DISSERTATION

ESSENTIAL COMPETENCIES OF EXCEPTIONAL PROFESSIONAL SOFTWARE ENGINEERS

Submitted by

Richard T. Turley

Department of Computer Science

In partial fulfillment of the requirements

for the degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Fall 1991

COLORADO STATE UNIVERSITY

October 21, 1991

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY RICHARD T. TURLEY ENTITLED **"ESSENTIAL COMPETENCIES OF EXCEPTIONAL PROFESSIONAL SOFTWARE ENGINEERS"** BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

[redacted]

Advisor

[redacted]

Co-Advisor

[redacted]

Department Head

ABSTRACT OF DISSERTATION

ESSENTIAL COMPETENCIES OF EXCEPTIONAL

PROFESSIONAL SOFTWARE ENGINEERS

This dissertation presents a differential study of exceptional and non-exceptional professional software engineers in the work environment. The first phase of the study reports an in-depth review of 20 engineers. The study reports biographical data, Myers-Briggs Type Indicator test results, and Critical Incident Interview data for 10 exceptional and 10 non-exceptional subjects. Phase 1 concludes with a description of 38 essential competencies of software engineers. Phase 2 of this study surveys 129 engineers. Phase 2 reports biographical data for the sample and concludes that the only simple demographic predictor of performance is years of experience in software. This variable is able to correctly classify 63% of the cases studied. Phase 2 also has the participants complete a Q-Sort of the 38 competencies identified in Phase 1. Nine of these competencies are differentially related to engineer performance. A 10 variable Canonical Discriminant Function is derived which is capable of correctly classifying 81% of the cases studied. This function consists of three biographical variables and seven competencies. The competencies related to Personal Attributes and Interpersonal Skills are identified as the most significant factors contributing to performance differences.

Richard T. Turley
Computer Science Department
Colorado State University
Fort Collins, CO 80523
Fall 1991

ACKNOWLEDGEMENTS

I would like to acknowledge the influence and support of many individuals throughout the creation of this dissertation. Dr. James Turley provided me with early inspiration and support encouraging me to undertake this adventure. He continued to provide critical review and guidance at every step in the research and was particularly helpful in the statistical analysis of my results. Dr. Lillian Eriksen was also a strong supporter throughout the research and did a wonderful job of making statistical analysis understandable to a computer scientist.

Professor Gerry Johnson proved to be the ideal advisor for this research. His positive, can-do attitude kept me moving during the slow periods of the project. His gentle guidance encouraged my learning but also ensured that help was always at hand. Professor Charles Neidt was invaluable in his contributions to the research method applied. He consistently went the extra mile to identify methods which would be most effective in this project. He also provided much of the keen insight needed to interpret the results.

Professor James Bieman, as co-advisor, kept the research and its goals on track. He continued to ask the hard questions that made the result stronger. Professor Kurt Olender also participated in the review of my work and I thank him for his input. The final member of my committee is Dr. Jack Walicki. Jack has been a long-time friend and is personally responsible for starting me on this long road a few years ago. I wish to thank him for his encouragement and his support and for believing in the quest.

Marilyn Pultz provided invaluable assistance in the application of the Myers-Briggs Type Indicator to the research at hand. She also provided great insight into the interpretation of its results. Marilyn took a keen interest in the research and met with the Phase 1 participants to interpret the results of their individual Myers-Briggs tests.

I would like to thank the Hewlett-Packard Company for setting a fine example as an employer that supports continued education and for encouraging my research. I would particularly like to thank Chuck House for granting initial approval for the work and for taking a personal interest in it. I'd also like to thank Tom Christian for the financial support and his patient understanding of my preoccupation.

I would like to thank the managers and engineers of the Company studied in this research. Without their active participation and support this would never have been possible.

I would like to acknowledge my parents, Virginia and Frank Turley, for instilling in me a lifelong love of learning.

Finally, I would like to thank my family who survived extended periods with out a husband or father while I pursued this research. I thank Joyce, Jeff, Kimberly, and Nate for their understanding and encouragement.

DEDICATION

To Joyce, Jeff, Kimberly, and Nate

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The wide range of intersubject variability has been treated as a source of variance that must be compensated for in experiments on other factors. At the same time, however, the source and properties of this variance represent important research questions in their own right.[1]

Individual differences in performance between software developers[2] have been noticed and measured. Differences have been treated as a source of undesirable variance and significant steps are taken to factor it out. This research is a study of this variance in an effort to improve the productivity of all software developers.

This study is based on the premise that *exceptional* software engineers exhibit different skills which they apply to the problems of software engineering. These unique skills can be identified by careful differential study of experienced software engineers. Further, once these skills are recognized, they can be transferred to the software engineering population at large. Thus *all* software engineers can be taught valuable improvement skills. These results provide a criterion for implementing improved toolsets, and allow for the use of the appropriate engineers on each project.

The objective of this dissertation is to help unlock the potential of individual software engineers. The study is based on the thesis that:

---

[1] R. E. Brooks, "Studying Programmer Behavior Experimentally: The Problem of Proper Methodology," *Communications of the ACM*, Vol. 23, No. 4, pp. 207-213, April 1980.

[2] In this dissertation the terms *software developer*, *software engineer*, and *programmer* are used interchangeably to refer generically to all participants in the various aspects of the software life-cycle: designers, coders, testers, and maintainers.

Highly productive software developers exhibit dramatically different skills, techniques, and attributes than others in the process of programming. It is possible to study these developers and discover their unique skills, techniques, and attributes in programming.

The research proposes to answer the question:

What are the skills, techniques, and attributes used by skilled programmers that are not used by less skilled programmers?

As software costs and delivered lines of code increase, the need for more effective software development becomes increasingly apparent [Boeh 88]. The complexity of current designs are exceeding the capabilities of our top programmers. If the software industry is to maintain its ability to deliver high quality software on reasonable schedules, it will need to dramatically improve its ability to develop software.

Much effort has been placed in the development of *engineering* approaches to software development such as *software tools, coding practices,* and *test technology,* but the overwhelming determiner of software production productivity is still *personnel and team capability.* Boehm [Boeh 81] found personnel and team capability to be twice as important as the next most important productivity factor. By studying exceptional programmers, the individual capabilities which most influence performance can be identified [Curt 81]. This research has potential implications for the teaching of programming, the evaluation of programmers, and the selection of programmers.

Most research into the development of software focuses on the individual only to the extent that individuals are members of a larger development effort. Although the team is a critical component in software development, most research misses a fundamental opportunity to identify and exploit the proven ability of highly talented individual contributors. Weinberg, in his text *The Psychology of Computer Programming* [Wein 71], attacked this dilemma and observed that "Our profession suffers under an enormous burden of myths and half-truths." The industry has a great lore about the

factors affecting software productivity, but few facts are known. Boehm cites a 25-to-1 ratio between the most productive and least productive software developers and a 10-to-1 difference in their error rates [Boeh 88]. If the personal attributes of these most productive individuals could be understood, a number of exciting opportunities present themselves:

* Understanding the characteristics of the most successful software developers could lead to the improvement of *all* software developers.
* Once the characteristics are understood, it may be possible to develop specific toolsets and aids to further increase the productivity of these individuals.
* A valuable criterion of the selection of software developers may be discovered.

Brooks [Broo 87] suggests that the "conceptual essence" of software development requires that new paradigms be invoked for significant increases in software productivity. He identifies the "use of great designers" as one of five promising approaches. Boehm [Boeh 83] defines seven basic principles of software engineering. Principle 6 is to "use better and fewer people," recognizing that individual performance variations can overshadow other characteristics affecting development productivity. An additional benefit to using *fewer* people is the reduction of communications overhead required for a project. This recognition of the value of the individual motivates this research.

Traditional experimental approaches to meeting the above objective start with an individual's experience and prejudices about software development [Broo 75]. A technique for improvement is proposed, implemented, and tested [Shne 76, Curt 79]. The results of these experiments are then analyzed and often valuable results are achieved.

This study breaks with tradition. The key is to start with individuals who are acknowledged for their software ability. This study will focus exclusively on professional software developers. The results will be more significant than studies of students since they will be more generalizable into a work setting.

The focus on the top individual contributors breaks with the traditional emphasis on the team. This study is not meant to diminish the value of the team but rather to enhance it by ensuring that each individual is operating at peak productivity.

Chapter 2 of this dissertation will review the literature for results and methods used in prior research on individual performance differences. Chapter 2 provides an overview of the most important research approaches including tightly controlled, well-defined experiments and less constrained, qualitative psychological techniques. Chapter 2 provides in-depth analysis of several relevant similar studies and builds the case for the techniques to be used in this dissertation. The chapter specifically addresses *Who to Study, What to Study,* and *How to Study.* The chapter explores the research tools required to collect data. It also discusses the statistical analysis techniques used to analyze the data.

Chapter 3 presents the specific research method and design to be used in this dissertation. It provides the specifics of how subjects are selected for study, how the research instruments are administered and analyzed, and how the results are captured. Chapter 3 indicates that the research is divided into two phases. Phase 1 corresponds to the *qualitative* portion of the research in which the competencies associated with the job of software engineering are first uncovered. Phase 2 corresponds to the *quantitative* portion of the research in which the competencies discovered in Phase 1 are validated and considered on a differential basis between exceptional and non-exceptional performers.

Chapter 4 presents the data collected in both Phase 1 and Phase 2. It presents the biographical data for the entire sample in order to characterize the population under study. It reviews the biographical information on a differential basis to determine if any simple predictors of performance exist. Chapter 4 also presents the competency information. The Phase 1 portion presents the competencies discovered and discusses the process of creating a single list of competencies from multiple sources. The Phase 2 portion presents the results of a sorting exercise in which participants rank order the Phase 1 competencies relative to the individual's actual behavior. Finally, the competencies are considered on a differential basis between exceptional and non-exceptional performers in order to determine which competencies are associated with exceptional performance.

Chapter 5 discusses the data of Chapter 4 and presents the research results. These results include individual biographical characteristics as well individual competencies which are related to exception performance. Further, Chapter 5 develops a model capable of predicting exceptional or non-exceptional performance based upon a set of predictor variables.

Chapter 6 discusses the implications of this research and reinforces the work's most important results. The chapter also discusses some areas of research for further consideration.

CHAPTER 2

REVIEW OF PRIOR RESEARCH

The investigator who is well versed in the literature now has a set of expectations the data can defy.[3]

This chapter will review the prior research relevant to this dissertation. The chapter opens with a review of basic research approaches and proposes a behavior-oriented approach to studying the process of software engineering. The chapter continues by critiquing significant research done in the area of software psychology related to individual performance. The chapter continues to explore the questions of *What to Study, Who to Study,* and *How to Study.* The chapter proposes specific research tools and statistical analysis techniques as appropriate for this dissertation.

2.1 Research Approaches

Numerous researchers have attempted to catalog the realm of possible research approaches applicable to behavior-oriented software engineering [Shne 80, Mora 81, Basi 86, Curt 80, Curt 87]. In general, the approaches lie along a continuum between tightly controlled, well defined experiments which may have limited generalizability and more broadly defined, less constrained studies which stress qualitative psychological techniques. Researchers approach the problem from either a Computer Science or a Psychology point of view. The Computer Science perspective stresses

[3] G. McCracken, *The Long Interview*, Sage University Paper Series on Qualitative Research Methods, Vol. 13, Newbury Park, CA, p. 31, 1988.

understanding the effect of formal education and problem structure on individual performance. The focus is on individual skills and techniques. The Psychology perspective focuses on individual personality and problem solving approach. These psychology studies attempt to discern the individual's personality attributes and how they relate to solving problems.

The merger of these two perspectives resulted in the new field of Software Psychology. Software Psychology is first discussed in Weinberg's classic text *The Psychology of Computer Programming* [Wein 71]. Weinberg proposes four fundamental approaches to the study of programming: *Introspection, Observation, Experimentation,* and *Psychological Measurement. Introspection* is a process in which a programmer analyzes his own thoughts and skills in an attempt to use self-evaluation as a discovery mechanism. *Observation* allows an impartial observer to see *what* a programmer is doing but not *why* he is doing it.

Experimentation allows for measurement of the observed result but at the cost of being too focused. At issue here is a tradeoff between the *scope* and the *generalizability* of the research. A rigid experiment is generally of narrow scope and provides detailed results within its domain. Such a narrow scope may not allow for sufficient generalizability of the results. For example, a detailed study on the effect of indentation on comprehension of FORTRAN programs by first year students does not generalize to a conclusion for experienced professionals, or to other programming languages or environments.

Psychological Measurement may provide the richest and most fertile ground for research. This area probes the fundamental mental and psychological processes at work in the developer's mind. Since this field is in its infancy, the clarity and precision of the results are weaker.

Shneiderman [Shne 80] proposes a similar hierarchy of research methods. His list includes: *Introspection, ProtocolAnalysis, Case Studies/FielaStudies,* and *Controlled Experiments.* *Protocol Analysis* is a structured form of Introspection merged with Observation in which a written or recorded transcript is generated by an experimental subject and analyzed by the researcher. Shneiderman points out that *Case Studies/Field Studies* lack the experimental controls necessary to provide statistically significant results. He favors *Controlled Experiments* which limit the independent variables, control for bias, measure the dependent variables, and perform statistical analysis.

The bulk of the research conducted to date has favored the tightly controlled experimental approach. A number of studies have attempted to correlate easily measured a priori factors which could be a basis for predicting programming performance. These studies have shown mixed results. Evans and Simkin [Evan 89] studied students in an entry-level business computer class in an attempt to predict class performance and test results. The Evans and Simkin study collected 34 easily measured demographic, academic, experience, and behavioral variables. The researchers tried to correlate these variables with performance. Evans and Simkin could not account for more than 23% of the variation in performance based upon these 34 variables.

On the other hand, Chrysler [Chry 78] was able to explain over 85% of the variance in performance based on only thirteen program variables and five programmer variables. A major distinction between these two studies is that Evans studied novice programmers in university classes while Chrysler studied experienced programmers in industry. Given that Chrysler found that only thirteen of the program factors studied proved significant while *all* five of the programmer factors were significant, it is clear that in the study of experienced software engineers individual programmer differences are significant, and programmer differences may be among the *most* significant factors.

As a further note on the Chrysler study, five factors were found to explain more than 80% of the variance in performance. Four of these factors were attributes of the program under development: *number of input files, number of control breaks in logic, number of input edits required,* and *number of input fields required.* The most significant programmer factor was *programmer experience at this facility.* The other programmer variables were: *number of months of programming experience, number of months of programming experience using the COBOL language, number of months of experience using the specific COBOL language compiler to be used for the subject program,* and *number of months of experience in programming business applications.* There was a high degree of interdependence among all of the programmer variables.

In a similar study Moher and Schneider [Mohe 81] studied both students and professionals searching for factors which could predict programming performance. Moher and Schneider collected answers to 78 biographical questions with 53 of those relating to specific programming experiences. The questions fell into the categories of general demographic information, general educational background, computer science education, general programming experience, and specific programming experience. For students, nearly every biographical item proved to be correlated significantly with the performance measures. The model generated was able to explain 45-55% of the performance variability across 3 programming tasks. By contrast, only one of the background measures were useful for predicting performance for the professionals. The only outstanding biographical predictor of performance was the number of years of programming experience. The study found that differences in performance of up to 3:1 could be explained well by differences in the number of years programming experience.

The foregoing research suggests that simple predictors of performance (beyond simple experience) do not exist. Searching for these predictors may be misplaced effort. This dissertation includes such a search in order to verify that simple predictors of performance do not exist. The dissertation emphasizes a *behavior-oriented* approach to researching the actual process of software engineering. This behavior-oriented approach provides greater understanding as to the "why" of exceptional performance.

2.2 Comparable Studies and Critique

> *Variance of programming performance attributable to individual differences between the programmers almost obscured the difference related to the programming mode*[4]

Many papers discuss experiments in software engineering performance [Curt 79, Shne 80, Para 90]. The four papers discussed in detail below illustrate the current *state of the art* in research in software engineering performance.

2.2.1 MCC Study of Designers

Guindon et. al. [Guin 87a, Guin 87b] report a study in which three experienced software developers were videotaped during the process of developing a design solution. Each subject provided a solution to a distributed systems design problem: *Design an N-lift control system to be installed in a building with M floors.* The researchers observed that the development process was not linear. Rather, the designers operated simultaneously at various levels of abstraction and detail. That is, subjects moved frequently between the problem domain and the solution domain. In addition, subjects

[4] Chrysler describing the Sackman study:
E. Chrysler, "Some Basic Determinants of Computer Programming Productivity," *Communications of the ACM*, Vol. 21, No. 6, p. 473, June 1978.

exhibited a highly iterative, interleaved and loosely ordered process over the life-cycle points of *requirements, design,* and *code*. Each designer exhibited a markedly different approach to design.

* The first subject was a trained software engineer with a Masters of Science degree in Software Engineering and five years of experience. He used a *meta-schema* about design which allowed an exploration of the problem environment before adopting an initial and final solution. That is, this designer focused on the *process* of design rather than on the design itself.
* The second subject was a Ph.D. candidate in Computer Science with less than three years of experience in logic programming. He did not consider alternative solutions and utilized a *generate-simulate-debug* strategy. This individual was too quick to focus on the initial solution without considering alternatives. He also had difficulty integrating all of the constraints on the solution.
* The third designer held a Ph.D. in Electrical Engineering and had more than ten years of experience in communications systems and hardware architecture. He was most familiar with the problem domain and used *specialized design schemas* relevant to distributed systems. This designer decomposed the problem into smaller subproblems and then addressed those with "standard" solutions from his design repertoire.

Guindon [Guin 88] recognized that an intrinsic aspect of system design is that requirements are incomplete and ambiguous. Hence a key part of design is clarification and completion of the requirements. The observed design activities fell into three categories:

1) *Mental and external simulations of scenarios in the problem domain.*
2) *Understanding and formalization of the requirements.* This included abstraction of critical point and testing of the consistency of requirements against their own knowledge.
3) *Definition, representation, and mental or external simulations of the design solution at various levels of abstraction.*

A fundamental conclusion from this study was that designers complete their designs by shifting between design activities and between different domains of knowledge at different levels of abstraction. Strict top-down design approaches were not exhibited. Guindon refers to this process as *serendipitous* or *opportunistic* design.

Guindon's study identifies the significant design process control strategies and recognizes that any design can be a mixture of these:

* *design method driven*, when the design process is underlied by a design method providing a plan for the design process,
* *heuristic driven*, when the design process is underlied by heuristic rules to reduce complexity,
* *goal driven*. when the design process is underlied by a problem decomposition plan in relevant specialized design schemas,
* *position driven*, as when the design process is underlied by a small set of a priori issues and selected values on these issues,
* *data driven*, as when the design process is underlied by an exploration of the problem environment and recognition of partial solutions at various levels of detail,
* *repair driven*. as when the design process is underlied by the need to debug or repair a faulty aspect of the solution, especially in a generate-simulate-debug strategy.

The selected design strategy proved to be a function of the individual subject's experience. This led to the conclusion that the cognition process for software development is highly dependent upon the domain experience of the subject.

Of particular interest in this study was the use of an observational technique for gathering information. The videotaping of the experiment allowed the researchers to listen to the subjects speak as they described their design process in order to obtain *thinking aloud reports.* Also, collecting the notes used in the designs allowed the researchers to reconstruct the actual design sequence. The Guindon experiment implies that it is realistic to obtain significant results through observation. This research proposes the significance of understanding the cognitive process as an important step in predicting performance.

The Guindon study used *protocol analysis* where the researchers listen to the subjects description of their development process in order to uncover the cognitive factors at work in design. Guindon clearly illustrates the value of studying individuals

in order to uncover the cognitive process. Since the study used a limited sample size, no general conclusions can be drawn. However, the design processes uncovered warrant further investigation.

2.2.2 **Evans - Simkin Study of Student Proficiency**

The Evans-Simkin study represents the use of a structured, scientific, controlled experiment approach to conducting research. The formalism applied in this study shows how issues of a personal psychology can be adequately quantified and studied. Evans and Simkin study multiple possible causal factors in the search for performance indicators. Although the final dependent variables were clear performance metrics (grades), the independent variables included easily collected biographic information, measured problem solving skills, and psychological profiles.

Evans and Simkin [Evan 89] studied students in an entry-level business computer class to determine:

1. What is the best way to measure a person's understanding of computer concepts?
2. What factors best predict this understanding?
3. How can we measure cognitive processes that might also predict this understanding?
4. If we can measure cognitive processes, which are the best predictors?

The study started with a 100-question survey administered at the beginning of the semester. This survey collected demographic information, performed psychological profiling, and evaluated general problem solving skills. The dependent variables in the study were exam and homework grades.

Each dependent variable was tested to see if it could be predicted by the independent variables. There was a different result for each dependent variable. For example, four variables best accounted for the variability in homework scores: high

school math courses, typing skills, sensing, and age. Sensing is one of the Myers-Briggs Type Indicator scales and indicates that the person would rather work with known facts than to look for new possibilities and relationships. However, the BASIC programming score relied on computer access, mother's occupation, the ability to speak a second language, the letter-set problem-solving variable[5], and the degree of student introversion to explain its variability.

In conclusion, the authors note that:

> *"No single set of variables - demographic, behavioral, cognitive, or problem solving - dominated the others as a 'best' set of predictors of student performance. Rather, the research results suggest that several factors from all four areas may be useful in forecasting computer aptitude."*

The Evans-Simkin study illustrates the problems associated with the search for predictive factors and the use of student subjects. It is essential to select the correct metric of successful performance. In the case of programming, it is not clear that grades constitute the best metric. The study does not address the issue of which individuals produced the *best* code or who completed their development in the most *efficient* manner.

Looking for predictive factors for performance is always dangerous without a supporting theory for explanation. Many factors will correlate with performance (e.g. *mother's occupation)* but may indeed have little to do with the cause of the performance. The search for these causal factors must be based on theory.

---

[5]Results from a particular question used on a problem solving test. The question is: Indicate the set of letters that is different:

BCDE FGHI JKLM PRST VWXY

The extension of results from the study of students to the realm of experienced professionals is unclear at best. Since the correlation between grades and professional success is not high [McCl 73], there is no reason to expect the predictive factors from a study of students to generalize to the study of professionals.

