	48	2	1	0	0	0	0	0	6	6	6	0	4

Table 27 Description: Human Factors Information

Column	Description
1	Subject Id Number
2	Overall QCA
3	Computer Science QCA
	Initial Confidence (1=Disagree thru 4=Agree):
4	Pascal
5	C
6	C++
7	Object-Oriented Paradigm
8	Software Engineering Concepts
	Reuse Disposition (1=Disagree thru 4=Agree):
9	Reuse Own Software
10	Reuse Others Software
11	Trust Others Code
12	Reuse Can Save Time
13	Never Reuse Software
14	Prefer C++ Over Pascal
15	Prefer Pascal Over C++
	Cognitive Abilities Test Scores:
16	Integrative Processes (IP-1)
17	Integrative Processes (IP-2)
18	Perceptual Speed (P-2)
19	Perceptual Speed (P-3)
20	Visualization (VZ-1)
21	Visualization (VZ-2)

Table 27: Human Factors Information

			Confidence					Disposition							Cognitive Tests					
Subj Id	OCA	CS OCA	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	2.70	2.90	4	4	2	3	3	2	2	1	4	2	3	1	100	85	43	70	91	100
2	2.60	3.00	3	3	1	2	2	1	1	2	4	4	1	4	78	48	55	91	66	100
3	3.30	3.96	4	2	1	2	3	2	2	2	2	2	1	4	70	78	43	97	91	93
4	3.00	3.20	4	3	3	4	3	3	2	2	4	1	2	4	78	85	51	95	86	95
5	3.60	3.60	4	2	1	3	4	2	3	2	3	3	2	4	70	93	61	95	86	85
6	2.99	3.38	4	3	1	1	2	4	3	3	3	1	3	2	78	25	45	90	96	100
7	2.60	3.00	4	3	2	2	3	1	3	2	4	3	1	4	63	40	47	78	89	55
8	2.70	3.30	4	2	2	2	2	3	3	2	3	2	3	2	85	70	51	97	94	85
9	2.43	2.70	4	3	1	2	3	2	2	1	4	3	2	3	70	85	35	71	75	93
10	2.90	3.00	4	3	3	3	4	4	4	2	3	1	4	3	85	78	67	80	80	35
11	2.90	2.50	4	3	2	2	3	3	2	1	2	1	1	4	100	85	57	80	88	85
12	3.00	3.10	4	2	1	2	3	4	4	3	3	1	2	3	93	78	76	89	80	85
13	3.10	2.90	4	1	2	2	2	3	3	2	2	1	1	4	48	70	65	79	93	40
14	3.40	3.70	4	3	2	4	4	2	3	1	4	2	2	4	78	70	51	89	75	93
15	2.47	2.60	4	3	2	2	3	1	4	3	4	2	1	4	78	93	45	92	85	93
16	2.50	3.50	4	4	4	4	3	4	4	2	3	1	4	1	70	93	65	88	88	93
17	3.40	3.90	4	4	1	2	3	2	3	1	3	3	4	1	85	85	55	90	58	90
18	2.16	2.55	4	3	1	3	3	3	3	2	3	2	1	3	93	93	69	92	79	70
19	2.80	3.10	4	4	2	3	4	4	3	2	3	1	4	1	70	85	59	65	95	93
20	2.15	2.56	3	3	1	3	3	1	3	2	4	3	1	4	85	85	65	94	62	65
21	3.40	3.60	4	2	1	1	4	1	3	1	4	4	2	3	93	48	47	77	75	75

Table 28 Description: Experiment Difficulty Ratings

Column	Description
1	Subject Id Number
	Difficulty Ratings (1=Easy thru 4=Difficult):
2	Introductory C++ Task
3	Task A
4	Task B
5	Development Questionnaires
6	Reuse Questionnaires

Table 28: Experiment Difficulty Ratings

	Difficulty Ratings				
Subject Id	Intro	Task A	Task B	Dev. Quest.	Reuse Quest.
1	3	4	4	3	NA
2	2	4	2	2	1
3	3	4	3	1	1
4	3	1	3	2	NA
5	3	3	3	2	NA
6	3	4	4	3	1
7	2	3	4	3	3
8	3	4	3	1	1
9	2	4	3	3	NA
10	3	4	4	4	2
11	3	4	4	1	NA
12	4	1	2	1	2
13	3	4	4	2	NA
14	3	1	3	4	3
15	4	3	2	1	1
16	1	3	3	3	3
17	4	4	3	2	NA
18	2	2	3	3	1
19	2	2	4	3	2
20	1	3	3	2	2
21	3	4	4	3	3

Vita

Vita

The author was born to Len and Dorothy Lewis on August 25, 1963 in Cleveland, Ohio. After moving to Lynchburg, Virginia, he graduated from Brookville High School in 1981. From there he went to Virginia Polytechnic Institute and State University (Virginia Tech) in Blacksburg, Virginia as a computer science major.

His undergraduate years were tempered by his participation in the cooperative education program, through which he gained commercial software development experience at the Babcock & Wilcox Company. After graduation in 1986, he immediately entered graduate school at Virginia Tech, encouraged by the research exposure he gained as an undergraduate in the area of software metrics.

His master's research involved developing a methodology for integrating metrics into large-scale software development environments. The Xerox Corporation in Rochester, New York sponsored the research.

Convinced at this point that a career in academia would best serve his love of research and teaching, he entered the doctoral program in computer science at Virginia Tech after graduating with his M.S. degree in 1988. Branching out from his concentration in the metrics area, he and advisor Sallie Henry defined an empirical study of software reuse, which is the focus of this dissertation.

As his college years finally draw to an close, he is searching for a suitable academic position at a major university. He looks forward to settling down with his fiancee, Sharon Mancuso, and establishing a career of research and instruction.