2.2.3 Kagan - Douthat Study of Learning FORTRAN

The Kagan-Douthat study is an example of the use of extensive psychological testing in order to predict performance. Through the use of controls and statistical techniques, significant predictability based on solely psychological characteristics was uncovered. The study points to the change in characteristics which become important as experience increases. Since this study focussed on students learning to program, it leaves open the issue of what happens as programmers become *very* experienced.

Kagan and Douthat [Kaga 85] posed the question, *"Does a tendency towards introversion give students an 'edge' in learning the science of computer programming?"* They also sought to determine if such an "edge" is persistent in time.

Three hundred twenty six students enrolled in an introductory FORTRAN class completed questionnaires to determine personality traits based on a number of psychological measures. These psychology measures were Eysenck's Personality Inventory, the Crowne-Marlowe Social Desirability Scale, Self-Monitoring of Expressive Behavior, the Hostility Inventory, and a Type A Behavior measure. The study found that "achievement in the latter half of the course was significantly associated with a cluster of traits that fit the global definition of 'introversion'." They found that temperament became more relevant to achievement as the course progressed. "In sum, being relaxed, stable, and aware of one's self in a social context was conducive to achievement *early* in the course, while the tendency to be withdrawn, hard-driving,

and ambitious appeared conducive to learning in the *final* portion of the course." There was no statistically significant difference between majors and non-majors in either temperament or achievement.

2.2.4 **Love Study of Student Performance**

The Love [Love 77] study illustrates the search for predictive factors in programming performance. The search itself is fairly brute force in that a wide array (24 factors) of data are collected for *each* run of a student assignment. Each factor is considered in an *analysis of variance* calculation to determine the factors most likely to be predictors of the observed performance variance. The study also attempts to relate human information processing abilities to computer programming performance. The information processing ability is measured prior to the class assignments and the correlated with performance. Again, this is a study of student performance and leaves open the issue of its relevance to professionals.

[Love 77] presents a study of computer science students. During the course of a semester, each student submitted a cover sheet with each run of a class assignment. The cover sheet asked the student to answer questions pertaining to that run of the program. The questions covered a broad range of suspected performance factors relating to the student's work style, environment, and state of mind.

The most significant variables affecting the success of a particular run of the program proved to be:

1. How many previous attempts the student had made to run the program.
2. The student's reported ability to concentrate.
3. The time of the day.
4. The time the student had spent in the design of the program.
5. The student's preparation time.
6. The time required to locate the previous error.
7. The student's reported overall confidence in this run.

In addition, the study collected further dependent variables designed to assess the basic skill level of each subject. The dependent human information processing variables were:

1. Performance on a continuous paired-associates task. This task asks the subject to "print out" from the programmer's memory the proper value for one of 4 variables whose assignments have been projected briefly on a screen. The assignments are made at a varying distance from the "print" request.
2. Digit-span. A variable length series of two digit numbers are presented at the rate of one per second. The subject is then asked to recall the list in order.
3. Perception speed. This is a speed and accuracy test for comparing strings of digits in which the subject compares variable length pairs of strings to determine if they are the same or different.
4. Subjective organization of words in a free-recall learning paradigm. The subject is tested on the number of words correctly recalled and the degree to which recalled words were clustered into semantic categories.
5. Classroom performance measured by grades.

Significant results include:

1. Students did a poor job of estimating the number of runs required to completion.
2. The number of changes made in programs does not correlate linearly with classroom performance. An "A" student is just as likely to require a few changes or hundreds of changes to get their programs working.
3. Those students who performed well in the *memory for programs* experiment as well as those who had higher scores on the *digit span test* took fewer runs to complete their programming assignments.
4. There was an inverse relationship between performance on the *free recall learning task* and the number of logical errors reported in the programs.
5. Students who did well in *remembering programs* took *longer* to locate errors in programs.
6. The longer one spent designing a program, the more runs required to complete the program. The more time coding and keypunching a program, the fewer runs.
7. Students with high scores on the *perceptual speed test* tended to make more changes.
8. Students who wrote larger programs tended to make more changes.
9. Only two measures were significantly related to score in the classroom: mean time to locate an error in a program, and frequency of syntax errors. *(Course grade seems to measure test-taking ability more than it measures programming performance as defined here.)*

10. The above results suggest two possible types of "A" students:
    1. Those who are able to develop a correct algorithm and implement it without making any logical changes, and
    2. Those who have both algorithm and implementation difficulties, but persist until the problem is solved correctly.

One surprising result is #6, the lack of extra design time resulting in improved productivity. It is likely that student assignments are not complex enough to benefit from a formal design method. Instead, the time spent designing is wasted because the problems are trivial enough to warrant jumping straight to coding. Perhaps, too, the inexperience programmer is not as capable in translating a design into actual working code.

This study begins to merge issues of cognition with other predictive factors. It attempts to uncover those cognitive abilities which are most closely related to improved performance. The study hypothesizes four programming related cognitive abilities and attempts to correlate these with the performance metric of grades. None of these hypotheses proved valid. This illustrates the danger of a *shot in the dark* approach to uncovering the cognitive processes at work in programming. The proposed significant cognitive abilities were correlated with other measures of performance (number of runs to complete assignment, and number of logical errors.) This suggests that measures of performance other than grades should be studied.

2.2.5 Issue of Cognitive Science

In nature hybrid species are usually sterile, but in science the reverse is often true. Hybrid subjects are often astonishingly fertile, whereas if a scientific discipline remains too pure it usually wilts.[6]

Curtis [Curt 87] proposes five psychological paradigms most often used in exploring programming problems:

1. Individual Differences.
2. Group Behavior.
3. Organizational Behavior.
4. Human Factors.
5. Cognitive Science.

In this paper Curtis proposes that the *Cognitive Science* approach to the problem is likely to be the most rewarding. Curtis notes that while the individual differences paradigm provides a method for predicting performance differences among programmers, it fails to offer an explanation of why these differences occur or how to reduce them other than through selection. The paradigm of cognitive science seeks to understand how knowledge is acquired, represented in memory, and how it is used in solving problems.

Cognition in programming is a very complex problem which will take extraordinary effort to solve. Also, it is a problem based more squarely in Psychology than in Computer Science. This dissertation proposes a pragmatic approach to the development of a cognitive model. It proposes to study the *behavior* of exceptional performers as it differs from that of non-exceptional performers. It is much easier to collect data on *what* someone does than on *why* they do it. This study will build a model of the behaviors associated with exceptional performance.

---

[6]Francis Crick, *What Mad Pursuit - A Personal View of Scientific Discovery*, Basic Books, Inc., New York, NY, p. 150, 1988.

2.2.6 Conclusions

These studies illustrate the current state of the art in research in the field of *Software Psychology*. These studies show that formal methods are being used to collect meaningful results in the area of predicting programming performance. They indicate that this field is replete with unsolved problems but that these research techniques are capable of providing significant results. The research focuses on biographical information, skills, and individual attributes as possible predictors for performance.

These studies also show that open problems remain. Curtis [Curt 86a] cites a major problem with the research completed thus far. The studies emphasize student subjects. The study of students can lead to two significant errors. First, novice performance is probably substantially different than experienced performance. Second, student problems are generally narrow and well defined. Thus the generalizabilty of results to practicing professionals is questionable. These large studies have also focused on easily measured biographical data. The studies that have probed an individual's approach or *processes* in depth have only been performed with a small numbers of programmers.

In essence, these prior studies serve as the research prototype. The studies discussed above have used many of the methods proposed and have studied the area of interest in this dissertation. Since the results have been significant, the above studies imply that this research will produce substantial results.

2.3 **Research Methods**

Several papers suggest powerful approaches to conducting research in the realm of *Software Psychology* [Weis 74, Broo 80, Mohe 81, Shei 81, Mohe 82, Basi 86]. These papers suggest appropriate answers to the questions of *Who to study; What to study;* and *How to study.*

These research areas are clearly on the boundary between Computer Science and Psychology. The field of Psychology provides the experimental methodology for formal recording and analysis of field data. Much of this method springs from field studies of other cultures. The statistical methods used are also critical to these studies. The Psychology backdrop also provides the connection to Cognitive Science. In order to adequately address the research results obtained by psychology methods, connections must be made to other areas of human task performance.

On the other hand, a Psychology-*only* approach to the study of programmer performance is unjust. The Computer Scientist must guarantee the relevance of the study by critiquing the research materials to ensure that they represent real-world software development. Also, it is the Computer Scientist who will be able to apply these results to actual development situations. Thus, although this research is in a boundary area, it is a bona-fide and valuable area of Computer Science research.

2.3.1 Who to study

The subjects must be representative of the study population at large in order to be able to generalize the results. The size of the subject pool must be large enough to be able to ascribe statistical significance to the result. The subject pool must be uniform with respect to the factors not under study in order to not be left with significant unexplained variance in the result.

Most of the large studies to date have used student subjects in order to keep costs low. Thus, the programs studied were usually small and not very complex since they were completed as part of a programming class. Results using student subjects are not readily generalized to large, complex projects undertaken by experienced professional software engineers. A better approach is to study experienced engineers on the tasks in which they are currently engaged.

2.3.2 **What to study**

Historically, the state of the art in Computer Science has been motivated by *rules of thumb* for best practices. Experienced professionals would identify tools and techniques that seemed useful and promote these as *bestpractices.* The field of Software Psychology tries to quantify these results to make more substantiated recommendations. The range of these items was generally from the very specific to the very general. On the specific end, notational differences in languages like conditionals, control flow, and data types were studied to see their effect on performance. More general along this continuum was the study of programming practices like flowcharting, indenting, variable naming, and commenting. At the most general end of the continuum was the study of tasks. Here the study of knowledge representation and cognition are applied to the problems of learning, coding strategies, and debugging. It is at this more general end of the *What to study* continuum that this research will reside.

While studying these more general tasks, this dissertation will focus on the study of programmer characteristics. These fall generally into two categories: software-independent characteristics and software-dependent characteristics [Mohe 82]. Software-independent characteristics include physical characteristics,

general intelligence, and formal education. Software-dependent characteristics include programming aptitude, programmer skill, programming experience, and formal programming education. This dissertation will primarily study software-dependent characteristics.

This dissertation will study *competencies.* A *competency* is any personal characteristic or attribute that contributes to effective performance [Char 82]. A *job competency* is any attribute that contributes to doing a specific job well. These attributes can be specialized knowledge, an ability, an interest, a trait, or a motivation. However, they are not a job competency unless they contribute to doing the job well.

The case for studying *competence* rather than *intelligence* was made by McClelland [McCl 73] in relation to the lack of predictive validity in current intelligence tests. McClelland argues that tests which sample job skills are the best predictor of competence. In order to create the tests, the researcher must know which skills are necessary to achieve competent performance in a particular job. The aim of this study is to uncover these competencies.

2.3.3 How to study

Sheil [Shei 81] expresses the concern that the current state of research in this field represents the *pseudopsychology of programming* and stresses the adoption of formal research methods. In order to have an adequate study, the fundamentals of behavioral research (psychometric theory, analysis of variance, and multivariate techniques) must be employed. Studies range from the experimental applying a rigorous, systematic method to a narrowly defined problem, to a discovery process employing interviews and verbal protocols to obtain a wider, richer result in a broader area.

The following sections will describe the research tools most appropriate for a study of this sort. The Myers-Briggs Type Indicator is presented as tool for determining psychological type. This tool will be used in this dissertation to determine if significant type differences exist between exceptional and non-exceptional engineers. The Critical Incident Interview coupled with Protocol Analysis is used to uncover the competencies associated with the job of software engineering. Q-Methodology is used to rank order the competencies uncovered and to determine if any of the competencies are associated with exceptional performance. Finally, Discriminant Analysis will be discussed as a statistical tool for developing a predictive model of exceptional performance.

2.3.3.1 Myers-Briggs Cognitive Style Type Indicator

Smith was a classic computer nerd - A Hobbitlike little man, short on social graces and all but innocent of personal hygiene, with the chopped-off blond curls of a fourteenth- century monk.[7]

An abbreviated version of the Myers-Briggs Type Indicator (MBTI) was used by Evans and Simkin [Evan 89] in their study of programmer productivity. This study demonstrated correlation between the *introversion, intuitive,* and *judging* types and performance on exams. A detailed description of the MBTI appears in [Shne 80]. The 166 question format computes a score for four contrasting personality pairs. Table 2.1 provides a general description of the quadrants used in the MBTI.

The Myers-Briggs Type Indicator's purpose is to identify, from self-report of easily recognized reactions, the basic preferences of people in regard to perception and judgement, so that the effects of each preference, singly and in combination, can be established by research and put to practical use [Buro 89]. The four preferences are

[7]Frank Rose, *West of Eden - The End of Innocence at Apple Computer,* Penguin Books, New York, NY, p. 51, 1989.

TABLE 2.1 Description of MBTI's 4 Preference Scales[8]

EXTROVERT VS. INTROVERT

People who prefer *extraversion* tend to focus on the outer world of people and the external environment. When you are extraverting, you are energized by what goes on in the outer world, and this is where you tend to direct your own energy. Extraverts usually prefer to communicate more by talking than by writing. They need to experience the world in order to understand it and thus tend to like action.

People who prefer *introversion* focus more on their own inner world. When you are introverting, you are energized by what goes on in your inner world, and this is where you tend to direct your own energy. Introverts tend to be more interested and comfortable when their work requires a good deal of their activity to take place quietly inside their heads. They like to understand the world before experiencing it, and so often think about what they are doing before acting.

SENSING VS. INTUITION

One way to "find out" is to use your *sensing* function. Your eyes, ears, and other senses tell you what is actually there and actually happening, both inside and outside of yourself. Sensing is especially useful for appreciating the realities of a situation. Sensing types tend to accept and work with what is "given" in the here-and-now, and thus become realistic and practical. They are good at remembering and working with a great number of facts.

The other way to find out is through *intuition,* which shows you the meanings, relationships, and possibilities that go beyond the information from your senses. Intuition looks at the big picture and tries to grasp the essential patterns. If you like intuition, you grow expert at seeing new possibilities and new ways of doing things. Intuitive types value imagination and inspirations.

THINKING VS. FEELING

One way to decide is through your *thinking.* Thinking predicts the logical consequences of any particular choice or action. When you use thinking you decide objectively, on the basis of cause and effect, and make decisions by analyzing the evidence, even including the unpleasant facts. People with a preference for thinking seek an objective standard of truth. They are frequently good at analyzing what is wrong with something.

The other way to decide is through your *feeling.* Feeling considers what is important to you or to other people (without requiring it to be logical), and decides on the basis of person-centered values. When making a decision for yourself, you ask how much you care or how much personal investment you have, for each of the alternatives. Those with a preference for feeling like dealing with people and tend to become sympathetic, appreciative, and tactful. (It is important to understand that the word "feeling," when used here, means making decisions based on values; it does *not* refer to your feelings or emotions.)

JUDGMENT VS. PERCEPTION

Those who take a *judging* attitude (either thinking or feeling) tend to live in a planned, orderly way, wanting to regulate life and control it. When you use your judging function, you like to make decisions, come to closure, and then carry on. People with a preference for judging prefer to be structured and organized and want things settled. (It is important to understand that "judging" as used here does *not* mean judgmental; any of the types can be judgmental.)

Those who prefer a *perceptive* process when dealing with the outer world (either sensing or intuition) like to live in a flexible, spontaneous way. When using your perception, you are gathering information and keeping your options open. People with a preference for perceiving seek to understand life rather than control it. They prefer to stay open to experience, enjoying and trusting their ability to adapt to the moment.

---

[8]Isabel Briggs Myers, *Introduction to Type,* Consulting Psychologists Press, Inc., Palo Alto, CA, pp. 5-6, 1990.

assumed to interact in complex nonlinear ways to produce one of 16 psychological *types.* Each type is identified by a four letter set which relates to the dominant preference on each of the four preference axes. Hence an **ISTJ** score relates to an individual who scores as an *Introvert, Sensing, Thinking, Judging* personality type. Each of the 16 possible types has significantly different attributes as described in [Isac 88]. Further, the MBTI can provide a continuous score [Myer 85] for each of the four preference scales allowing for statistical analysis of significant differences.

2.3.3.2 Critical Incident Interviews

Flanagan [Flan 54] provides an overview of the Critical Incident Technique for data collection. The technique was introduced during World War II in the Aviation Psychology Program to study combat leadership and pilot disorientation. The technique has since been refined and applied to measures of performance, measures of proficiency, training, selection, job design, equipment design, and leadership.

The critical incident technique attempts to discover the critical job requirements that have been demonstrated to have made a difference between success and failure in carrying out an important part of the job assigned in a significant number of instances. The technique is based on two fundamental principles:

1. Reporting of facts regarding behavior is preferable to the collection of interpretations, ratings, and opinions based on general impression.
2. Reporting should be limited to those behaviors that according to competent observers, make a significant contribution to the activity.

The method consists of five major steps:

1. *Determination of the general aim of the activity.* The aim should be clear so that subjects can comment directly on relevant incidents.
2. *Development of plans and specifications for collecting factual incidents regarding the activity.*
3. *Collection of the data.* This is generally provided via interview with the subject. The technique requires only the simplest types of judgements from the subject.

4. *Analysis of the data.* The classification of the critical incidents is an inductive process based on the analysis of all respondents.
5. *Interpretation and reporting of the statement of the requirements of the activity.*

The technique of *Protocol Analysis* is described in [Webe 85, McCr 88]. The technique translates the verbatim copy of an interview to a generalized set of cross-transcript results. By using a formal process, there is a record of the analysis and the relations identified can be tied to specific utterances in the original transcripts. The process itself is a movement from the specific to the general. One must be careful to ensure that generalizations are valid and that meaning isn't changed in the process

The process moves from transcripts to results in the following stages. Stage 1 converts an utterance to an observation by recognizing it as significant. Until a sentence or phrase is identified as being relevant to the research, it has not entered into the analysis. In this step the transcript is read carefully with the research question in mind in order to identify those utterances that must be identified and collected for later study. This happens individually for each transcript.

Stage 2 develops the logical relationships that occur in the transcript. These relationships can be with the utterance itself, with the rest of the transcript, or with previous literature. Stage 2 is the first step in generalization in that it begins to attach meaning to the utterance and begins the process of classifying it.

Stage 3 refines the observation in relation to all of the other Stage 2 observations in all of the available transcripts. This stage moves from the study of one transcript to forming relationships across transcripts. The focus also moves away from the transcripts per se, and onto the observations themselves.

In Stage 4 the researcher uses judgement to look for patterns of inter-theme consistency and contradiction. Redundant themes are combined or eliminated.

Themes that do not appear useful for the research question are eliminated. All of the themes must be simplified and clarified in order to best represent the data from which they are derived.

Stage 5 creates the conclusions of the study by identifying and commenting on the patterns across the themes derived from the entire interview process. This stage results in the creation of the theses from the interview process.

2.3.3.3 Q Methodology

The Q Methodology encompasses the Q-Sorting Technique, which is designed to provide practical means for subjects to sort and researchers to analyze large lists of items [McKe 88]. The method stresses the reliance upon the individual's perception *of value* in a set of statements as the actual data under study. The technique has a long history being first promoted by William Stephenson in the mid 30's. His text on the subject [Step 53] continues to be a significant source of information on the methodology and the technique.

In *Q-Sort* a subject is asked to rank order a set of items against a specific condition of instruction. The ordering is *quasi-normal* in that it asks subjects to place the item in one of a limited number of bins or piles. The number of items is expected to far exceed the number of piles. Each pile maintains a specific relationship to the other piles. The number of items to be placed in each pile is meant to be proportional to a roughly normal distribution of the items. For example, if there were 10 items to distribute across five piles, the first pile would have one item, the second pile would have two items, the third pile would have four items, the fourth pile would have two items and the fifth pile would have one item. This approximates a normal distribution with the center pile being the center of the distribution.

Critical to the sorting is the *condition of instruction.* A subject may provide a radically different sorting based upon the instructions given. For example, in this study if a subject were instructed to sort competencies based upon the order which most related to being exceptional, there would be a different result than if the subject was instructed to sort them relative to their own behavior on the job. The same *Q-sample stimuli* can even be used with the same subjects under different conditions of instruction to study different aspects of an area.

2.3.3.4 Statistical Analysis

The technique of *Predictive Discriminant Analysis* is used to predict membership of experimental units into two or more criterion groups [Hube 89]. This technique uses a set of predictor variables and one criterion variable. The criterion variable is a grouping variable with two or more levels. That is, the analysis tries to place all of the samples into two or more groups based upon the predictor variables. Predictive discriminant analysis is a multivariate data-analysis method of the dependent type. Hence it is related to other techniques like multiple correlation analysis, canonical correlation analysis, and multivariate analysis of variance (MANOVA).

The Discriminant Analysis is based on the following assumptions [Klec 80].

1. There are two or more groups that are being distinguished.
2. There are at least two cases per group.
3. There can be any number of discriminating variables, provided that the number of variables does not exceed the number of cases minus 2.
4. The discriminating variables must be measured at the interval level.
5. No discriminating variable may be a linear combination of other discriminating variables.
6. The covariance matrices for each group must be approximately equal, unless special formulas are used.
7. Each group must be drawn from a population with a multivariate normal distribution on the discriminating variables.

Klecka comments on violating these assumptions. Discriminant analysis seems to be robust enough to tolerate some deviation from the assumption of a multivariate normal distribution on the discriminating variables and equal group covariance matrices. The consequence of violating these assumptions is some reduction in efficiency and accuracy. If the normality assumption is violated the procedure will be non-optimal and result in a greater number of misclassifications.

Klecka notes that "*With large samples, however, we can ignore the tests of significance or interpret them 'conservatively' when our data violate the assumptions.*" This assumption can be tested by observing the percentage of correct classifications of the cases under study. If this number is high, the violations of assumptions is not considered harmful.

Other problems which can affect discriminant analysis include large amounts of missing data, highly correlated variables, a variable with zero standard deviations within one or more groups, grossly different group sizes, and outliers. Tests for these and other conditions are supported by the SPSS/PC package used to generate the result [Frud 87].

In particular, the variables used in discriminant analysis should be tested for normality of distribution. This is accomplished in two major ways. First the *skew* of the distribution is tested. Second the *kurtosis* of the distribution is tested. In both cases, values of plus or minus 1.00 are considered acceptable for descriptive studies such as this [Cohe 83].

2.4 Conclusions

Two broad experimental approaches can be applied to empirical research on programming: tightly controlled experiments which are rigorous, systematic, and narrow in scope, or less tightly controlled exploratory experiments which allow for collection of a wider and richer range of data [Curt 86b].

A number of critical issues arise in behavior-based studies of this type. First and foremost, subject selection is critical. To study excellence in performance, one must be sure that excellence is observed! Prior research faced this problem in the study of creativity in a research setting [Neid 64]. To ensure that excellence is observed, one can create a suitable measure of excellence, refine the subject pool to ensure that only excellent performers are in the study group, and provide an appropriate control group for defining average performance. Subject selection involves a series of filtering criteria for further refining the group. The subject pool is further filtered by other objective measures. Neidt, for example, considered performance ratings as well as the reference to entries in a research progress log. Much of the literature has confined itself to the use of student subjects due to their availability and economy. The industrial investigations have generally studied only single individuals or very small groups.

Figure 2.1 summarizes the research area by modeling the refinement of choices as a funnel. The field of study for this dissertation is between Computer Science and Psychology. It uses psychology methods to study performance of software engineers. The experimental design is broadly defined. That is, rather than creating a tightly controlled experiment in order to validate a particular result, this dissertation will perform a broad study attempting to determine which competencies are most like the behaviors of exceptional software engineers. The experimental method is protocol analysis of critical incident interview transcripts. The subjects are all experienced

software engineers rather than novices or students. Studies such as Moher and Schneider [Mohe 81] demonstrate that there are significant differences in the attributes of high performance students and professionals. This dissertation studies individuals. Although much of the task of software engineering is a team effort, this dissertation will study the role of the individual and individual behaviors. Finally, the dissertation explores software dependent characteristics of the individual's performance.

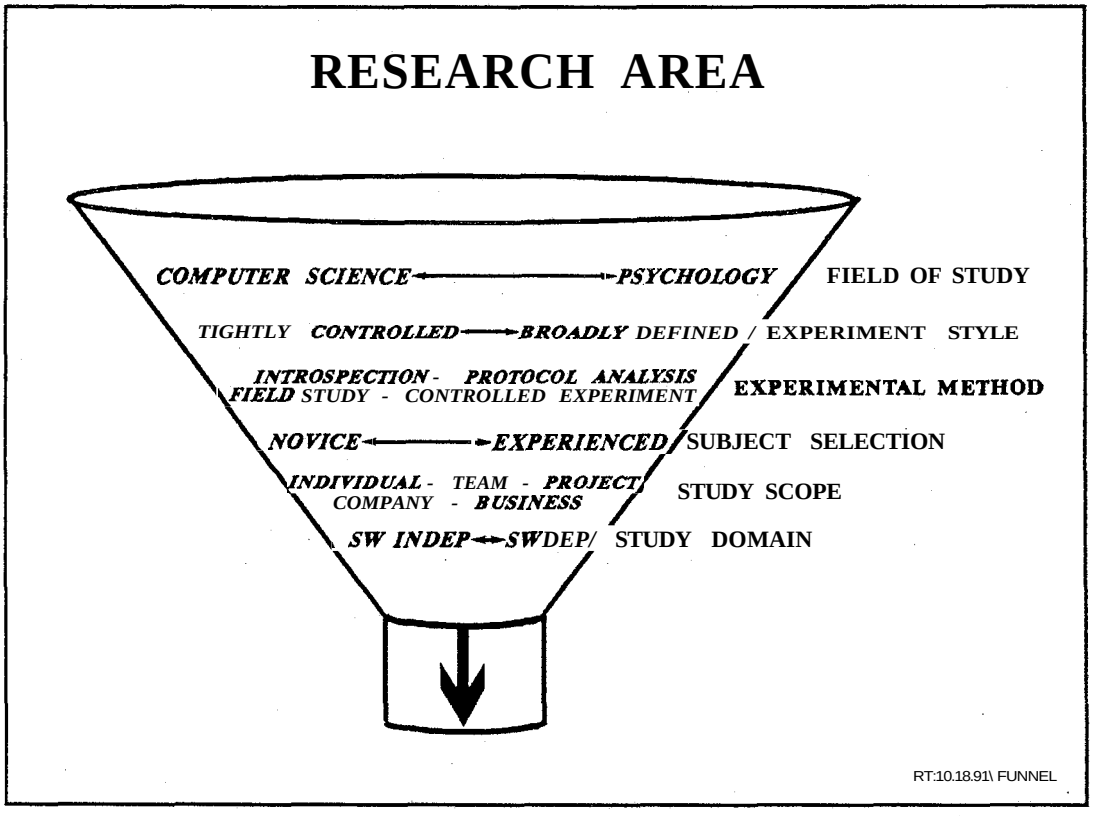

FIGURE 2.1 Research Area Funnel

CHAPTER 3

RESEARCH METHOD AND DESIGN

The techniques of cognitive psychological experimentation can help resolve specific issues in programming and explore the broader issues of programmer behavior.[9]

The research for this dissertation is divided into two phases. Phase 1 uses a qualitative technique. Phase 1 is intended to discover the competencies which may be related to exceptional performance. Phase 2 uses a quantitative approach to validate the results of Phase 1. Phase 2 provides the statistical basis for the preliminary conclusions reached in the Phase 1 work. Overall, the study is descriptive in nature. The dissertation's intent is to describe the difference in *competencies* exhibited by Exceptional and Non-exceptional engineers. Further, the dissertation creates a model capable of predicting exceptional ratings based upon discovered competencies. This predictive model differs in two important ways from prior research attempts. First, the predictive model will prove to be a complex rather than simple model of performance. Simple predictors will not work well. Second, the model will be heavily based on *behaviors* rather than on demographics.

The research method relies on a *triangulation* [Fiel86] technique to provide further validity of the results. This triangulation is illustrated in Figure 3.1. The basic two phase structure of the study triangulates the data by two basic approaches: qualitative and quantitative study. Further, the triangulation occurs in three major

[9]B. Shneiderman, "Exploratory Experiments in Programmer Behavior," *International Journal of Computer and Information Sciences*, Vol. 5, No. 2, 1976, p. 123.

forms: The study is triangulated by **subject** since Phase 1 surveys five R&D labs and Phase 2 studies nine labs. In each phase, subjects are selected from three different sites. The subjects in Phase 1 are working on three application types, while the subjects in Phase 2 are working on five application types. This diversity of subjects allows for confirmation of results by different subject types.

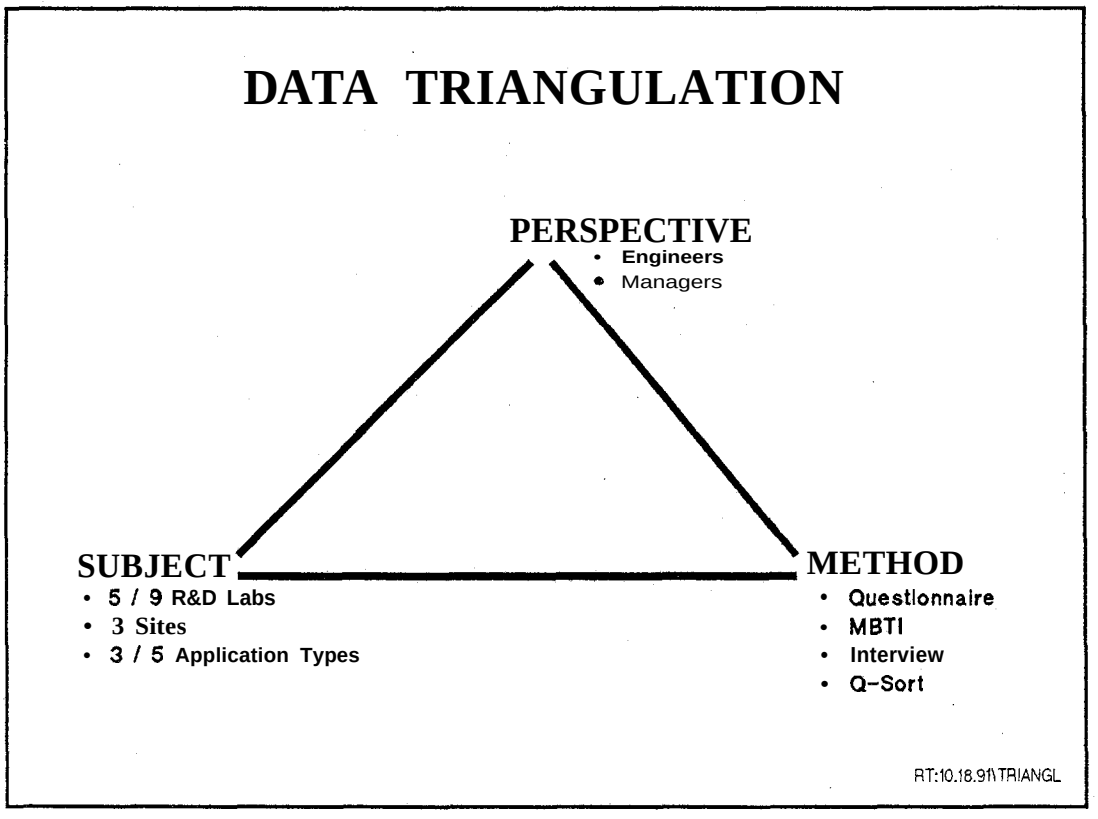

FIGURE 3.1 Research Data Triangulation

The research is triangulated by **method** since multiple research methods are working in concert to collect the data. The questionnaires provide one source of data and the results from the questionnaire can be compared with the results from the other instruments. The Q-Sort is a technique for ranking lists of behaviors and will be used

to provide another view of the competencies first discovered in the Critical Incident Interview. The MBTI used in Phase 1 provides another view of the psychological type of the individual in the study.

In Phase 1 each subject's behavior was studied to observe competencies. Each subject was also asked to directly identify additional competencies that they exhibit or see in others. Also, the manager of each lab was asked to identify the essential competencies for exceptional performance. This use of engineers and managers to identify competencies provides triangulation by **perspective.** Triangulation provides further validity of the results by confirming the result from different viewpoints. The following sections describe the use of this triangulation method in more detail.

3.1 Phase 1 Qualitative Data Phase

Behavioral psychology is the science of pulling habits out of rats.[10]

The research objectives for Phase 1 were to:

1. Determine differential competencies of exceptional versus non-exceptional software engineers, and
2. Determine the MBTI type profile of exceptional and non-exceptional software engineers.

This is the phase in which the research uncovers the set of competencies related to exceptional performance in software engineering. Phase 1 studies demographic information, psychological type, and actual job activity in order to uncover as much information as possible.

[10]Dr. Douglas Busch, from *Peter's Quotations - Ideas for our Time,* by L. J. Peters, William Morrow and Company, 1977:

3.1.1 Selection of Subjects

This study makes use of two matched subject pools in order to study the differences between exceptional and non-exceptional skills in software engineering. The study covers 10 subjects in each of the exceptional and non-exceptional pools. The subjects are matched by *time in current organization.* Thus if an exceptional engineer with four years in the current organization is identified, a second *non*-exceptional engineer with approximately four years experience in the same organization is added to the study. This approach controls for the effect of the organization on the individual's performance. These differences could be clearer goals and objectives, better management, or a particular product type. The study does not attempt to control any other factors since all are possible contributors to the exceptional performance under study.

All subjects are professional software development engineers from a major U. S. corporation with a minimum of two years of experience in developing software. Each subject has successfully completed a project released to the end user. From here on the population will be referred to as coming from *The Company.*

Subjects are selected by a supervisor selection process in which managers identify the top performers in their organization. In a letter to the manager, the researcher asked managers to identify an exceptional and average performing pair of individuals. The pair should each have been in the organization for approximately the same amount of time. The manager was also directed, "In order to ensure exclusiveness, no more than 5% of your lab should be considered as exceptional." The guideline ensured that the subject sample is indeed exceptional. Note that this subject selection technique introduces a series of biases on the part the manager doing the selection. Also, it

introduces the bias of the organization since ranking is an organizational process. This dissertation will study the individuals defined as exceptional by their managers. Hence this manager bias is an inherent part of the research design.

Once the subjects were nominated, each was contacted to determine if they were interested in participating since the study was to be strictly voluntary. Each subject received a letter that provided the assignment for each participant and asked each to complete the Biographical Questionnaire and the Myers-Briggs Type Indicator Test prior to the interview. The letter clearly indicated the voluntary nature of participating in the research. Each subject was also provided with a "standard" ethics disclosure as shown in Appendix B Phase 1 Standard Ethics Protocol which indicates the nature of the research and what will be done with the results. The ethics disclosure defined the subjects' rights and indicated their option to discontinue participation in the research at any time. Finally, the note established the date, time, and place for the interview, and indicated what the subject should expect.

3.1.2 Biographical Questionnaire

The study uses a questionnaire to gather descriptive information about the subject pool. The data are used to

1. Allow this data to be used by other researchers by adequately describing the subject pool to make comparisons possible.
2. Validate that
   - Subjects represent experienced rather than naive programmers.
   - Subject are a valid cross-section of developers covering different language use, target applications, and development environments.
3. Confirm or refute the following hypotheses relating to demographic and academic information.
   1. Exceptional engineers will make broader use of various tools and methods, especially acting as early adopters of new tools and techniques.

2. Demographic and academic factors will not emerge as predictors of performance.
3. Age and experience will correlate with performance, but will emerge as a threshold effect where it will not be significant beyond a certain point.

The questionnaire is provided in Appendix A Phase 1 Biographical Questionnaire and includes questions concerning education, on the job training, experience, languages used, and methods employed. The questionnaire was pretested many times in order to determine its appropriateness and ease of use. It was extensively modified as a result of these pretests. This questionnaire was completed by subjects prior to the opening of the *Critical Incident Interview.*

3.1.3 Critical Incident Interview

The Critical Incident Interview was conducted in a private room at the subject's worksite. Each interview was tape-recorded, and the recordings were transcribed for later use. The general outline of the interview is described in Appendix C Phase 1 Critical Incident Interview Outline. The interview was pretested to determine how well it would flow. The interviews began with casual conversation to set the subject at ease. The interviewer described the scope of the research and outlined the general flow of the interview. Once the subject was comfortable with the process, the tape recorder was started and the interview began. The interview followed the basic structure and practices defined in [Hori 89].

A typical interview began with an introduction similar to the following one taken from the transcript of the interview of subject #14.

> *What I'd like you to do is start off by thinking about a time which represents for you perhaps your personal best associated with software engineering in whatever form, so be it software development, software maintenance, testing, whatever it is, but a time at which you feel you were at your personal best, and when you've got one of those situations in mind, give me kind of a broad*

overview, a fifty word summary overview which is, how did you get involved in the situation, who were the other players, what was the nature of the task, and then we'll come back and we'll walk through it step by step in gory detail to find our exactly what you did in each case of that task.

The subject would then describe a particular incident and the interviewer would probe for clarification or increased depth of response. The interviewer used probes, open-ended questions, questions of clarification, and reflective listening to keep the participant on the subjects of interest. The only way that the interviewer tried to direct the conversation was to provide additional clarification or to move on to other topics.

The subject generally described two to three significant incidents in the course of one interview. When each incident was completed, the subject was asked to describe the critical skills or competencies which were essential to the successful completion of the task. At the end of the discussion of the subject's incidents, the subject was asked to describe the list of essential competencies for an exceptional software engineer. With reference to the data triangulation mentioned above, the incidents formed one set of data regarding competencies, the self-description of skills formed a second set, and the description of essential competencies for exceptional software engineers formed a third. All three sets of data are used to create the list of competencies which are subsequently validated in Phase 2.

Data analysis of the Critical Incident Interviews proceeded by the method of *Protocol Analysis* [McCr 88]. Here each written transcript was reviewed and highlighted to identify *tasks, incidents, competencies, self-described skills, and identified competencies for exceptional performance.* Each transcript was reviewed individually to identify consistent themes which could be generalized as competencies for that individual. After each transcript was reviewed individually, the set of transcripts was examined to identify competencies which appeared across multiple transcripts. These competencies

were generalized and reworded as required in order to emphasize the similarities. However, great care was taken to not over-generalize or distort the original meanings. A set of behaviors was identified based on all the transcripts and served as a detailed explanation of the intent of the competency. At this point, original transcript text was retained and attached to the competency as further definition. A final pass allowed for the combination of related competencies into a single competency.

All of the analysis to this point was done *blindly*. That is, the transcripts were tagged with an identification number and the researcher did not know the name of the subject. Further the researcher did not know if the transcripts were from an exceptional or non-exceptional subject since these results were left in a sealed envelope. At this point the identification of the exceptional performers was made.

The next step of the process was to count the number of subjects exhibiting an identified competency fromeach of the exceptional and non-exceptional groups. Those competencies exhibited by few subjects were dropped from further consideration. In general, at least three subjects had to identify a competency before it was retained. However, if one exceptional and one non-exceptional subject identified a competency, it was also retained.

The competencies identified from a subject's self-assessment of skills and from the subject's opinion of which competencies are related to exceptional performance were also identified and categorized. Those that appeared most often across transcripts were retained and used in Phase 2 research.

Finally, each manager who had provided subjects for the Phase 1 research was asked to identify the competencies used in selecting the exceptional subjects for study. The manager was asked to list the skills, knowledge, or attributes that differentiated

exceptional performers from non-exceptional performers in the study. The competencies identified most frequently across the five managers participating in the study were also used in the Phase 2 study.

3.1.4 Phase 1 Method Summary

Figure 3.2 presents a summary of the Phase 1 research method. Phase 1 is designed to identify the competencies of software engineers. Further, it will test the biographical questionnaire. The Phase 1 subjects are drawn from five Company divisions on three sites and are active in the creation of three types of software applications. Phase 1 studies 10 exceptional and 10 non-exceptional engineers. Phase 1 uses a biographical questionnaire, the MBTI, and the critical incident interview as research instruments. Although tests for difference are conducted in Phase 1, the true value of Phase 1 lies in the identification of the competencies.

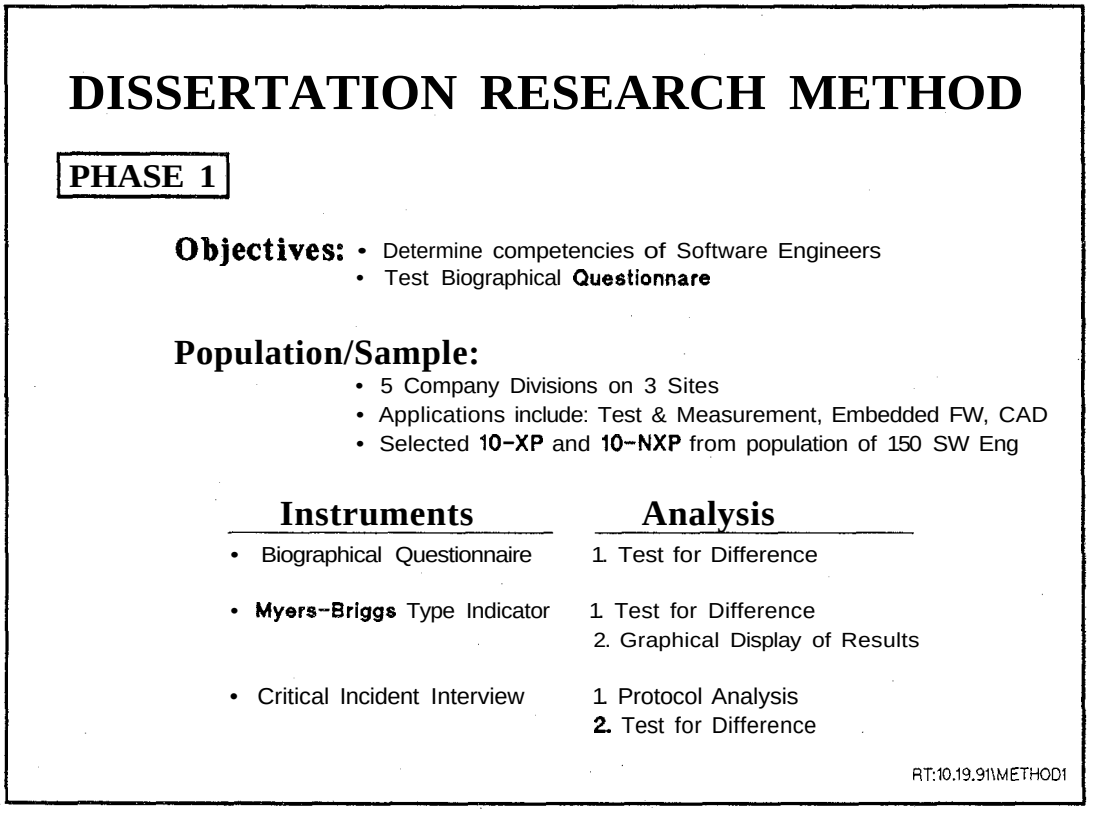

FIGURE 3.2 Dissertation Research Method (Phase 1)

3.2 Phase 2 Quantitative Data Phase

The research objectives for Phase 2 were to:

1. Validate the competencies identified in the Phase 1 research, and
2. Determine if a simple predictor of performance exists.

This phase consists of an attempt to validate that the competencies identified in Phase 1 are indeed significant and actually relate to performance. In particular, the validation determines whether competencies hypothesized to be differential after

Phase 1, are in fact differential. Further, the data collected in this phase are used to build a predictive model that uses the identified competencies to predict whether a particular engineer will be ranked as exceptional or non-exceptional.

3.2.1 Selection of Subjects

For Phase 2, the objective is to validate the results of Phase 1 against a broader population. Thus the research pool was expanded both in quantity and diversity. Matching for time in the organization was no longer required since the breadth of participants was expected to make differences in experience in the organization no longer relevant. In addition, the "exclusivity" of the exceptional label was relaxed to allow for a more even mix of exceptional and non-exceptional engineers in the study. This is a conservative approach in that allowing more subjects to be defined as exceptional merely increases the risk that a competency will not be identified as differential. However, the competencies that are shown to be differential are much more likely to be so. The mix was also changed to allow for a sufficiently large number of exceptional subjects to power the required statistics.

As in Phase 1, all subjects are Company software engineers. All engineers were allowed to participate regardless of length of service. There was not the 2 year minimum experience criterion as in Phase 1. Managers were invited to have their labs participate in the study via letter. The letter asked the participating lab managers to distribute surveys to their entire lab on a differential basis. Seventy percent of the surveys would go to non-exceptional performers and 30% of the surveys would go to exceptional performers. The determination of exceptional versus non-exceptional was again made by the managers according to the criteria in a follow-on letter. The 30/70 split was to be used as a guideline. However, the managers were to use judgement and were allowed

to distribute exceptional surveys to more than 30% of their lab. This met with the spirit of surveying based on performance. The managers were instructed to keep the differential nature of the survey confidential.

When a manager elected to allow their lab to participate in the study, he or she received a set of survey packets to distribute to their engineers. Extra surveys of both the exceptional and non-exceptional variety were distributed so that the manager could slightly skew the population based on the performance of the individuals in their organization. Each packet contained a letter of instruction that outlined the assignment and clearly indicated the voluntary nature of the survey. Each packet included a Biographical Questionnaire and a set of Q-Sort cards. The packet included a pre-addressed return envelope for returning the completed survey. This method kept the results *blind* in that the researcher never knew the names of study participants or their corresponding rating.

3.2.2 **Biographical Questionnaire**

The Biographical Questionnaire used in Phase 2 is presented in Appendix G Phase 2 Biographical Questionnaire. It is identical to the Phase 1 questionnaire with the following exceptions.

1. The *Interview Date* and *Subject ID Number* were removed. In this phase of the study, there were not multiple study products to relate as there were in Phase 1 (MBTI, Questionnaire, and Interview Transcript) so these were no longer necessary.
2. *Company Division* was added to allow for reporting of results by division in order to track response rates. Also, some managers agreed to participate on the condition that they were able to see composite results for their own labs relative to the full population. These separate results will not be presented in this dissertation.
3. The degree of *Engineer was* dropped as a degree choice. Few engineers hold this degree as it is only offered by two universities in the United States. In addition, it caused some confusion with a major in engineering.

4. The text was changed to show that the only number used for *Training Hours* was the grand total. Some respondents had trouble classifying their training, and the final number is all that is used in the study.
5. Further clarification is provided in the description for *Years of Experience* in order to eliminate confusion.
6. The **MED** answer was removed from the *Language* question. In Phase 1, many subjects exhibited a "central tendency" and answered **MED** for all languages. This was dropped in an attempt to increase variance. Also, the directions were improved to better clarify what languages qualified. In particular, the use of languages was scoped to those used professionally.
7. The entire *Method/Tool* section was dropped. This question proved difficult for subjects to answer as there was considerable confusion about what *was* a method or tool. Also, the results from Phase 1 were not significant.
8. A *Results of Sorting Exercise* section was added to capture the results of the Q-Sort activity.

The data collected via the biographical questionnaire was analyzed for difference using statistical tests. The tests used in this analysis come from the area of *Analysis of Variance.* The *t-test, Fisher's Exact Test,* and *Chi Square Test* are used to determine if the mean value of variables are significantly different. The Chi Square test is used for distinguishing nominal values on a differential basis. For example, if we wished to study the factor of *Highest Degree Held* for exceptional and non-exceptional engineers, the Chi Square test is used since *Highest Degree Held* takes on multiple discrete values. If the variable takes on only two values then *Fisher's Exact Test* is used. This occurs for the study of the effect of *Gender* on exceptional performance. Finally, variables which take on continuous values are studied using the *t-test.* For example, studying the effect of *Years of Experience* would require the use of the *t-test.* In all cases, a significance level of 0.05 will be used which indicates that there is only a one in twenty possibility that the observed difference does not represent the actual population. These tests of difference will be used to further validate the results of the discriminant analysis and offer further insight into the data.

3.2.3 Q-Sort

The common denominator of success ... lies in the fact that he or she formed the habit of doing things that failures do not like to do.[11]

The Q-Sort activity was to be completed as described in the letter in Appendix F Survey Instructions for Participants. According to Q Methodology, a Q-Sort task is normally completed by a subject with the help of the researcher. This simplified approach allowed the subjects to complete the assignment on their own.

Each subject received a set of *Competency Cards* with one competency listed on each of 38 3"x5" index cards as shown in Appendix H Phase 2 Competency Statements . The *direct manipulation* attribute of the Q Methodology allows the subject to sort a much longer list of items. A set of *Pile Marker Cards* is also included in order to prompt the subject to create the correct number of piles and to include the correct number of cards in each pile. Further, the Pile Marker Cards include prompts to remind the subject of the definition of the continuum across which the competencies are sorted.

It is important to emphasize that the criterion for sorting the competency cards is the subject's own behavior. There are a multitude of possible criteria for sorting these competencies *(most important, best for achieving a high rank, best for creating defect-free code, ...)* but to be compatible with Phase 1 results, only behavior-based sorting makes sense. Recall that the original source of the competencies was the transcripts describing the Phase 1 subject's behavior. Hence from a validation point of view, actual behavior must be the criterion. Also, since the study is attempting to predict an individual's classification (exceptional or non-exceptional,) behavior is the only observable criterion upon which to base this judgement.

---

[11] Albert E. N. Gray, "The Common Denominator of Success," *Insurance Sales,* April, 1989.

3.2.4 Phase 2 Method Summary

Figure 3.3 presents a summary of the Phase 2 research method. The objectives of Phase 2 are to validate the competencies identified in Phase 1 and determine which are differentially related to software engineer performance. Phase 2 also provides a predictor of performance. The sample for Phase 2 is drawn from nine Company divisions whose engineers participate in the development of five application types. The total software engineering population in these nine divisions is 275 engineers. Phase 2 uses a biographical questionnaire to collect demographic information about the subject pool. Phase 2 uses the Q-Sort as a means of rank ordering the identified competencies based on the degree to which they match the subject's behavior.

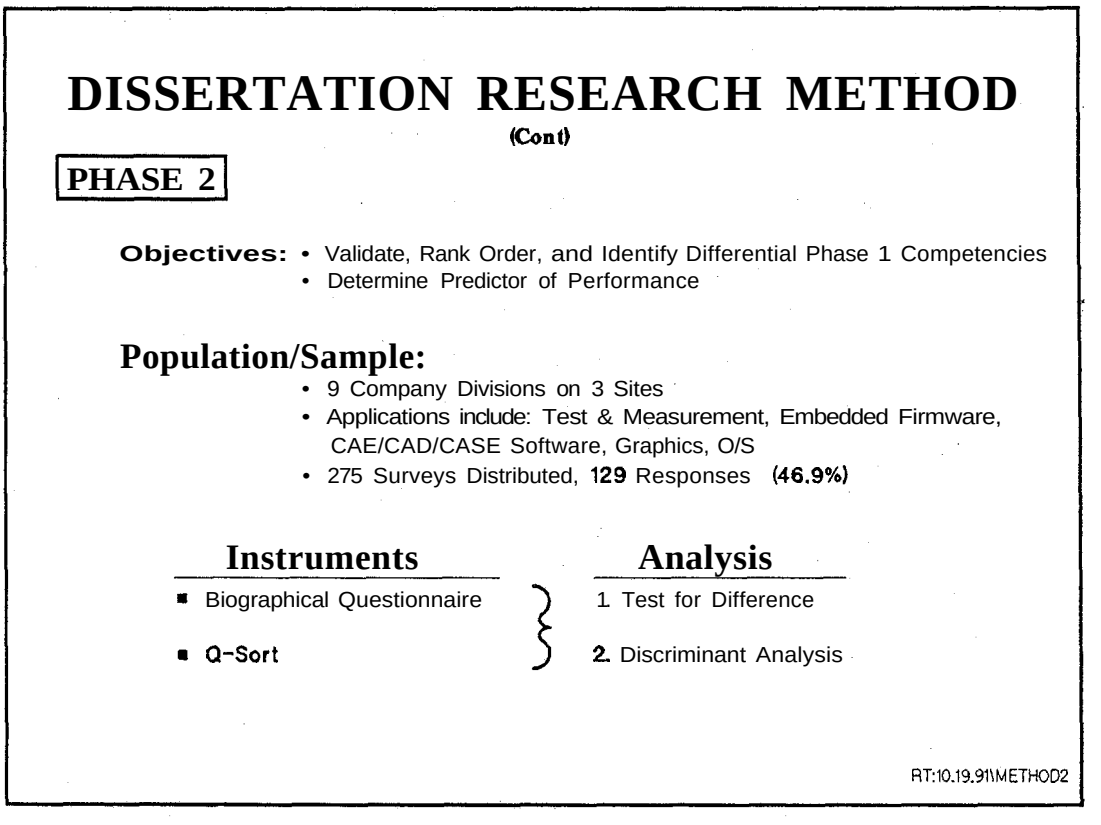

FIGURE 3.3 Dissertation Research Method (Phase 2)

CHAPTER 4

DATA PRESENTATION

What we are witnessing now is a return to the descriptive statistics of the nineteenth century, completing unfinished business left over from that era. ... In descriptive statistics, you see things, but you cannot test them in a formal way. In mathematical statistics, you may test for something preconceived, but you may overlook something you hadn't built into the test.[12]

This chapter presents the data collected in both Phase 1 and Phase 2 of the study. This report is divided into each of the two phases. In each phase, the study sample is described. The data for each phase are reported first as they relate to the sample as a whole. Finally, the data from each phase are reported on a differential basis to determine if statistically significant differences exist between the responses from exceptional and non-exceptional software engineers. All data reported in this dissertation were analyzed using SPSS/PC [Frud 87].

The data presented in this chapter represent the triangulation discussed earlier. The data are triangulated in time since the Phase 2 study was conducted approximately nine months later than the Phase 1 study. The data support the notion that time did not affect the results and the Phase 2 results are consistent with the Phase 1 results. The data are also triangulated by study subject. In Phase 1 the interviewees give two separate responses. They discuss their behavior, which forms one observation, and they report the competencies they consider important, which forms a second observation. The managers of the groups studied also report the competencies they

[12]Dr. Peter Huber quoted in: *For All Practical Purposes: Introduction to Contemporary Mathematics,* W. H. Freeman and Company, New York, p. 169, 1991.

consider important. This forms the third observation. Finally, the study is triangulated by method. In Phase 1 an interview technique is employed. In Phase 2 a survey technique is used. The power of these multiple triangulations is that the results are shown to be consistent across time, responses, and methods.

4.1 Phase 1 Qualitative Data Collection

Phase 1 represented the qualitative portion of the research study in which the research aimed to capture the relevant competencies of software engineers. The study covered ten exceptional and ten non-exceptional engineers from five R&D labs across three Company sites. These organizations produce applications software and embedded microprocessor firmware. Table 4.1 summarizes the population from which the study participants were drawn. The *#Engineers* represents the total of engineers of all disciplines in the total population. The *#SW Engineers* indicates the number of engineers that the managers describe as software engineers. The *#StudyParticipants* indicates how many engineers from the total population participated in the Phase 1 study. The number is even since it represents the matched set of exceptional and non-exceptional engineers paired for time in the organization. Finally, the *% of ExceptionalSW Engineers* is the ratio of *#ExceptionalSW Engineers Studied* to the *#SW Engineers* in the population.

The total number of engineers in the participating divisions was 252. These organizations were selected for study based upon their willingness to participate, and convenient geographical proximity. The number of Software Engineers in these divisions was 150, or 60%. If we assume that the Phase 1 research sampled the best 10 Software Engineers out of this population, we find that the subjects represented the top 6.7% of the software engineers in these labs. This matches well with the assignment

TABLE 4.1 Population Summary (Phase 1)

(n=20)

POPULATION SUMMARY	TOTAL
#Engineers	252
#SW Engineers	150
#Study Participants	20
#Exceptional SW Engineers Studied	10
% of Exceptional SW Engineers	6.7%

for the lab managers to supply as exceptional the top 5% of their labs.

4.1.1 Biographical Questionnaire

The Biographical Questionnaire shown in Appendix 3.1.2 Biographical Questionnaire was used to collect descriptive information for the population under study. Tables 4.2 through 4.8 provide this information for the population at large.

Fifteen subjects representing 75% of the sample were male. Five subjects representing 25% of the sample were female. The 3 to 1 ratio of males to females is consistent with published reports [Pear 90] that women constitute only 30% of the employed computer scientists.

The mean age of the subjects in the Phase 1 sample (n = 20) was 33.45 years. The values ranged from 27 to 42 years. The Phase 1 subjects are generally older and there is a broad range in their ages.

The subject pool is well educated as shown in Table 4.2. The mean number of degrees held is 1.6. The number of degrees held ranges from one to three. Half of

the subjects (n= 10) had completed two degrees. Few subjects have partially completed degrees as shown by the mean of only 0.15 partially completed degrees per subject. Fully 85% (n= 17) of the subjects are not currently part way through a degree program.

TABLE 4.2 Subject Education (Phase 1)

(n=20)

	FREQUENCY	% of TOTAL	CUMM %
# DEGREES COMPLETED			
1	9	45%	45%
2	10	50%	95%
3	1	5%	100%
# DEGREES PARTIALLY COMPLETED			
0	17	85%	85%
1	3	15%	100%

Another way to look at education is by the *Highest Degree Held* shown in Table 4.3. Sixty five percent of the subjects (n= 13) completed a Bachelors degree as the highest degree. Thirty percent (n=6) have completed a Masters degree. Only one subject held a Ph. D. All subjects held at least a Bachelors degree.

TABLE 4.3 Highest Degree Held (Phase 1)

(n=20)

HIGHEST DEGREE HELD	FREQUENCY	% OF TOTAL	CUMM %
Bachelors	13	65%	65%
Masters	6	30%	95%
PhD	1	5%	100%

There is a preponderance of Electrical Engineering degrees held as shown in Table 4.4. Fully half of the engineers (n= 10) in the Phase 1 study held such a degree. This related to three factors about the subjects in this study.

TABLE 4.4 Majors for Degrees Held (Phase 1)

(n=20)

MAJORS FOR DEGREES HELD	FREQUENCY	% of SUBJECTS[1]
Electrical Engineering	10	50%
Computer Science	7	35%
Computer Engineering	3	15%
Mechanical Engineering	3	15%
Math	2	10%
Physics	1	5%
Chemistry	1	5%
Molecular Biology	1	5%

1. Total will exceed 100% since some subjects held multiple degrees.

1. The Company has historically been an Electrical Engineering company. An employer will most often hire workers similar to those already on the job.
2. Most of the subjects for the Phase 1 study are older and more experienced. AComputer Science degree is newer and tends to be held by younger workers.
3. Many of the divisions surveyed in Phase 1 are developing firmware or electronic test software. These applications are much closer to the hardware than higher level applications and often attract and require engineers who can easily move between the electrical engineering and software disciplines.

For these reasons this skew toward Electrical engineering graduates is not a concern in this study.

The mean number of training hours completed in the past two years by study participants (n=20) is 117.70 hours. Completed training ranged from zero to 306 contact hours. This training included college classes, corporate classes, seminars, self-study, and any other area considered significant to the subject. For formal classes

contact hours is the number of hours spent in the classroom. For self-study, contact hours is the number of hours spent in self-training. The subjects reported a wide range of training hours and the mean indicates that engineers spend a significant amount of time each year in training. The 117 training contact hours over two years reported corresponds to almost a week and a half of training per subject per year.

Table 4.5 illustrates the number of Languages and Methods applied with Low, Medium, and High skill. Low skill was defined as "novice user, little experience." Medium skill was defined as "comfortable, some experience, familiar." High skill was defined as "expert, lots of experience, well versed." For languages, the ranges for each category are comparable. There is some tendency to know more languages with increasing skill level as shown by the mean value of languages known increasing from 1.1 at Low Skill to 2.45 at High Skill.

TABLE 4.5 Language/Method Usage (Phase 1)

(n = 20)

LANGUAGES	MEAN	RANGE
High Skill	2.45	0-6
Medium Skill	1.80	0-5
Low Skill	1.10	0-4
METHODS/TOOLS		
High Skill	1.05	0-4
Medium Skill	1.00	0-5
Low Skill	0.35	0-2

The Methods/Tools questions showed mixed results. The skill definitions were the same as for Languages. The prompt provided was "describe the software engineering methods and tools that you use now or in the past in your job. For example,

SA/SD, Yourdon, Jackson, McCabe, Halstead, ..., but be sure to include anything you feel is relevant." In general, subjects were confused about exactly what was a tool or method. Some subjects responded with very high level tools like *structured design* and *object-orienteddesign* while others responded with very low level tools like *make* and *awk*. Due to the vagueness of the question, the range of the results, and the similarity of the responses at the Medium and High level, the results are inconclusive.

Table 4.6 defines the work experience of the Phase 1 subjects. The table is divided into quadrants. The first quadrant reports the experience in years for subjects working in Software Engineering at The Company. With a mean of 7 years, this constitutes the greatest amount of experience for the subject pool. The range of 2 to 15 years substantiates that no subject had less than 2 years of experience and shows that very experienced engineers are also in the study. The other quadrants show various combinations of Software Engineering versus Non-Software Engineering work at The Company and other companies.

TABLE 4.6 Work Experience (Phase 1)

# *YEARS EXPERIENCE*	SOFTWARE ENGINEERING			NOT SOFTWARE ENGINEERING		
	MEAN	RANGE	n[1]	MEAN	RANGE	n[1]
Company	7.03	2-15	20	1.75	0-6.5	8
NOT Company	0.68	0-7	4	0.65	0-10	3

1. Indicates the number of subjects with non-zero values. The mean and range still describe the full sample of n = 20.

The least experienced engineer studied had 5 years of experience spread across all four quadrants of Table 4.6. The mean years of total experience across all four

quadrants of Table 4.6 is 10 years. The maximum total years worked in the sample is 20 years. This again substantiates that the subject pool for the Phase 1 research is indeed experienced.

All of the preceding data were analyzed for statistical significance when studied on a differential basis. That is, all of the data were split between *Exceptional* and *Non-Exceptional* subjects and compared. The *Fisher's Exact Test* is used to compare nominal variables with only two values *(e.g. gender.)* The *t-test* is used to compare the means of ordinal values *(e.g. training hours.)* All test were performed looking for a significance of 0.05 or better. Since this was such a small sample, no significant differences were expected. The results of these tests are summarized in Table 4.7.

Years at Company in Software are significantly related to *Exceptional Performance* with the 2-tail *t-test* calculated value of -3.21 and a calculated probability of 0.007. The significance of experience indicates that although the design of the study matched engineers for length of service in their current organization, the research design did not match them for total experience. Hence the study did not remove experience as a determining factor for exceptional performance. The failure to remove experience as a determining variable could be due to a number of factors. The subjects were to be selected on a matched basis for time spent in the organization. This definition of matched experience did not control for experience of each subject prior to joining the organization. One subject could, for example, have worked many more years developing software in another Company organization before joining the current one. Further, one subject may have been in the organization for the same amount of time, but spent much of that time in a role other than software engineering. At any rate, years spent in Software Engineering at The Company emerges as a significant differentiator in the Phase 1 study.

TABLE 4.7 Tests for Difference (Phase 1)

(n=20)

VARIABLE	TEST	CALCULATED TEST VALUE	CALCULATED SIGNIFICANCE LEVEL[1] *(2 Tail)*
GENDER	Fisher's Exact		1.000
AGE	T	-0.88	0.388
#Degrees Completed	T	-1.55	0.139
#Degrees Partially Completed	T	0.60	0.556
Highest Degree Held	T	-1.55	0.139
Training Hours	T	-0.40	0.691
Languages - High Skill	T	-2.01	0.059
Languages - Medium Skill	T	0.00	1.000
Languages - Low Skill	T	0.95	0.355
Languages - Total	T	-0.55	0.586
Methods - High Skill	T	-1.67	0.112
Methods - Medium Skill	T	-0.73	0.476
Methods - Low Skill	T	0.33	0.749
Methods - Total	T	-1.53	0.143
Years at Company in Software	T	-3.21	0.007 **
Years at Company NOT in Software	T	0.54	0.598
Years NOT at Company in Software	T	1.46	0.176
Years NOT at Company NOT in Software	T	-[2]	-
Total Years Worked	T	-2.09	0.056

1. Differences are considered statistically significant when the calculated significance is less than 0.05. These are marked with '**'.
2. Not calculated since one or more samples had no variance.

Table 4.8 presents the information about *Years at Company in Software* on a differential basis. The mean time at the company for exceptional performers is

significantly higher at nine years than for non-exceptional performers at 5 years. The range is much broader for exceptional engineers with the longest length of service twice that for the longest length of service of a non-exceptional engineer.

TABLE 4.8 Years at Company in Software - Differential (Phase 1)

(n=20)

Years at Company in Software	MEAN	STD DEV	RANGE
Exceptional	9.05	3.59	4-15
Non-Exceptional	5.00	1.75	2-7.5

This section has described the demographic characteristics of the sample studied. It indicated that the sample represented an experienced and well educated pool of software engineers. With the exception of the experience variable, no demographic data were significantly different between the exceptional and non-exceptional sub-samples. The lack of other statistically significant differences indicates experimental control of the other variables or speaks to the uniformity of the sample.

4.1.2 Myers-Briggs Type Indicator

All 20 Phase 1 subjects completed the *Myers-Briggs Type Indicator* test. The results are summarized on a differential basis in Figure 4.1. The figure shows the distribution of all the Phase 1 study participants according to one of 16 personality types. The figure is organized in such a way that each cell is located next to cells which are different in only one of the four MBTI characteristics. The rows and columns are labelled with the attributes which are valid in that row or column.

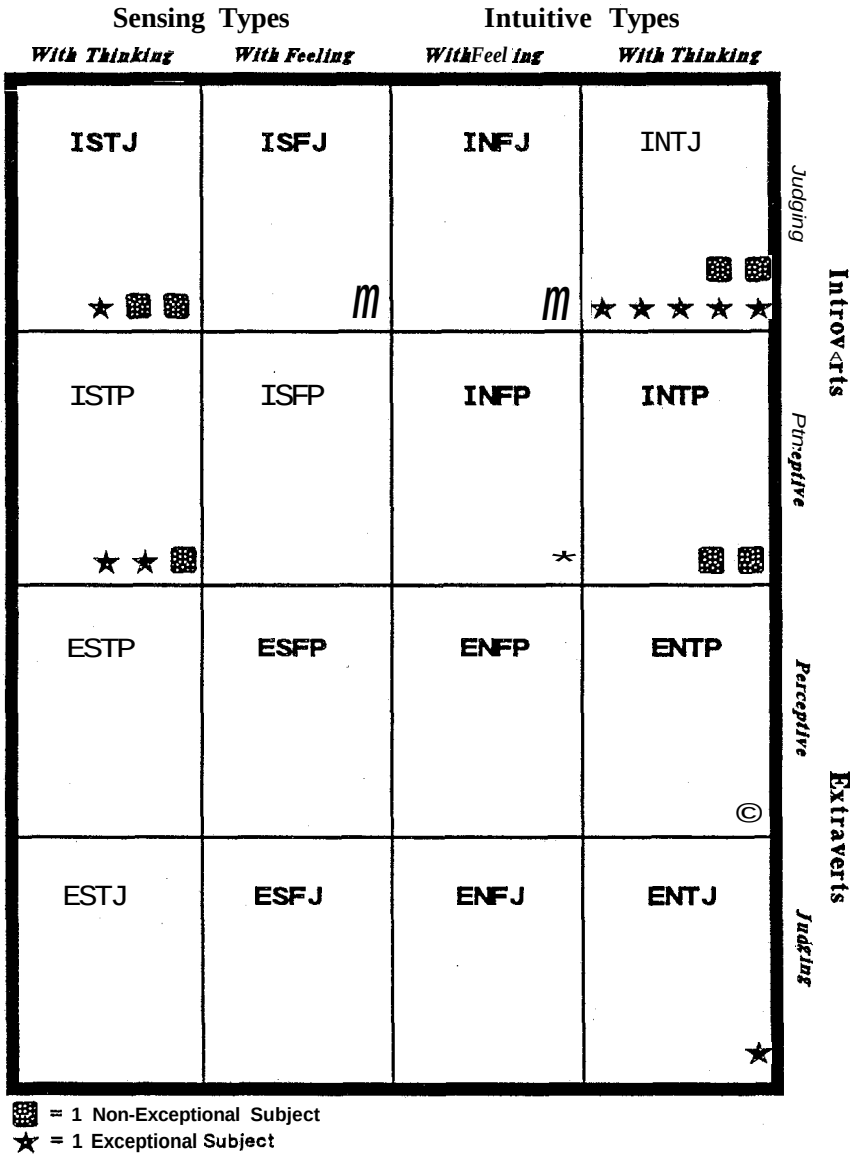

FIGURE 4.1 MBTI Results (Phase 1)

Looking first at the overall distribution of subjects, we see that there is a preponderance of the *Introvert* type amongst the entire subject pool. Eighteen out of

twenty exhibit this type. Prior studies of engineers also found a significant tendency toward the *Introvert* types. Also, the *Introvert* tendency is consistent with the Kagan-Douthat study of students learning FORTRAN [Kaga 85] which found a tendency toward introversion in higher performing programming students. Further, there is a tendency toward the *Thinking* type with 17 out of 20 exhibiting it. Prior studies also found the tendency toward *Thinking* types in engineers. The *Thinking* tendency indicates that engineers rely on logic and analysis of cause and effect in their decision making, rather than emphasizing the effect of decisions on people and their feelings. In fact, computer specialists exhibit the thinking preference 67% of the time in broader studies [Myer 85].

The most frequent classification for exceptional performers is the **INTJ** *(Introvert, Intuitive, Thinking, Judging)* type. Amongst male college graduates, the **INTJ** type occurs only 10% of the time [Myer 85]. Hence these exceptional engineers differ from the population at large. The pattern displayed in Figure 4.1 is typical of R&D organizations.

The differential continuous scores are captured in Table 4.9. Scores below 100 favor the first preference listed in the pair while scores above 100 favor the second preference listed. Statistical analysis of these scores via the *t-test* revealed that there were no significant differences between the scores. This indicates that personality type is not a good predictor of performance. This could also be a result of the small sample size. Since the *MBTI* test is time consuming to complete, and somewhat expensive to distribute and score, and since the results from Phase 1 were inconclusive, the test is not used in Phase 2.

TABLE 4.9 MBTI Differential Scores (Phase 1)

(n=20)

	Exceptional	Non-Exceptional	T-Test Calculated Value	T-Test Significance Level[1] *(2 Tail)*
Extrovert - Introvert	128.6	117.0	-1.45	0.165
Sensing - Intuitive	109.0	100.8	-0.76	0.456
Thinking - Feeling	79.4	77.4	-0.20	0.841
Judging - Perceiving	90.8	96.4	0.43	0.673

1. Differences are considered statistically significant when the calculated significance is less than 0.05.

4.1.3 Critical Incident Interviews

Even though I feel like I'm good at it, its always been very mysterious to me, I mean, you start with this gigantic mass of information, and you have to sift through it fairly rapidly to take out the pieces that you need, and even though I feel like I do it fairly well, I've never known why, you know, is it brain chemistry, or is it something that somewhere because of some fourth grade teacher I picked it up, and it's always been just kind of this innate thing that I've felt comfortable doing. And so I do think about this a good bit at times and I really don't know whether it was because of genes that I got or because of some chemical imbalance in my brain or because some teacher had a real effect on me at some point that I don't even remember now.[13]

This section discusses the results gleaned from the analysis of 20 interview transcripts. It describes the method employed and discusses the decisions made during the identification and classification processes. This section describes competencies. Competencies are the skills, techniques, and attributes of job performance. This section describes the three different sources of competencies: subjects describing their own behavior, subjects reporting the competencies they think are related to exceptional performance, and managers describing the competencies of the subjects they selected as exceptional. Finally, the section describes how these competencies from three sources were merged into a single list of 38 competencies.

Critical Incident Interviews are structured interviews designed to probe a subject's behaviors to isolate the skills essential to success on the job [Flan 54]. The procedure for completing these interviews is quite formal, and requires certain key behaviors on the part of the interviewer. The interviewer must not lead the interview in any particular direction. Rather, the unfolding of the subject's story should itself guide the interview

[13] From the transcript of Subject #20.

process. As a subject describes a particular incident, the interviewer should only probe for more detail or to clarify certain points. The actual direction of the interview and the selection of which topics are important must be left to the subject.

The 20 *Critical Incident Interviews* yielded a massive amount of data. Each interview lasted, on average, two hours. Hence the full set of data consists of 40 hours of taped interviews. The transcription of these tapes produced almost 20,000 lines of text with over 200,000 words just for the subject responses. These figures do not include the interviewers prologue or any questions or probes.

4.1.3.1 Identified Competencies

This section describes the set of competencies identified through the analysis of the transcripts while focusing on the behaviors subjects described. The essential process is to extract important competencies by converting the subject's description of a situation into an observation.

These competencies were derived from the analysis of the interview transcripts via Protocol Analysis [McCr 88]. Each transcript was reviewed and each observation was marked to identify incidents and competencies. An incident was a particular event or project which the subject described in detail in response to the prompt: *Describe a time during your career as a Software Engineer which, for you, was a personal best.* Generally the interviews were able to cover two or three incidents in two hours.

Competencies are identified by marking the skills, knowledge, or personal attributes the subject alluded to while describing their role in the incident. Each observation of one of these skills, knowledge, or attributes is marked with its location in the transcript. A key phrase, usually in the subject's own words, is used to record

the observation. Since the subject was only given general directions and not prompted to focus on a particular area, the skills, knowledge, and attributes discussed were only those the subject chose to express.

Each interview transcript was studied to identify all the possible competencies exhibited by the subject. As these competency "fragments" were collected within a transcript, they were organized by related topics. Whenever possible, multiple observations were collapsed into a single competency with multiple examples. Care was taken not to over-generalize a concept in order to collapse these sets of fragments. Each transcript analysis resulted in a set of exhibited competencies for that individual.

When all 20 of the transcripts had been analyzed in this manner, the researcher collected the identified competencies across the interviews and created a single list for all 20 subjects. Common competencies identified across transcripts were combined into a single competency statement. The observations from each transcript were merged into a single list whenever competencies were coalesced. These observations were still marked with their location in the source transcript. Each grouping of competency fragments was given a name to help identify and summarize the competency observed. The final result of this process was a list of 53 competencies requiring 42 single spaced pages of description. When this single list was completed, it was again analyzed in an attempt to further reduce the number of competencies by identifying concepts which could be merged. This produced a final list of 38 competencies. This is coincidentally the number of competencies used later in the Q-Sort process. However, this relationship is purely accidental.

The process of selecting which of these competencies to retain for further consideration was both subjective and objective. It was subjective in the sense that the competency must be clear and narrowly enough defined to express a specific knowledge,

skill, or attribute. The process was objective in that the competency had to be expressed by at least one exceptional and one non-exceptional subject, **or** by three or more different subjects before it would be included for further consideration. The next pages describe the 38 identified competencies in more depth. **The** pages also describe the process of refining this list to the final list of 27 competencies retained in this step. The competencies are discussed from the most frequently exhibited to the least frequently exhibited in the interview transcripts. The final 27 competencies are described in more depth with key behaviors and illustrative examples in Appendix [XREF].

4.1.3.1.1 Retained Identified Competencies

The competency *Uses Decomposition Design Techniques* was exhibited by 16 subjects. The major themes of this competency include top down, modular designs with a prioritized set of alternatives. They emphasize decomposition and keeping specifications in sync with implementations. These subjects see the value in design itself. As one subject put it, "Just doing a good design to start with because [it makes] the rest of it ... a piece of cake." Interestingly, some subjects describe the lack of design or formal method for design. Hence this competency will make an interesting scale from those who value and do design to those who don't value and hence ignore this step.

A collection of different competencies were merged to form the *Team Oriented* competency. The competency of *Interacts with Other Engineers* forms the basis for the Team Oriented competency. In general, the interaction with other engineers demonstrated a preference for promoting the team results over individual results. The subjects described interactions with other engineers that stressed cooperation,

peacemaking, and constant communication. The subjects described team dynamics which lead to high synergy and significant team results. The category of *Influencing Others* was also combined to form the Team Oriented competency. Subjects described techniques they would use to convince another engineer of their point of view. Generally they stressed the non-confrontive interaction style they used to achieve their results. The *Extensions of Another's Ideas and/or Efforts* competency was also merged in Team Oriented. This category described how subjects would leverage off another person's ideas to create a synergistic, improved result. Finally, the *Prefers to be Left Alone* category was also merged with Team Oriented. This area is really the antithesis of Team Oriented and will be measured by a low value on Team Oriented. Constructed as shown, the Team Oriented competency appeared in 14 of the interview transcripts.

The *Use ofPrototypes* competency relates to how the subjects used prototypes in design. In particular, many subjects were clear in their own minds that prototypes are used primarily to assess critical performance metrics prior to actual implementation. They attempt to not allow their prototypes to be converted directly into a final product. The do, however, make good use of prototypes to make important design decisions. One subject was so determined that his prototype not be turned into a product that "I explicitly did it in *LISP* so that I could throw it away and start over." As another subject notes, "I felt very strongly that what I had done was not meant to be evolved into the real product. ... They took the prototype that I had worked on and just basically turned it into the real thing, and that piece of code, that one segment is the biggest nightmare of this whole project and one of the reasons that we kept having major schedule problems on it." One other subject notes the value of the prototype by saying, "By doing a little bit of prototyping, it really drove some decisions there." Fourteen subjects alluded to the use of prototyping in their interviews.

Thirteen subjects *Write/AutomateTests in Parallel with Coding.* This competency expressed two important attributes. First, subjects created test suites as the same time they developed their code. This allowed them to complete the bulk of their testing and verify their modules at the same time that they completed their implementation. As one engineer puts it, "I'm one that will code a piece and then will probably take the time to write some kind of shell around to test." The second attribute of these tests was that they were automated to allow regression testing of the code as changes were made. Another engineer says, "We already had set up about 500 automated tests in our test suite, and we were adding to that as we went along, so as a new area would be developed we'd try and write a new test for it so that it could be automated. Engineers used these tests to ensure reliability at the module level as they developed their products.

Many subjects described the level of *Unique Knowledge* they had coming into a job and indicated that this knowledge was essential to their success. The examples here tended to be along the lines of having recently taken a particular class that was relevant to the subject or having pursued the knowledge as a personal area of interest. In certain cases it related to the specific domain knowledge of the problem to be solved. In other cases it related to the skills or tools knowledge required to get the job done. Thirteen of the 20 subjects related information substantiating this as an important competency. It was retained for further analysis without further combining with other competencies.

Twelve subjects expressed a tendency to take *Pride in Quality and Productivity.* The subjects felt that the very essence of good engineering was in the quality of the resulting product and the personal productivity with which it was accomplished. In this competency the pride is paramount. This is how these individuals want to be

measured. The quality and personal productivity is their personal badge of honor. The subjects seem to link quality with productivity often, which is why they appear in the same competency. One subject particularly enjoyed the challenge of writing code to be delivered in Read Only Memory (ROM.) Since this make software very expensive to change, the whole attitude about defects is different. The subject notes, "The mind set that most people had was that you developed this code and then burn it in ROM and you don't ever get to change it. ... We would never have admitted to ourselves that it was okay to have defects in the software."

Twelve subjects exhibited the competency of *Obtains the Necessary Training.* Here subjects seek the required documentation for their current project, take classes helpful in completing the assignment, keep current by reading trade or technical journals, and improve skills and awareness by attending conferences. As one subject states, "I had to gain more depth in the operating system structure, in constructs and facilities that are available with Windows(R), Presentation Manager(R), Macintosh(R), and even a little bit on the UNIX(R) machine with X(R) and MOTIF(R)." This competency is also related to *Direct Application of Classroom Training* which is listed below under rejected competencies.

Eleven subjects referred to the competency of *Focus on User or Customer Needs.* These engineers were driven to produce products reflecting the true needs of the customer or end user. The subjects described situations in which they used or proactively collected customer data in order to define a product. The subjects valued the customer and felt success only when the end user appreciated their products.

Eleven subjects described situations in which they *Seek Help from Others.* The subjects describe situations in which they use others to significantly improve their own work result. Subjects would use others to review their work and offer suggestions for

improvement. The subjects even viewed the inputs from others as a way to grow as an individual. They would contact others to get ideas about how to approach a particular problem. They would call others in a attempt to find the answer to a specific problem.

In the *Proactive/Initiator/Driver* competency, subjects demonstrated that they influenced the course of a project by choosing to become more actively involved. They would influence others in their decisions about design or organizational structure. They would look for ways to surmount barriers and remove obstacles. One subject described a situation in which she attempted to get project decisions made. "I would publish memos, and I published a decision since the steering committee did come to a decision, I published the decision, and everyone went, 'No, no, no, you can't publish a decision; who are you to do this?" Eleven subjects exhibited the Proactive/Initiator/Driver competency.

Many subjects exhibited a *Proactive Role with Management* in an attempt to move a project in the direction they favored. This proactivity took the form of lobbying a manager for a specific point of view, creating demonstrations to give a clearer picture of a desired direction, creating portions of a product on their own time, and raising issues to management when they threatened to interfere with project progress. One subject described a situation in which he and another engineer worked in their spare time to create a product definition. "We created project schedule; We created resource requirements; We created the overall product plan at a high level." Ten subjects exhibited this competency somewhere in their interview so it was retained for further consideration.

Many subject described situations in which they *Leverage or Reuse Code*. Subjects looked for sources of code that they could use directly in a current design. As one subject put it, "I'm up for stealing anything I can, so it's the throw out the 'not invented

here' attitude. ... There are other places where I can make contributions rather than rewriting someone else's stuff." Other subjects emphasized designing their own code so it was readily reused. Ten subjects described incidents in which they exhibited the Leverages or Reuses Code competency. Of course, some subjects took an alternate point of view. One notes, "I really wanted to rewrite everything and of course my manager wouldn't let me."

The *Use of Structure Techniques* competency describes a very pragmatic approach to the use of formal methods. In general, these subjects emphasized the use of these techniques as a communication aid during joint development or when a design needed to be passed on to another individual. These subjects did not, as a group, see structured techniques as a panacea for design, but as a useful tool in certain situations. This competency was exhibited by nine subjects in the study.

Nine subjects described their *Response to Schedule Pressure* in the course of the interviews. In general, the subjects described the aspects of the design that they felt forced to sacrifice when a project fell behind schedule. When schedule pressure hit, these subjects were forced to provide incomplete documentation, perform inadequate inspection or testing, and failed to completely design the product. This is a negative competency in that it describes a result not usually praised by management. However, it realistically portrays the conflict the subjects faced.

In the *Methodical Problem Solving Approach* competency subjects describe situations in which they used a careful method for solving problems. They would build mental or physical system models to enhance their understanding of the problem. The category of *Debugging Approach* was merged into this competency. Here subjects would design well controlled experiments to efficiently locate the problem's cause. As one subject described the process, "We drew several partial successes, each of which gave

The *Sense of Vision* competency was combined with the *Goal Setting* competency to create the *Driven by a Sense of Mission* competency. The combined competency was referred to by six subjects overall. Both of these categories expressed the sense that the individual was clear on the goals they were trying to achieve and was driven to achieve them. This competency expresses the concept of focus and shared direction among team members. It also relates to the sense that they were working for a higher purpose rather than just a job. This new competency is retained for further analysis.

In the *Use of New Methods or Tools* competency subjects described situations in which they sought out the best approaches for completing their tasks. They recognized value in new tools and methods and proactively searched for these. As one subject notes, "I spent enough time looking into a new debugger which was a different one than the standard MS-DOS(R) one, because it was much easier to look at structures and things like that; It made it a lot more productive too." This competency was described by five subjects.

Four subjects described the situations in which they *Schedule and Estimate Well* on their projects. There seemed to be two forms of the situations described here. One group showed a high degree of concern for schedules and took pride in being able to estimate project schedule well. Subjects from this group report creating good schedules and then meeting them. The other group in this area show low concern with scheduling or describe having done a poor job of scheduling. As one subject put it, "I don't think I'm that good at estimating. They just kind of take as long as they take, and then they're done, that's kind of how I see it."

The competency of *Uses Code Reading to Ensure Final Code Quality* was exhibited by four subjects. This category grew out of a more general *Coding Style* area. Upon closer analysis of the observations in this category, it became apparent that the key

concept expressed here was the theme of code reading in the production of high quality code. The subjects broke into two groups on this theme. One group viewed code reading as a public team activity while another viewed it as a private individual activity.

The *Lack ofEgo* competency describes the degree to which subjects are open to the ideas of others and don't feel that they need to be the source of all good ideas. Subjects describe conflict situations in which issues are discussed in a very heated manner, but personal attacks are never made. This competency was exhibited by four subjects.

The *Strength of Convictions* competency describes the subjects who express a principle base in their actions. They describe a strong sense of personal involvement in the job and describe the personal integrity of the individual. One subject described a situation where he was asked to allow a prototype to be developed directly into a product by another engineer. He described his continued objections to the notion, resulting ultimately in his being labelled as uncooperative. However, he stuck to his convictions. This competency was uncovered in three of the interview transcripts.

Three subjects exhibited the competency of *Willingness to Confront Others.* Here subjects are reluctant to let a conflict simmer and will openly confront another person in order to resolve it. The engineers would raise this issues with other engineers or with managers.

Three subjects exhibited the *Mixes Personal and Work Goals* competency. Here subjects found ways to align their project goals with their own personal development goals. These subjects lobby with their managers to receive the work assignments which match their own personal desires.

The *Helps Others* competency was expressed by two subjects during the interviews. One subject described their role as a lab-wide consultant to help with scheduling issues and people issues. Another subject described situations in which he helped others accomplish their tasks, often without being asked. The Helps Others competency is a natural analog to the Seeks Help from Others competency. It is curious that while 11 subjects described seeking help from others, only two described giving help to others. This may be a natural outcome of the subjects deciding just what the important topics are in the interview. They may not adequately value their help to others.

4.1.3.1.2 Rejected Identified Competencies

Some of the competencies were rejected from further consideration. These competencies could be dropped if either they were ill-defined or if they were expressed by too few of the participants. Since the subjects selected to topics to discuss in the interviews, this does not mean that these competencies are not important. It could also mean that they are important but assumed. Many of the competencies will reappear when subjects define which competencies they think are important for exceptional performance. The competencies removed from further analysis at this stage are discussed below.

Motivations emerged as a catch-all category describing a broad range of responses. The motivations category included responses from 12 subjects. The category generally referred to the specific things that motivated and individual on the job. The category contains references to those things the individual likes and does not like to do. Particular examples of things subjects liked to do include building mental models, enjoying the tasks of the job, working alone, and pushing hard. Things the subjects did not like to do include playing on computers, documentation, and working alone. The

competency *of Acceptance of Difficult Situations* was merged with Motivations since it described the personal response to frustration and hence related to the motivation of the individual. Since the Motivations category was rather ill defined, and many of the key points were included in other competencies, the motivation competency was dropped from further consideration.

Direct Application of Classroom Training is highly related to the competency of *Obtains Necessary Training* above. Since only three subjects expressed this somewhat different competency, it was dropped as a separate item.

Flexibility related to subjects who were able to work competently in many different areas on a project. It also relates to the willingness of the individual to take on these multiple roles. This competency was dropped from further consideration since only two subjects described these activities.

Two subjects exhibited a competency of *Recognizes Need for Leadership.* Here the subjects express the need for leadership in their teams and frustration that no leader emerged. The category recognizes the need for someone to ultimately settle disputes among peers.

Two subjects describe their *Use of Metrics* in creating or analyzing designs. One used performance metrics to asses the quality of a solutions. The other used design metrics to measure progress in the design.

The competency of *Enjoys Freedom of Action* was exhibited by two subjects. It relates to the degree to which the individual can be self-directed in completing assignments. It is rejected from further consideration since it appears in only two transcripts.

The competency of *Celebrates Success* was observed in only one subject, hence it was not included in any further study. The subject described the celebration that he and his team engaged in to recognize their results. He also described situations where team members would take time to congratulate others on their individual successes.

One subject exhibited a trait of *Pursuing Multiple Parallel Paths* in accomplishing his objectives. He described a trait of keeping multiple design paths active in parallel with the expectation that most would fail to succeed, but overall much progress was made. This competency was not pursued since only one subject described it.

One subject described the use of *Automation* for the creation of test suites useful in the design of his product. The concept of building tests in parallel with code is captured elsewhere, but the concept of automation is dropped from further consideration.

One subject exhibited the competency of *Wants to Try New Things.* This was an attitude that the more variety he experienced on the job, the more he enjoyed his work. It is related to the very general category of *Motivation* above and is dropped from further consideration.

4.1.3.1.3 Summary of Identified Competencies

The full analysis of the retained competencies identified from *Protocol Analysis* of the transcripts in included in Appendix [XREF]. Table 4.10 lists these retained competencies in summary form. The competency names are suggestive, but the appendix should be consulted for the full definition. The competencies were analyzed on a differential basis using the Fisher's Exact Test. The score used for this test was the number of subjects which described behavior exhibiting this competency. Only one of the competencies exhibited significant differences based on application of

Fishers Exact Test with a 2-tail probability looking differentially at exceptional and non-exceptional subjects. *Use of Prototypes* was significantly different with a 2-tail computed significance level of 0.0108. This is especially noteworthy given that the sample size is so low. None of the rest of the competencies exhibited significance at the 0.05 level or better. This is not surprising given the small sample size of 20.

The identification of competencies of software engineers is an important result in its own right, even if they are only *threshold* competencies. Threshold competencies are those competencies which are important to the job and are exhibited equally by exceptional and non-exceptional performers. These will lend significant insight into the job of software engineering.

This section presented the set of competencies identified via protocol analysis of 20 Phase 1 Critical Incident Interviews. The section discussed the method used, defined each competency discovered, rationalized the merging of related concepts, and identified those competencies worthy of further analysis in Phase 2.

TABLE 4.10 Retained Competencies from Transcript Analysis (Phase 1)

COMPETENCY NAME	# Exceptional Subjects	# Non-Exceptional Subjects
Uses Decomposition Design Techniques	7	9
Use of Prototypes	10	4
Team Oriented	9	5
Writes/Automates Tests in Parallel with Coding	9	4
Possesses Unique Knowledge	7	6
Obtains the Necessary Training	6	6
Pride in Quality and Productivity	6	6
Seeks Help From Others	4	7
Proactive/Initiative/Driver	6	5
Focus on User or Customer Needs	6	5
Leverages/Reuses Code	5	5
Proactive Role with Management	5	5
Uses Structured Techniques for Communication	6	3
Methodical Problem Solving	6	3
Responds to Schedule Pressure	4	5
Emphasizes Elegant and Simple Solutions	5	3
Driven by Desire to Contribute	5	3
Sense of Fun	5	2
Driven by a Sense of Mission	4	2
Use of New Methods or Tools	4	1
Schedules and Estimates Well	4	0
Uses Code Reading to Ensure Final Code Quality	3	1
Lack of Ego	3	1
Mixes Personal and Work Goals	0	3
Willingness to Confront Others	1	2
Strength of Conviction	2	1
Helps Others	2	0

4.1.3.2 Self-Described Competencies

This section reports the responses given when the subjects were asked to name the skills, knowledge, or personal attributes most important in helping them achieve their success in the incident described. The subjects were prompted for this response by a very open-ended question. Hence the replies are presumed to be the competencies considered most significant by the study participants.

Each subject enumerated those competencies which they felt most contributed to their success. A summary list was created for each subject. All summary lists for each of the 20 subjects were combined into a single list of competencies. Related competencies were merged to form a single competency. The number of subjects, both exceptional and non-exceptional, expressing the competency was noted. What follows is a description of these self-described competencies. The descriptions are ordered by the number of subjects mentioning the competency.

The competency of*Perseverance* was noted by thirteen subjects. It relates to the extra work, discipline, stubbornness, compulsiveness, dedication, and willingness to work hard on a task. One subject described his testing experience as, "Testing is tedious, but if you're disciplined and have a procedure you get through it; It took 60 hours per week."

Twelve subjects described *Knowledge* as essential to the completion of their assignments. They describe particular skills, knowledge of the problem domain, knowledge of the solution domain, and general familiarity with the type of work as being significant in completing their tasks. One subject put it as, "I think I was the best person suited for this mainly because I knew the most about the overall operation of the program to begin with, the diagnostics program, as far as how it ran and all the

intricacies of it, because I had done the most work on it earlier." This competency is highly related to the *Possess Unique Knowledge* competency uncovered in the protocol analysis.

Teamwork was frequently mentioned as an important competency for success. This area included recognizing and using the strengths of others, ability to work with others, being sensitive to others, taking joint ownership, and paying attention to more than the technical aspects of a project. One engineer summed up this way, "I think a lot of times we overvalue the technical skills and undervalue the people skills for people that are in technical positions." Twelve subjects identified this as a key competency. This competency is related to the *Team Oriented* competency from protocol analysis.

Eleven subjects identified *Skills/Techniques* as a key competency. They cite comfort with design techniques, debugging skills, technology choices, and technical and software development background as keys to their success. One subject identified his own unique skill this way, "There probably aren't that many people ... who can run through assembly language these days and do it with any kind of panache." This competency is related to the *Use of Methods and Tools* competency from protocol analysis.

These subjects defined *Thinking* as the ability to think algorithmically and structuredly. They saw value in being able to build models in their mind, ability to structure problems, ability to think of alternative solutions, and being able to see basic structure and basic theory. As one subject notes, "I'm a fairly good problem solver, I think I can gather information and extract the part of the information that's relevant to my problem and then apply it to a solution."

Eight subjects valued *Communication* as a key to success. They defined this as making sure that they understand others, engaging in constant communication, responding to new ideas, being open, and being sensitive to others in conversations. One subject notes that, "Communication is paramount, technology has been less important." This competency is related to the *Uses Structured Techniques for Communication* from protocol analysis.

The competency of *Learning* was considered key by many subjects. This competency is defined as being able to pick up new techniques quickly, a willingness to learn and train yourself, and a focus on improving skills. One subject describes the concept as, "It's so interdisciplinary - you've got to be willing to do anything and believe you can do it with training as well as the people who are already doing it." Learning was expressed as an important competency by seven subjects. This competency is related to the *Obtains Necessary Training* competency from protocol analysis.

Five subjects related *Desire to Do* as an important competency. They defined it as a bias for action, sense of urgency, being results oriented, and a willingness to try things. One subject described the skill as the ability to, "Just jump right in and start working - you develop the capability."

Five subject found that *Challenge* was essential to their performance. They cite curiosity and enjoying working in new areas as important. This competency is related to the *Driven by a Desire to Contribute* competency identified in protocol analysis.

Subjects describe the competency *ofAttention to Detail* as the ability to deal with complexity, and being detail oriented. One subject notes that, "[I'm] good at taking a lot of detail from several different things and getting it into my mind all the same time and kind of working on it and munching it around in my head." This competency was identified by four subjects.

Four subjects identified *Thoroughness* as an important competency. They define this as making sure all paths are covered, being methodical, being organized, and being overcautious. One subject described his experience, "I just had a real fear of ever being in that situation [lots of defects in code]; ... I just went slowly and overdid if anything."

Four subjects identified *Innovation* as key to their success. They defined this as having creative ideas. One subject states, "I like to think of alternatives, being creative and ... practical at the same time."

The *Conviction* competency was identified by four subjects as being important in their own success. They defined conviction as belief in the project or product, doing the right thing, and selling projects. This competency is related to the *Strength of Convictions* competency identified in protocol analysis.

Four subjects indicated that *Seeks Help* is an important competency. They value the encouragement provided by other people. They see it as important to know when you don't have enough knowledge and go and get help. It is also important to allow others to criticize one's work or to give new input. This competency is related to the *Seeks Help from Others* competency identified in protocol analysis.

Three subjects *identified Experience* as an essential competency for success. They defined this as prior experience with a similar project.

Three subjects identified *Prototyping* as important to their success. They defined prototyping as an approach for demonstrating feasibility and as a start on implementing the most feasible alternative. This competency is related to the *Use of Prototypes* competency identified in protocol analysis.

The *Desire to Improve Things* competency was identified by three subjects as important for their success. They defined this as not being satisfied with the status quo, setting high personal expectations and goals, and giving yourself time for improvement.

Two subjects identified *Scheduling* as in important competency. This was defined as planning ahead, and schedule setting ability. Although this would normally have been below the cut line, it was retained for consideration since it is related to the *Schedules and Estimates Well* competency identified from protocol analysis.

Simplicity was identified by two subjects as essential to their success. They defined this as an aversion to complexity, simplifying things as much as possible, and not letting a problem get too complex. One subject found that, "A little bit of vision that we could do things by rethinking things we could do things simpler and more elegantly." Although this competency would normally have fallen below the cut line, it was retained for consideration since it is related to the *Emphasizes Elegant and Simple Solutions* competency identified from protocol analysis.

Two subjects identified *Quality* as an important competency. They defined this as making sure the result is readable, a concern for reliability, and a commitment to high quality. This competency is retained since it is related to the *Pride in Quality and Productivity* competency identified in protocol analysis.

Additional competencies were identified that were not included in any further analysis. These were generally those competencies that were mentioned very infrequently by subjects. Those competencies excluded from further study are listed below.

1. Attitude
2. Judgement
3. Mentoring
4. Reputation for Being Right
5. Knowing What's Going On
6. Inquisitiveness
7. Responsive to Customer Needs
8. Involvement
9. Willingness to Take Risks
10. Ability to Deal With Uncertainty
11. Consistency
12. Pride
13. Thorough Testing
14. Patience
15. Maintaining a "Big Picture" View
16. Clarity
17. Incremental Investigation
18. Flexibility
19. Lax Management
20. Break From Problem
21. Fear of Having Something Bad Happen
22. Ability to Create Win/Win Situation

Many of the competencies appear in other portions of the analysis and will be included for that reason.

The full table of the competencies is included in Appendix E Self-Described Competencies. Table 4.11 shows the retained self-described competencies in abbreviated format. The *# Exceptional Subjects* column indicates the number of exceptional subjects who expressed this competency. Likewise, the

 # Non-Exceptional Subjects column does the same for the non-exceptional subjects. All competencies were tested for significant difference on a differential basis for exceptional and non-exceptional subjects using a 2-tailed Fisher Exact Test. None were found to be statistically significant differences.

TABLE 4.11 Retained Self-Described Competencies (Phase 1)

COMPETENCY NAME	# Exceptional Subjects	# Non-Exceptional Subjects
Perseverance	6	7
Knowledge	6	6
Teamwork	6	6
Skills / Techniques	6	5
Thinking	5	4
Communication	4	4
Learning	4	3
Desire to Do	2	3
Challenge	1	4
Attention to Detail	2	2
Thoroughness	2	2
Innovation	2	2
Conviction	2	2
Seeks Help	0	4
Experience	1	2
Prototyping	2	1
Desire to Improve Things	3	0
Schedules Well	2	0
Simplicity	0	2
Quality	2	0

This section has presented the competencies that engineers indicated were important to their success on the job. One set of these competencies has been retained

for further analysis. As second set of competencies was mentioned too infrequently to warrant further consideration. Note that many of the competencies cited by engineers as being important to their own success, are in fact the same competencies identified from the analysis of the transcripts.

4.1.3.3 Manager Described Competencies

This section presents the competencies that managers feel differentiate their exceptional performers from their non-exceptional performers. This data further contributes to the triangulation of the results by providing another source of data for describing essential competencies.

The following list of competencies was created by asking the lab managers who participated in Phase 1 of this study: *What are the Knowledge, Skills, or Attributes that differentiate your exceptional performers from your non-exceptional performers?* They are listed in order of frequency of mention. There were five labs involved in the Phase 1 study and hence five managers answered this question. The number in parenthesis following the competency indicates the number of managers, out of five, who offered this as a differential competency. There was no discussion with these managers to provide further elaboration on these competencies.

1. Skilled in Architected Approach to Software Engineering *(3)*
2. Breadth of View and Influence *(3)*
3. Technical Expertise in Problem Solving and Implementation *(3)*
4. Breadth of Knowledge and Experience *(3)*
5. Leadership *(2)*
6. People Skills and Teamwork *(2)*
7. Technical Judgement *(2)*
8. Positive Attitude / Can Do Approach / Self Starter *(1)*
9. Willingness to Understand Different Disciplines and Make Tradeoffs *(1)*
10. Religiously Following Best Design and Testing Approaches *(1)*
11. Consistent Performance *(1)*
12. Flexibility *(1)*
13. Expertise That is Sought *(1)*
14. Reliable *(1)*
15. Accurate Schedules *(1)*
16. Pleasant Surprises *(1)*

Many of these competencies are similar to the ones either identified in the analysis of the transcripts or cited by engineers as those leading to exceptional performance.

4.1.3.4 Summary of Competencies

Table 4.12 summarizes the competencies identified from the multiple sources listed above. The item number column indicates which competency card located in Appendix H Phase 2 Competency Statements describes the competency in more depth.

TABLE 4.12 Retained Competency Summary Table (Phase 1)

(n=20)

LABEL	COMPETENCY	Derived	Self-Described	Manager
Item #1	Team Oriented	14	12	2
Item #2	Seeks Help	11	4	
Item #3	Helps Others	2	1	1
Item #4	Use of Prototypes	14	3	
Item #5	Writes/Automates Tests with Code	13		
Item #6	Knowledge	13	12	
Item #7	Obtains Necessary Training / Learning	12	7	
Item #8	Leverages/Reuses Code	10		
Item #9	Communication / Uses Structured Techniques for Communication	8	8	
Item #10	Methodical Problem Solving	9		
Item #11	Use of New Methods or Tools	5		
Item #10	Schedules and Estimates Well	4	2	1
Item #13	Uses Code Reading	4		
Item #14	Design Style	16		
Item #15	Focus on User or Customer Needs	11	1	
Item #16	Response to Schedule Pressure	9		
Item #17	Emphasizes Elegant and Simple Solutions	8	2	
Item #18	Pride in Quality and Productivity	12	1	
Item #19	Proactive/Initiator/Driver	11		
Item #20	Proactive Role with Management	10		
Item #21	Driven by Desire to Contribute	8	5	
Item #22	Sense of Fun	7		
Item #23	Sense of Mission	6		
Item #24	Lack of Ego	4		
Item #25	Strength of Convictions	3	4	
Item #26	Mixes Personal and Work Goals	3		
Item #27	Willingness to Confront Others	3		
Item #28	Thoroughness		4	
Item #29	Skills / Techniques		11	
Item #30	Thinking		9	
Item #31	Desire to Do / Bias for Action		5	1
Item #32	Attention to Detail		4	
Item #33	Perseverance		13	
Item #34	Innovation		4	
Item #35	Experience		3	
Item #36	Desire to Improve Things		3	
Item #37	Quality		2	
Item #38	Maintaining a "big picture" view / Breadth of View and Influence		1	3

The **Derived** category refers to those competencies extracted from the analysis of the interview transcripts. They represent those areas which the subject chose to discuss during their narration about their experiences. The number in this column reflects the number of subjects which described behaviors related to this competency.

The **Self-Described** column refers to the competencies offered when subjects were asked to describe the skills, knowledge, and attributes associated with their successful performance on projects. The number in the column indicates the number of subjects which expressed this competency as important to their success.

The **Manager** column relates to the competencies suggested when managers were asked what are the skills, knowledge, or attributes which differentiate the exceptional and the non-exceptional performers in their labs. The number in the column indicates how many of the 5 managers participating in Phase 1 cited this competency as important.

The competencies with the most occurrences from the most sources were given preference for further study. The competencies derived from the protocol analysis were considered to be more important than the competencies offered directly by the engineers or the managers. This is because this study is based on the notion that behaviors associated with high performance are the unit of study. Competencies which are validated by multiple sources are considered to be more important than competencies which come from only one source. Finally, some degree of judgement was applied to pick the best and most complete set of competencies to study further. An arbitrary limit of 38 final competencies was imposed in order to make subsequent study practical.

Table 4.13 summarizes the competencies which have been rejected for further study. In some cases this was due to overlap with competencies which were included above. In most cases it was due to the fact that few people identified the competency, or it was not validated by multiple sources.

TABLE 4.13 Rejected Competency Summary Table (Phase 1)

(n = 20)

COMPETENCY	Derived	Self-Described	Manager
Flexibility	2	1	1
Judgement		2	2
Skilled in Architected Approach to Software Engineering			3
Technical Expertise in Problem Solving and Implementation			3
Leadership			2
Willingness to Understand Different Disciplines and Make Tradeoffs			1
Religiously Following Best Design and Testing Approaches			1
Consistent Performance			1
Reliable			1
Pleasant Surprises			1
Testing	4	1	
Direct Application of Classroom Training	3		
Enjoys Freedom of Action	2		
Celebrates Success	1		
Pursues Multiple Parallel Paths	1		
Recognizes Need for Leadership	2		
Use of Metrics	2		
Automation	1		
Wants to Try New Things	1		
Attitude		2	
Reputation for Being Right		1	
Knowing What's Going On		1	
Inquisitiveness		1	
Involvement		1	
Willingness to Take Risks		1	
Ability to Deal With Uncertainty		1	
Consistency		1	
Timing and Dumb Luck		1	
Patience		1	
Clarity		1	
Incremental Investigation		1	
Lax Management		1	
Break from Problem		1	
Fear		1	
Win/Win		1	

This section has completed an analysis of three sources of competencies: protocol analysis of interviewee transcripts; self-described competencies of interviewees; and differential competencies proposed by managers. The section described these competencies in depth, and indicated which competencies should be merged into a single competency. All the competencies were identified and subjected to a weighting scheme in order to determine which competencies should be retained for further analysis. The final 38 competencies to be retained were summarized in a single table.

4.1.4 Summary of Phase 1 Data

This closes the presentation of data for Phase 1 of the research. This data has described the population from which the sample of 20 Phase 1 subjects was drawn. The data show that the biographical information collected does not relate to exceptional performance with the exception of *Years at Company in Software.* This variable will be examined closely in Phase 2. The Myers-Briggs Type Indicator test did not reveal any statistically significant differences between exceptional and not exceptional performers. It did reveal a preponderance of *Intuitive* and *Thinking* types in the full study sample. The search for competencies revealed a rich set of competencies which warrant further study. A set of 38 competencies has been selected for further study in Phase 2.

4,2 Phase 2 Quantitative Data Collection

Phase 2 forms the validation portion of the research project. It attempts to validate the results of Phase 1 and provide a predictive model of performance based upon the competencies discovered in Phase 1. Phase 2 repeats the study of demographic information from Phase 1 in order to again validate that no simple predictors of performance exist. Phase 2 analyzes the competencies discovered in Phase 1 to determine which are associated with exceptional performance. Finally, Phase 2 collects the data required to see if a model, albeit complex, can be proposed to explain the difference in performance between exceptional and non-exceptional performers.

4.2.1 Subject Selection

This study surveyed Software Engineers in nine Company divisions. All were located in the same geographical area. The divisions represent a broad range of software development including languages and compilers, test and measurement software, graphics system software, embedded firmware for hardware control, and CAE/CAD/CASE applications software. Table 4.14 indicates the total response rates to the survey from all of the sample population.

Of the responses provided in Phase 2, only four were unacceptable due to being incomplete in some of the major independent variables. The remainder of this analysis considers only valid surveys for each variable under study. Hence there will be some variation in the reported number of samples *n*. The response rate of close to 50% is a strong result and may well indicate the level of interest in this information at the Company studied. The sample of 129 participants provides sufficient statistical power to complete the study. The response rates for exceptional and non-exceptional

TABLE 4.14 Population Summary (Phase 2)

POPULATION SUMMARY	TOTAL
Surveys Distributed	275
Total Responses	129
Response Rate	46.9%
# Exceptional Responses	41
# Non-Exceptional Responses	88
% Exceptional	31.8%
% Non-Exceptional	68.2%

performers was close to identical. Thirty percent of the surveys were distributed to exceptional performers and 70% of the surveys were distributed to non-exceptional performers. The return rates of 31.8% exceptional surveys and 68.2% non-exceptional surveys closely match this distribution ratio.

4.2.2 **Biographical Questionnaire**

This section discusses the descriptive statistics collected for the population at large. The data are compared with the Phase 1 results to ensure that the samples are similar. The data are also analyzed for normality so that subsequent statistical steps will be valid. Distributions will be considered normal when both the *skew* and *kurtosis* of the data are within the range of plus or minus 1.00. The Biographical Questionnaire used to collect this information in presented in Appendix G Phase 2 Biographical Questionnaire. Tables 4.15 through 4.23 present the data from Phase 2.

Table 4.15 presents the distribution of subjects by gender. The Phase 2 population is very similar to the 75/25 split shown in Phase 1. This is taken as an indication of the population at large for Software Engineers at The Company. The skew is out of

range on gender indicating that the population is not normally distributed. This is a reflection of the population itself. This ratio of males to females is consistent with published reports [Pear 90] that women constitute only 30% of the employed computer scientists.

TABLE 4.15 Subject Gender (Phase 2)

(n = 128)

GENDER	FREQUENCY	% of TOTAL
Female	27	21.1%
Male	101	78.9%

Skew = -1.385; Kurtosis = -0.84

Table 4.16 presents the age of Phase 1 participants. The mean age of 32.45 years is comparable to Phase 1's mean of 33.45 years. The range is broader in Phase 2. Part of that is due to not limiting the survey to engineers with two or more years of experience. Also, a larger population is just more likely to have some older participants. The skew and kurtosis show that the sample is normal with respect to age.

TABLE 4.16 Subject Age (Phase 2)

(n = 121)

AGE	MEAN	STD DEV	RANGE
	32.45	5.68	22-49

Skew = 0.718; Kurtosis = 0.376

Table 4.17 presents the educational information for the Phase 2 subjects. Again the subject pool is well educated with over 53% of the sample holding two or more degrees. The *Number of Degrees Completed* are normally distributed as indicated by

a skew of 0.539 and kurtosis of 0.119. The *Number of Degrees Partially Completed* is quite skewed toward the no degrees completed end with a skew of 1.698 and a kurtosis of 1.724. This is because 103 of the 129 subjects had no partially completed degrees.

TABLE 4.17 Subject Education (Phase 2)

(n = 129)

	FREQUENCY	% of TOTAL	CUMM %
# DEGREES COMPLETED			
None	2	1.6%	1.6%
One	58	45.0%	46.5%
Two	56	43.4%	89.9%
Three	12	9.3%	99.2%
Four	1	0.8%	100%
# DEGREES PARTIALLY COMPLETED			
0	103	79.8%	79.8%
1	25	19.4%	99.2%
2	1	0.8%	100%

As shown in Table 4.18, 40% of the subjects hold a Master's degree as their highest degree. This reflects a well educated subject pool and corresponds well with the Phase 1 finding.

Table 4.19 indicates the frequency with which subjects held certain popular degrees. For each of the Computer Science, Engineering, and Mathematics degrees, the table notes the frequency with which at least one degree is held with that major by the subject pool. Over 74% of the subjects held at least one degree in Computer Science. This is in contrast to a rate of only 35% in the Phase 1 subjects. This may be

TABLE 4.18 Highest Degree Held (Phase 2)

(n = 125)

HIGHEST DEGREE HELD	FREQUENCY	% OF TOTAL	CUMM %
None	2	1.6%	1.6%
Bachelors	68	54.4%	56.0%
Masters	50	40.0%	96.0%
PhD	5	4.0%	100%

Skew = 0.477; Kurtosis = -0.238

due to the fact that Phase 2 subjects come from a broader population and the additional labs surveyed produce Operating Systems, Graphics Systems, and Application Software products. These groups may be more likely to hire these graduates.

TABLE 4.19 Majors for Degrees Held (Phase 2)

(n = 125)

DEGREE HELD	FREQUENCY	% OF TOTAL SUBJECTS HOLDING DEGREE[1]
Computer Science[2]	94	75.2%
Engineering[3]	49	39.2%
Mathematics	26	20.8%

1. Degrees held will exceed the number of study participants since participants hold multiple degrees.
2. Computer Engineering degrees are double counted in Computer Science and Engineering.
3. Includes all disciplines of Engineering - Electrical, Computer, Mechanical, ...

The number of subjects with Engineering degrees is still strong at 39%. Again, this is in keeping with the culture of The Company. Finally, the next most frequently cited degree was in Mathematics with over 20% of the population reporting at least one degree in this field.

The mean number of hours of training in the past two years similar for the Phase 2 and Phase 1 subjects. Phase 2 subjects (n = 125) completed 102 hours of training on average in the past two years, while Phase 1 subjects (n = 20) completed 117 hours on average. This training included college classes, corporate classes, seminars, self-study, and any other area considered significant to the subject. It is normalized to *contact hours*. For formal classes this would be the number of hours spent in the classroom. For self-study, this is the number of hours spent in the form of study. The Phase 2 subjects reported a range of zero to 683 hours of training in the past two years. The high end of this range is more than double in the Phase 2 pool over the Phase 1 pool. The skew value in for training of 2.196 indicates that the distribution of training hours is not normally distributed. Further, the kurtosis value of 5.490 indicates that the distribution of training hours is peaked in certain areas. The distribution has a large peak at zero hours with sixteen subjects reporting no hours of training in the past two years. The distribution is also skewed. The median value is 50 hours indicating a very small but long tail of the distribution into the higher values of training hours.

The *Languages* variable covered in Table 4.20 shows another non-normal distribution. There is both high skew and kurtosis on this variable. The mean number of languages known with high skill at 2.7 is comparable to the Phase 1 value of 2.5. The low skill value of 1.7 is higher than the Phase 1 value of 1.1 but can be at least partially explained by the removal of a *medium skill* category for the answer. The total languages row shows a more normal variable that has no problem with its kurtosis and

is only slightly over the skew limit value. Since the languages total variable more closely matches a normal distribution, this is the variable which will be retained for further study.

TABLE 4.20 Language Usage (Phase 2)

(n = 125)

LANGUAGES	MEAN	RANGE	SKEWNESS	KURTOSIS
High Skill	2.696	0-10	1.519	4.141
Low Skill	1.720	0-8	1.432	3.703
Total Languages	4.416	1-11	1.084	0.844

The variables presented in Table 4.21 indicate the years of experience subjects have with the Company and outside of the Company. Further, the table indicates the years of experience the subjects have in Software Engineering and outside of Software Engineering. The mean number of years worked at The Company in software engineering dropped from 7 years in Phase 1 to 6 years in Phase 2. This can be explained by the fact that Phase 2 surveyed *all* software engineers in each lab and did not specify a minimum experience criterion. This is demonstrated by the low end of the range for this variable being 0.5 years. The number of years spent in software engineering at the Company shows acceptable kurtosis and nearly acceptable skew. The rest of the variables are not normally distributed. This is to be expected given the relatively small number of subjects who have non-zero years of experience in these areas. A new variable, *Software Total*, was calculated. This variable is more normal as shown by skew and kurtosis. It is also the variable measured in previous studies which attempted to predict performance [Chry 78]. This is the variable which will be retained for further analysis.

TABLE 4.21 Work Experience (Phase 2)

(n = 123)

YEARS EXPERIENCE	SOFTWARE ENGINEERING					NOT SOFTWARE ENGINEERING				
	MEAN	RANGE	SKEW	KURT	n[1]	MEAN	RANGE	SKEW	KURT	n[1]
Company	5.974	0.5-19	1.028	0.996	123	1.191	0-10.5	2.345	4.729	33
NOT Company	0.974	0-10	2.421	5.961	37	0.374	0-12	5.663	37.470	14
Software Total	6.948	0.67-20	0.991	1.068	123					

1. **Indicates the number of subjects with non-zero values. The reported statistics still describe the full sample of n = 123.**

The *Total Years Worked* variable in Table 4.22 shows acceptable skew but unacceptable kurtosis indicating unusually high peaks in the distribution. The mean value in Phase 2 is 1.5 years lower than in Phase 1. Again, this can be explained by the fact that Phase 2 was open to all lab members regardless of the amount of experience.

TABLE 4.22 Total Years Worked (Phase 2)

(n = 125)

TOTAL YEARS WORKED	MEAN	STD DEV	RANGE	SKEW	KURTOSIS
	8.565	5.036	0.67-31	1.000	2.142

4.2.2.1 Differential View of Descriptive Statistics

Table 4.23 presents the biographical questionnaire information on a differential basis. The table lists each variable collected or generated. It indicates which test is used to determine statistical significance. The Chi-Square test is used to test for statistically significant differences between categorical variables such as gender. The

t-test is used to test for statistically significant differences between ordinal variables. For each test, the *n* is included to indicate the number of samples used in the test. The calculated test values and calculated significance levels are also provided. Significance levels which fall within the 0.05 significance range are marked with '**'.

TABLE 4.23 Tests for Difference (Phase 2)

VARIABLE	TEST	n	CALCULATED TEST VALUE	CALCULATED SIGNIFICANCE LEVEL[1] *(2 Tail)*
GENDER	Chi-Square	128	0.28	0.594
AGE	T	128	-1.44	0.153
#Degrees Completed	T	129	-0.07	0.946
#Degrees Partially Completed	T	129	-0.18	0.854
Highest Degree Held	T	129	0.25	0.801
CS Degree Held?	Chi-Square	129	0.34	0.560
Engineering Degree Held?	Chi-Square	129	0.25	0.618
Math Degree Held?	Chi-Square	129	0.34	0.560
Training Hours	T	129	1.96	0.052
Languages - High Skill	T	129	-0.75	0.456
Languages - Low Skill	T	129	0.07	0.945
Languages - Total	T	125	-0.41	0.684
Years at Company in Software	T	127	-5.13	0.000 **
Years at Company NOT in Software	T	127	-0.26	0.799
Years NOT at Company in Software	T	127	0.86	0.393
Years NOT at Company NOT in Software	T	127	1.10	0.272
Total Years in Software	T	123	-3.75	0.000 **
Total Years Worked	T	129	-2.87	0.005 **

1. Differences are considered statistically significant when the calculated significance is less than 0.05. They are marked with '**'.

Table 4.23 indicates that three biographical variables are associated with exceptional performance: *Years at Company in Software, Total Years in Software,* and *Total Years Worked.*

4.2.3 Q-Sort Results

This section reports the results of the Q-Sort exercise portion of the Phase 2 survey. The section first reports the results from the full sample and indicates that the results are normally distributed. Then the section reports the results of a differential view of the data for exceptional and non-exceptional subjects. The section identifies those competencies which are associated with this difference in performance.

Table 4.24 presents the results of the Q-Sort activity on the survey. Participants sorted a set of 38 competencies into a quasi-normal distribution of seven piles. Each pile was assigned an integer value from zero to six. Zero means *Least Like My Behavior* while six means *Most Like My Behavior.* For each survey, the Q-Sort item was assigned the integer value associated with the pile that the subject placed it into. Table 4.24 presents the mean value of this score for all subjects for each item. The table is sorted by the mean Q-Score for the full sample of both exceptional and non-exceptional engineers. The skew and kurtosis numbers indicate that all Q-Sort items are normally distributed.

The competencies contained in Table 4.24 are sorted by the mean score of all study participants. This sorting confirms that competencies are indeed exhibited more or less frequently by the study participants. The mean values of competency scores range from 4.168 to 2.024.

TABLE 4.24 Q-Sort Competency Responses
(Sorted by Mean Score of Full Sample)
(n = 125)

LABEL	COMPETENCY	MEAN	STD DEV	SKEW	KURT
Item #37	Concern for Reliability and Quality	4.168	1.324	-0.590	0.351
Item #15	Focuses on User or Customer Needs	3.912	1.314	-0.507	0.266
Item #30	Strong Analytic Skills - Ability to Think and Visualize	3.880	1.383	0.051	-0.857
Item #21	Driven by Desire to Contribute	3.864	1.110	-0.086	0.064
Item #2	Seeks Help From Others	3.672	1.288	-0.032	-0.222
Item #18	Takes Pride in Quality and Productivity	3.632	1.298	-0.138	0.426
Item #17	Emphasizes Elegant and Simple Solutions	3.584	1.265	0.197	0.085
Item #34	Innovative	3.432	1.266	0.107	-0.154
Item #22	Enjoys Challenge of Assignment - Has Fun	3.400	1.524	0.006	-0.381
Item #32	Pays Close Attention to Detail	3.352	1.504	0.174	-0.739
Item #8	Leverages/Reuses Code	3.312	1.292	0.128	-0.132
Item #28	Perseverance	3.304	1.444	0.040	-0.573
Item #24	Stresses Solution over Source of Solution - Lack of Ego	3.256	1.069	0.154	0.098
Item #1	Team Oriented	3.176	1.476	0.088	-0.450
Item #29	Mastery of Skills and Techniques	3.136	1.279	0.047	0.106
Item #10	Uses Methodical Problem Solving Approach	3.128	1.114	0.098	-0.239
Item #36	High Personal Expectations and Goals	3.112	1.172	0.023	-0.112
Item #5	Write/Automates Tests in Parallel with Code	3.080	1.418	-0.023	-0.276
Item #19	Takes Initiative to Identify Ways of Completing Important Tasks	2.984	1.085	0.109	-0.271
Item #35	Prior Experience	2.960	1.340	0.115	-0.153
Item #7	Obtains the Necessary Training	2.896	1.230	-0.010	-0.472
Item #38	Maintains "Big Picture" View	2.896	1.565	0.252	-0.569
Item #13	Uses Code Reading to Ensure Final Code Quality	2.832	1.372	-0.205	-0.443
Item #11	Uses New Tools or Methods	2.816	1.352	0.242	0.293
Item #14	Uses Decomposition Design Techniques	2.744	1.361	-0.089	-0.090
Item #31	Driven by Bias for Action and Sense of Urgency	2.616	1.590	0.262	-0.457
Item #12	Schedules and Estimates Well	2.608	1.447	0.146	-0.272
Item #3	Helps Others	2.608	1.497	0.169	-0.560
Item #33	Methodical, Organized, and Cautious	2.560	1.682	0.215	-0.711
Item #25	Exhibits and Articulates Strong Beliefs and Convictions	2.544	1.644	0.032	-0.856
Item #27	Willingness to Confront Others	2.520	1.377	0.033	-0.391
Item #23	Driven by a Sense of Mission	2.504	1.182	0.333	0.264
Item #4	Uses Prototyping to Assess Design	2.488	1.423	0.366	-0.130
Item #26	Mixes Personal and Work Goals	2.440	1.316	-0.062	-0.222
Item #20	Proactively Attempts to Influence Project Direction by Influencing Management	2.320	1.619	0.428	-0.405
Item #6	Possesses Unique Knowledge	2.168	1.236	0.039	-0.337
Item #16	Responds to Schedule Pressure by Sacrificing Parts of Design Process	2.072	1.607	0.260	-0.901
Item #9	Uses Structured Techniques for Communication	2.024	1.434	0.291	-0.567

Table 4.25 lists the results of a t-test comparison of means for each of the Q-Sort Competencies. The means are calculated separately for exceptional and non-exceptional performance and tested for difference. The two means are considered different when the calculated significance level is less than 0.05. These are denoted by '**' in the table. The table is sorted by the mean scores of the exceptional responses. The *Delta* column represents the number of places that a particular competency moves in its rank order when sorted by exceptional means rather than sorted by the full sample means.

Table 4.25 demonstrates that nine competencies show statistically significant differences in the mean values reported by the exceptional and non-exceptional engineers. Thus 24% of the 38 identified competencies are related to the difference in performance of exceptional and non-exceptional engineers. The five competencies which have a higher mean for exceptional performers are:

1. Helps Others.
2. Proactively attempts to Influence Project Direction by Influencing Management.
3. Exhibits and Articulates Strong Beliefs and Convictions.
4. Mastery of Skills and Techniques.
5. Maintains "Big Picture" View.

The four competencies which have a higher mean for non-exceptional performers are:

1. Seeks Help from Others.
2. Responds to Schedule Pressure by Sacrificing Parts of Design Process.
3. Driven by Desire to Contribute.
4. Willingness to Confront Others.

This section presented the results of the Q-Sort exercise for sorting 38 competencies. The section presented the data for the full sample indicating that each competency's statistics were normally distributed. The full sample report also indicated

TABLE 4.25 Differential Q-Sort Competency Responses - T-Test Results
(Sorted by Mean Score of Exceptional Responses)

DELTA	LABEL	COMPETENCY	XP MEAN (n=40)	NXP MEAN (n=85)	TEST VALUE	SIG LEVEL[1] (2 Tail)
0	*Item #37*	Concern for Reliability and Quality	4.050	4.224	0.68	0.497
0	*Item #15*	Focuses on User or Customer Needs	4.025	3.859	-0.66	0.512
0	*Item #30*	Strong Analytic Skills - Ability to Think and Visualize	3.925	3.859	-0.25	0.804
+2	*Item #18*	Takes Pride in Quality and Productivity	3.750	3.576	-0.70	0.488
+2	*Item #17*	Emphasizes Elegant and Simple Solutions	3.725	3.518	-0.85	0.395
-2	*Item #21*	Driven by Desire to Contribute	3.550	4.012	2.20	0.029 **
+8	*Item #29*	Mastery of Skills and Techniques	3.500	2.965	-2.22	0.028 **
+20	*Item #3*	Helps Others	3.500	2.188	-4.99	0.000 **
-1	*Item #34*	Innovative	3.425	3.435	0.04	0.966
+12	*Item #38*	Maintains "Big Picture" View	3.425	2.647	-2.66	0.009 **
-2	*Item #22*	Enjoys Challenge of Assignment - Has Fun	3.325	3.435	0.38	0.707
-7	*Item #2*	Seeks Help From Others	3.325	3.835	2.10	0.038 **
0	*Item #24*	Stresses Solution over Source of Solution - Lack of Ego	3.250	3.259	0.04	0.966
+6	*Item #35*	Prior Experience	3.250	2.824	-1.67	0.097
-5	*Item #32*	Pays Close Attention to Detail	3.225	3.412	0.65	0.519
+3	*Item #19*	Takes Initiative to Identify Ways of Completing Important Tasks	3.100	2.929	-0.82	0.414
-3	*Item #1*	Team Oriented	3.050	3.235	0.65	0.515
-7	*Item #8*	Leverages/Reuses Code	3.050	3.435	1.56	0.120
-2	*Item #36*	High Personal Expectations and Goals	3.050	3.141	0.40	0.687
-8	*Item #28*	Perseverance	3.000	3.447	1.63	0.107
+9	*Item #25*	Exhibits and Articulates Strong Beliefs and Convictions	2.975	2.341	-2.04	0.044 **
+13	*Item #20*	Proactively Attempts to Influence Project Direction by Influencing Management	2.925	2.035	-2.95	0.004 **
+4	*Item #12*	Schedules and Estimates Well	2.900	2.471	-1.56	0.122
-8	*Item #10*	Uses Methodical Problem Solving Approach	2.900	3.235	1.58	0.117
-7	*Item #5*	Write/Automates Tests in Parallel with Code	2.775	3.224	1.66	0.099
+6	*Item #23*	Driven by a Sense of Mission	2.750	2.388	-1.61	0.111
-3	*Item #11*	Uses New Tools or Methods	2.700	2.871	0.66	0.513
-3	*Item #14*	Uses Decomposition Design Techniques	2.700	2.765	0.25	0.805
-3	*Item #31*	Driven by Bias for Action and Sense of Urgency	2.675	2.588	-0.28	0.777
-9	*Item #7*	Obtains the Necessary Training	2.600	3.035	1.86	0.065
-8	*Item #13*	Uses Code Reading to Ensure Final Code Quality	2.550	2.965	1.59	0.115
-1	*Item #4*	Uses Prototyping to Assess Design	2.550	2.459	-0.33	0.740
+3	*Item #6*	Possesses Unique Knowledge	2.325	2.094	-0.97	0.332
0	*Item #26*	Mixes Personal and Work Goals	2.250	2.529	1.11	0.270
-6	*Item #33*	Methodical, Organized, and Cautious	2.150	2.753	1.89	0.061
-5	*Item #27*	Willingness to Confront Others	2.125	2.706	2.23	0.027 **
+1	*Item #9*	Uses Structured Techniques for Communication	2.050	2.012	-0.14	0.890
-1	*Item #16*	Responds to Schedule Pressure by Sacrificing Parts of Design Process	1.600	2.294	2.29	0.024 **

1. Differences are considered statistically significant when the calculated significance is less than 0.05. These instances are denoted by '**'.

that there is a reported difference between the competencies considered to be *Most Like My Behavior* and the competencies considered to be *Least Like My Behavior*. This was demonstrated by the broad range of mean values for the competencies.

This section also presented the results of the Q-Sort exercise on a differential basis. T-test analysis of these results demonstrates that nine competencies are associated with the differences between exceptional and non-exceptional performers.

4.2.4 Discriminant Analysis

This section reports the Discriminant Analysis results of the survey data. It begins by analyzing the correlations of variables since highly correlated variables cannot be used in the analysis. The section defines the variables to be used in the analysis and provides a rationale for their selection. The section reports the results of the full discriminant analysis and closes with the results of a practical analysis using fewer variables.

Discriminant Analysis is a technique used to predict membership in a group based on a set of predictor variables and one criterion variable [Hube 89]. The analysis presumes that the predictor variables are normally distributed, not highly cross-correlated, and that no predictor variable is a linear combination of other predictor variables [Klec 80]. The next few paragraphs will discuss the predictor variables with respect to these criteria.

All predictor variables, biographical data and competencies, were analyzed for cross-correlation. Table 4.26 lists the correlation values for statistically significant correlations at the 0.05 level which have a correlation coefficient greater than 0.6. A correlation at the 0.6 level explains 36% of the variance in one variable through the use of the other variable.

TABLE 4.26 Cross-Correlations of Predictor Variables[1]

(n = 121)

	AGE	YEARS IN SOFTWARE AT COMPANY	TOTAL YEARS IN SOFTWARE	TOTAL YEARS EXPERIENCE	MATH DEGREE?
AGE	1.000		0.638	0.766	0.766
YEARS IN SOFTWARE AT COMPANY		1.000	0.883	0.735	
TOTAL YEARS IN SOFTWARE			1.000	0.791	
TOTAL YEARS EXPERIENCE				1.000	
MATH DEGREE?					1.000

	#LANGUAGES / LOW SKILL	#LANGUAGES / HIGH SKILL	#LANGUAGES TOTAL
#LANGUAGES / LOW SKILL	1.000		0.656
#LANGUAGES / HIGH SKILL		1.000	0.728
#LANGUAGES TOTAL			1.000

	DEGREES COMPLETED	HIGHEST DEGREE HELD
DEGREES COMPLETED	1.000	0.828
HIGHEST DEGREE HELD		1.000

1. All correlations are significant at the 0.05 level or better.

Table 4.26 demonstrates a clustering of age and experience variables which are highly intercorrelated. This is to be expected as those engineers who are older are much more likely to have more experience. It is presumed that experience rather than age is the important variable here. Since not all of these variables can be used in the subsequent discriminant analysis, *Total Years in Software* is selected as the most appropriate variable. It is consistent with the choice made in prior literature [Chry 78]

and is the most intuitively appealing. Hence *Age, Years in Software at the Company,* and *Total Years of Experience* will not be used. The *Math Degree?* variable was only correlated with age. Since age will not be used in the analysis, the math degree variable can be used.

There is also a high correlation between the *Number of Degrees Completed* and the *Highest Degree Held.* Again, this is natural as engineers who earn subsequent degrees tend to receive higher degrees. The *Highest Degree Held* is selected as the most natural choice between the two variables.

As one would expect, *#Languages Total* correlates both with *#Languages / High Skill* and *#Languages / Low Skill.* Hence this analysis will use the total languages variable.

None of the 38 competency variables were correlated with each other or with the biographical variables at a level of 0.60 or better. Hence all will be used in the subsequent discriminant analysis.

Table 4.27 reports the variables from Phase 2 which violate the normality criterion of plus or minus 1.00 levels of skew or kurtosis. The gender skew is a reflection of the underlying population and hence will be included in further analysis. The number of degrees partially completed is dropped due to its distribution and due to the fact that it represents a response from only 26 of the sample of 129.

TABLE 4.27 Non-Normal Variables (Phase 2)

VARIABLE	SKEW	KURTOSIS	n
Gender	-1.385	-0.840	128
# Degrees Partially Completed	1.698	1.724	26[1]
Training Hours	2.196	5.490	125
# Languages - High Skill	1.519	4.141	125
# Languages - Low Skill	1.432	3.703	125
# Languages - Total	1.084	0.844	125
Years Software at Company	1.028	0.996	123
Years Not Software at Company	2.345	4.729	33[2]
Years Software Not at Company	2.421	5.961	372
Years Not Software Not at Company	5.663	37.470	142
Years Total in Software	0.991	1.068	123
Years Total Experience	1.000	2.142	125

1. Number of subjects with one or more **partially** completed degrees. Statistics are calculated on full sample of n= 129.
2. Number of subjects with non-zero number of years in this category. Statistics are calculated on the full sample of n= 123.

The training hours variable exhibits a strongly skewed distribution with its peak at the low end of training hours and with a long tail in its distribution at the high end. Training hours is an important variable to retain for further analysis. The fact that it is not normally distributed will underpower its statistics in further analysis. Hence the conservative approach is to include the variable since it will only enter the discriminant function if it is a significant effect.

The total number of languages known variable had already replaced the low and high skill versions of the variable due to high cross-correlations. Also, the total number of languages known exhibits much better skew and kurtosis characteristics.

The set of experience variables listed exhibit non-normal characteristics. This is especially true of the variables for which relatively few subjects reported non-zero values. Since there is also a high intercorrelation among these variables, the *Years*

Total in Software variable is selected as the stand in for all other experience variables. It has the best skew characteristic and corresponds most closely with the variables used in the literature.

The variables remaining in the analysis at this point are listed in Table 4.28. The variables listed in Table 4.28 were entered into a stepwise discriminant analysis using Wilks' method. A total of 122 cases were used in the analysis. The results of the 24 step process are summarized in Table 4.29.

TABLE 4.28 Retained Variables for Discriminant Analysis

VARIABLE
Gender
Highest Degree Held
Computer Science Degree?
Engineering Degree?
Math Degree?
Training Hours
Total Years in Software
Total Number of Languages
Item #1 *thru* Item #38

Table 4.29 shows that 49% of the variance (1 minus Wilks' Lambda) can be explained by the 20 variables remaining in the Canonical Discriminant Function following the analysis. The more significant result is demonstrated in Table 4.30. Here we see that the function composed of the 20 variables in Table 4.29 is able to correctly classify over 86% of the cases collected in this study.

TABLE 4.29 Full Discriminant Analysis - Summary Table

(n = 122)

Step	Action Entered	Removed	Vars In	Wilks' Lambda	Sig.	Label
1	ITEM#3		1	.83854	.0000	Helps Others
2	SW_TOT		2	.76331	.0000	Total Years Software Experience
3	ITEM#31		3	.72147	.0000	Driven by Bias for Action / Urgency
4	LANG_TOT		4	.69362	.0000	Total Languages used Professionally
5	ITEM#27		5	.67090	.0000	Willingness to Confront Others
6	ITEM#25		6	.64644	.0000	Exhibits and Articulates Strong
7	ITEM#28		7	.62845	.0000	Perseverance
8	ITEM#23		8	.61493	.0000	Driven by a Sense of Mission
9	ITEM#16		9	.59993	.0000	Responds to Schedule Pressure
10	MATH_DEG		10	.58924	.0000	Math Degree Held?
11	ITEM#4		11	.57852	.0000	Uses Prototypes to Assess Design
12	ITEM#12		12	.56804	.0000	Schedules and Estimates Well
13	ITEM#38		13	.55790	.0000	Maintains "Big Picture" View
14	ITEM#9		14	.55016	.0000	Uses Structured Techniques for Comm
15	ITEM#1		15	.54255	.0000	Team Oriented
16	ENG_DEG		16	.53526	.0000	Engineering Degree Held?
17	ITEM#18		17	.52980	.0000	Takes Pride in Quality and Productivity
18	TRAINING		18	.52393	.0000	Total Training Hours
19	ITEM#13		19	.51692	.0000	Uses Code Reading to Ensure Final
20	ITEM#15		20	.51101	.0000	Focuses on User or Customer Needs
21		ITEM#38	19	.51579	.0000	Maintains "Big Picture" View
22		ITEM#4	18	.51986	.0000	Uses Prototypes to Assess Design
23	ITEM#5		19	.51465	.0000	Writes/Automates Tests in Parallel
24	ITEM#10		20	.50901	.0000	Uses Methodical Problem Solving App

TABLE 4.30 Full Discriminant Analysis - Classification Results

Actual Group	No. of Cases	Predicted Group Membership 0	1
Group 0 NON-EXCEPTIONAL	83	73 88.0%	10 12.0%
Group 1 EXCEPTIONAL	40	7 17.5%	33 82.5%

Percent of "grouped" cases correctly classified: 86.18%

Classification Processing Summary

125 Cases were processed.
0 Cases were excluded for missing or out-of-range group codes.
2 Cases had at least one missing discriminating variable.
123 Cases were used for printed output.

As a practical refinement, the discriminant analysis was rerun over the same variables but only allowing the first 10 variables of Table 4.29 to enter the Canonical Discriminant Function. This was an attempt to create a more tractable predictor

function which could be more readily used in practice. The full discriminant analysis presented in Table 4.29 was analyzed to recognize that after the first 13 variables entered the discriminant function, the subsequent variables explained less than 1% of the remaining variance. This was deemed as a practical cutoff for additional variables. Further, the eleventh and thirteenth variables entered (*Item#4and Item#38*) were subsequently removed from the full analysis. This was taken as an indication that the eleventh, twelfth, and thirteenth variables were not important to retain for further analysis. Hence the analysis was stopped with only the first ten variables allowed to enter the Canonical Discriminant Function. Table 4.31 provides the classification results for the reduced case of 10 variables. The function of 10 variables is able to correctly classify over 81% of the cases collected in this study. This means that, practically speaking, this ten variable function is as valuable as the full twenty variable function.

TABLE 4.31 Limited Discriminant Analysis - 10 Variables

Classification Results

Actual Group	No. of Cases	Predicted Group Membership 0	1
Group 0 NON-EXCEPTIONAL	83	68 81.9%	15 18.1%
Group 1 EXCEPTIONAL	40	8 20.0%	32 80.0%

Percent of "grouped" cases correctly classified: 81.30%

Classification Processing Summary

125 Cases were processed.
0 Cases were excluded for missing or out-of-range group codes.
2 Cases had at least one missing discriminating variable.
123 Cases were used for printed output.

Figure 4.2 shows the explained variance for the discriminant analysis using the ten variables identified. The total variance explained by these variables is 41%. The *Helps Others* competency explains 16% of the variance in the sample. The *Total Years Software Experience* variable explains another 8% of the variance in the sample. The *Bias for Action* competency explains 4% of the variance of the sample. All other variables explain less than 3% of the variance of the sample.

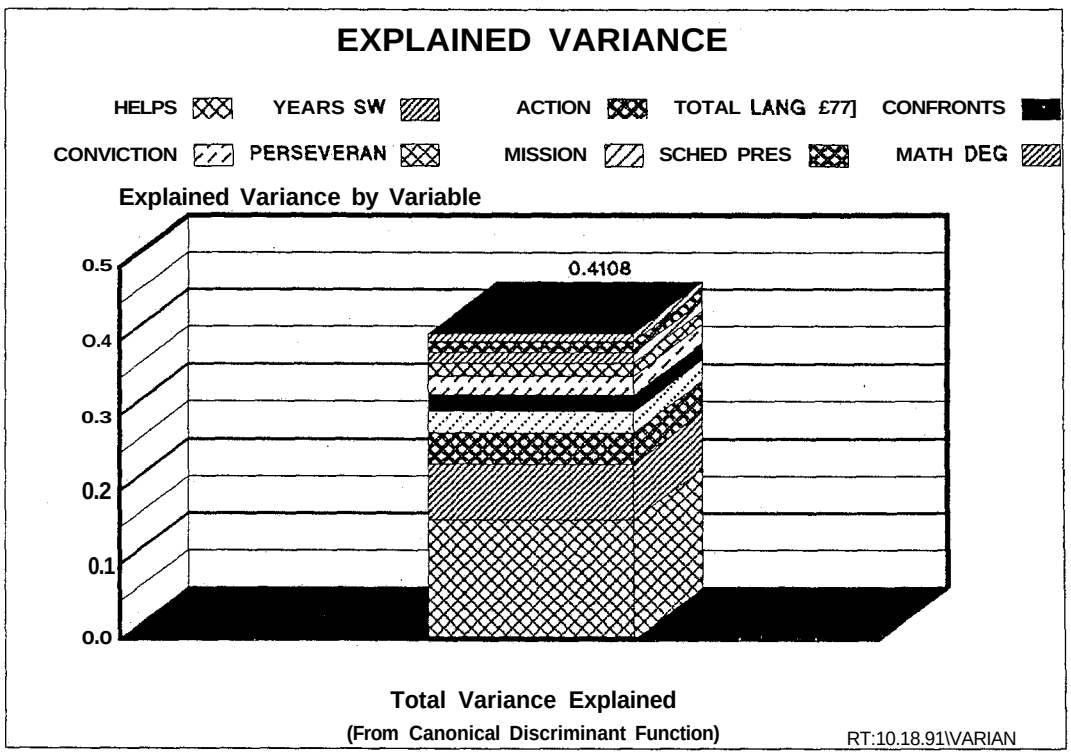

FIGURE 4.2 Explained Variance of 10 Variable Discriminant Function

This section presented the results of the Discriminant Analysis of the study variables. The section discussed the criteria for including variables in the analysis and presented the rationale for limiting the variables considered. The section presented the results of the full discriminant analysis resulting in a Canonical Discriminant

Function of 20 variables capable of correctly classifying over 86% of the cases used in the study. Finally, the section presented a restricted Canonical Discriminant Function with only ten variables capable of correctly classifying over 81% of the cases in the study.

4.2.5 Summary of **Phase 2 Data**

The Phase 2 portion of this research has collected data from over 120 subjects all of whom are experienced professional software engineers engaged in the creation of software products. The biographical portion of this data describes a well educated and experienced sample from the population at large. The only biographical data demonstrating a statistically significant relationship to exceptional performance are years of experience variables. These experience variables, along with age, show a high degree of cross-correlation and are replaced with the single variable of years of software engineering experience in further analysis.

The analysis of the competency scores from the Q-Sort exercise shows nine competencies are statistically related to performance under univariate analysis. Five of these competencies are more related to the behavior of exceptional performers while four of the competencies are related to non-exceptional performers. When using the multivariate technique of discriminant analysis, the research discovered that an equation of twenty variables is able to correctly classify the exceptional and non-exceptional cases under study 86% of the time. A simplified equation of only ten variables provides correct classification 81% of the time.

4.3 Summary of Data Presentation

Chapter 4 presented the data collected from both Phase 1 and Phase 2 of this research. The data are triangulated in time since Phase 1 occurred approximately nine

months before Phase 2. The data are triangulated by method. Phase 1 used a Critical Incident Interview technique coupled with Protocol Analysis to collect and identify a set of competencies related to exceptional performance of software engineers. Phase 2 verified these competencies using Q-Methodology to rank and weight the importance of these competencies. Finally, the data are triangulated by source since both engineers and managers were used in Phase 1 to identify competencies. These three triangulation methods led to a strong validation of the identified competencies as being very important to professional software engineers.

Both Phase 1 and Phase 2 also collected biographical information about the sample population in an attempt to identify any simple predictors of performance. The descriptive information about these samples was quite consistent across the two phases. Each sample proved to be generally male, well educated, and well experienced. In both phases, experience did emerge as a differential attribute associated with exceptional performance. Thus, more experience seems to lead to higher performance. This is certainly not a counterintuitive result.

When looking at all the factors associated with exceptional performance, experience did not emerge as the predominant predictor. The discriminant analysis of all predictor variables showed that although experience was a factor, there are many other factors associated with high performance. It is the combination of the identified competencies with experience which proved able to explain the most difference between the exceptional and non-exceptional groups.

CHAPTER 5

FINDINGS AND CONCLUSION

So he spent most of his time submerged in chaos,
knowing that the longer he put off setting into a
fixed organization the more difficult it would become.
But he felt sure that sooner or later some sort of format
would have to emerge and it would be a better one for his having waited.[14]

This chapter presents an analysis and interpretation of the data reported in Chapter 4. This chapter draws conclusions from the analysis of the data and interprets the results in light of the literature in this field. This chapter first discusses the *univariate* statistical analysis and attempts to draw conclusions from the analysis of each of the predictor variables in isolation. Then the chapter considers *multivariate* statistical analysis and discusses the models proposed for predicting whether individuals are exceptional or non-exceptional performers. The chapter closes with a proposed dynamic system model of performance based on the competencies identified in this research and on the level of an engineer's experience.

5.1 Univariate Analysis

Table 4.23 presented the results of a differential analysis of the biographical questionnaire data. The table shows that three variables, *Years at Company in Software*, *Total Years in Software*, and *Total Years Worked*, are all related to exceptional performance with their calculated significance levels from a 2-tailed t-test each less than 0.05. Table 4.26 indicates a high degree of intercorrelation among these same variables. This indicates that there is basically one concept at work in the relationship

[14]Robert M. Pirsig, *Lila: An Inquiry into Morals*, Bantam Books, NY, 1991.

between these experience variables and performance. This result is consistent with the result found by Chrysler [Chry 78] in his study of professional software developers. Chrysler found that several experience variables and age were all highly correlated.

This dissertation concludes that experience is indeed a significant predictor of performance. This is particularly true when the experience is in software engineering and the experience is received at the company where a subject still works. It seems that either companies reward the experience at their own company more, or the experience at the company is more relevant to the tasks of that company. Hence this experience is more valuable.

The experience variable by itself is not a satisfying predictor of performance. Operating by itself it is only able to correctly classify 63% of the 123 cases from this study. Although this is 13% more than the 50% correct classification provided by chance, it is not too powerful a result. Although experience is important, it is not enough. No other biographical variables are simply related to performance. These other variables will become important again as multivariate statistics are considered.

The competencies are identified and reported in Table 4.24. The competencies can be further organized into four categories, *Task Accomplishment, Personal Attributes, Situational Skills,* and *Interpersonal Skills* as shown in Table 5.1. The competencies in each category of this table are listed in rank order based upon the mean competency score for the entire sample.

The categories shown in Table *5.1* form natural clusters of related competencies. *Task Accomplishment* competencies are those competencies most closely related to the unique skills or capabilities required to complete the task at hand. Many of the competencies are specific to the production of software, for example, *Leverage/Reuses Code, Mastery of Skills and Techniques, Uses New Tools or Methods,* and *Uses Code*

TABLE 5.1 Competencies by Category

Task Accomplishment	T-Test[1]	Discrim[2]
Leverages/Reuses Code		
Uses Methodical Problem Solving		
Mastery of Skills and Techniques	**XP**	
Write/Automates Tests in Parallel with Code		
Prior Experience		
Obtains the Necessary Training		
Uses Code Reading to Ensure Final Code Quality		
Uses New Tools or Methods		
Schedules and Estimates Well		
Uses Prototypes to Assess Design		
Possesses Unique Domain Knowledge		
Uses Structured Techniques for Communication		
Personal Attributes		
Driven by Desire to Contribute	**NXP**	
Pride in Quality and Productivity		
Enjoys Challenge of Assignment - Sense of Fun		
Stresses Solutions over Source of Solutions - Lack of Ego		
Perseverance		**NXP**
High Personal Expectations and Goals		
Takes Initiative to Identify Ways of Completing Important Tasks		
Maintains "Big Picture" View	**XP**	
Driven by a Bias for Action and a Sense of Urgency		**XP**
Methodical, Organized, and Cautious		
Driven by a Sense of Mission		**XP**
Exhibits and Articulates Strong Beliefs and Convictions	**XP**	**XP**
Mixes Personal and Work Goals		
Proactive Role with Management	**XP**	
Situational Skills		
Concern for Reliability and Quality		
Focuses on User or Customer Needs		
Strong Analytic Skills - Ability to Think and Visualize		
Emphasizes Elegant and Simple Solutions		
Innovative		
Pays Close Attention to Detail		
Uses Decompositions Design Techniques		
Responds to Schedule Pressure by Sacrificing Parts of the Design Process	**NXP**	**NXP**
Interpersonal Skills		
Seeks Help From Others	**NXP**	
Team Oriented		
Helps Others	**XP**	**XP**
Willingness to Confront Others	**NXP**	**NXP**

1. Entries indicate which competencies have statistically significant differences for exceptional and non-exceptional performers. Those competencies related to exceptional performance are marked XP. Those competencies related to non-exceptional performance are marked *NXP.*
2. Entries indicate which competencies entered the canonical discriminant function of ten variables. Those competencies related to exceptional performance are marked XP. Those competencies related to non-exceptional performance are marked NXP.

Reading to Ensure Final Code Quality. Other competencies in this category relate to , the knowledge required to accomplish the task. For example, *Obtains the Necessary Training* and *Possesses Unique Domain Knowledge* refer to the unique knowledge required for the particular task. Finally, the competency *of Prior Experience* relates to the degree to which familiarity with the task itself is important. One subject describes his view of the need for the ability to acquire new skills as, "It's so interdisciplinary, you've got to be willing to do anything and believe you can do it with training as well as the people who are already doing it."

Personal Attributes are those competencies which describe inherent traits of the individual. These are generally presumed to be competencies which are independent of the task itself. Many of these competencies relate to the personal motivation of the individual. For example, *Driven by a Desire to Contribute, Driven by a Bias for Action and a Sense of Urgency, Driven by a Sense of Mission,* and *High Personal Expectations and Goals* all relate to internal motivations and drive of the individual. Other competencies in this category refer to the generic behaviors of the individuals applied in all circumstances. For example, *Methodical, Organized, and Cautious, Takes Initiative to Identify Ways of Completing Important Tasks,* and *Proactive Role with Management* all refer to the actions an approaches of an individual in completing a task. One engineer described his struggle in learning which results were worth fighting for, "I need to be less stubborn on the things that don't matter so that I can save my wind for the ones that do matter."

Situational Skills are the competencies which relate to the process by which an individual completes a task. These competencies will be stressed or de-emphasised depending upon the nature of the current task. For example, *Focuses on User or Customer Needs, Emphasizes Elegant or Simple Solutions,* and *Uses Decomposition*

Design Techniques all relate to the basic approach an engineer takes in solving a particular problem. The competencies stress the *how* of solving a problem. As one engineer explains the difference between technical knowledge and customer needs, "The technical aspects are fairly clear for the kinds of products we do here at [the Company]. ... They're mostly challenging with respect to the design and customer usage."

Interpersonal Skills describe the competencies related to the interactions among engineers. The competencies *of Seeks Help from Other, Team Oriented, Helps Others,* and *Willingness to Confront Other* all describe the way engineers interact. One engineer described an essential skill as, "The ability to work with different people from different areas that have different concerns from mine and yet be able to work with them towards a common end."

This collection of competencies is a significant contribution in its own right. All competencies listed are important even if they prove to not be differential between exceptional and non-exceptional performers. The list is important because it describes all the competencies found in software engineers in this study. The list of competencies provides a well rounded view of the extent of skills, knowledge, and attributes required for a software engineer to be successful in their job.

Table 5.1 indicates which competencies are considered differential via the T-test. Those competencies which are differential are highlighted in bold. The T-Test column indicates whether the competency is associated with exceptional (XP) or non-exceptional (NXP) performance. Five competencies are associated with exceptional performance and four competencies are associated with non-exceptional performance via the t-test.

The competencies associated with exceptional performance, *Mastery of Skills and Techniques, Maintains "Big Picture" View, Exhibits and Articulates Strong Beliefs and Convictions, Proactive Role with Management,* and *Help Others* generally cluster around the theme of *external focus.* The exceptional engineer is differentiated by behaviors associated with externalization. The behaviors are directed at people or objects outside the individual. The exceptional engineer takes a broad view of situations and develops strong convictions about how to proceed. The exceptional engineer drives toward this vision by proactively working with management to set goals on directions for the team. The exceptional engineer also helps other engineers in an attempt to ensure the full success of the project. The one internal skill exhibited by exceptional engineers is *Mastery of Skills and Techniques.* This is a more self-directed competency and reinforces the fact that engineers need to be completely capable in their own discipline before they achieve the exceptional status related to an external focus. One engineer in the study states, "My perception of someone who is successful is not someone that knows the most, it is someone who can use the knowledge they do have the best."

The non-exceptional engineer is associated with four competencies, *Driven by a Desire to Contribute, Responds to Schedule Pressure by Sacrificing Pans of the Design Process, Seeks Help from Others,* and *Willingness to Confront Others.* Here the unifying theme is one *of internal focus.* These competencies all relate an individual acting largely alone attempting to complete tasks. The interaction with others is either one of seeking help or one of confrontation. These engineers find that they give in to the external schedule pressure and sacrifice parts of the design process that they would rather not sacrifice. The motivation of the non-exceptional engineer comes from a personal desire to contribute. This contrasts with the exceptional engineer who takes the broader view and works to influence project direction.

One way of viewing these characteristics is to place them in the context of experienced versus inexperienced individuals. Many of the competencies related with non-exceptional performance can be viewed as the behaviors of an inexperienced engineers. When an engineer first begins a career, they will be unsure of their skills and capabilities. As a result, they will concentrate heavily on their own performance and exhibit an internal focus. As they mature in the job and become more confident of their skills, they will begin to take a broader view and be more proactive in setting project direction. This explains the fact that experience was shown to be differential characteristic of exceptional performers.

The relationship between experience and the competencies exhibited by exceptional performers does not explain all of the difference in the sample, however. Many experienced software engineers never become exceptional. These experienced engineers fail to exhibit the externally focussed competencies even after many years of experience. It would seem that if there is a relationship between experience and certain key competencies, that the mechanism by which experience reinforces or transfers the key competencies doesn't work for all individuals. This raise the question of how competencies are reinforced. Why do some software engineers use their experience to develop the competencies associate with exception performance while others do not? This question is beyond the scope of this dissertation, but indicates a significant direction for future research.

This section has described the results of univariate analysis of biographical and competency variables. Experience emerged as the single biographical variable associated with exceptional performance. Nine competencies are differentially related to performance. The full set of 38 competencies fall naturally into four categories: *Task Accomplishment, Personal Attributes, Situational Skills,* and *Interpersonal Skills.*

The competencies related to exceptional performance are also related to experience. The competencies related to non-exceptional performance are indicative of an inexperienced individual. Thus the experience variable is itself related to the competencies which differential exceptional from non-exceptional performance.

5.2 **Multivariate Analysis**

Table 4.29 presented the results of a stepwise discriminant analysis allowing all retained variables to enter into a single Canonical Discriminant Function. As shown in Table 4.30, this equation of 20 variables is able to correctly classify over 86% of the 123 cases used in the study. In an effort to make the results more tractable, the analysis was rerun with the same variables but only allowing the first 10 variables to become part of the Canonical Discriminant Function. This simplified equation of 10 variables is still able to correctly classify 81% of the 123 cases studied as shown in Table 4.31.

The three biographical variables which entered the Canonical Discriminant Function of 10 variables are *Total Years of Software Experience, Total Languages Used Professionally,* and *Math Degree Held?.* The Total Years of Software Experience and Math Degree Held? variables are of the same sign indicating that each is associated with exceptional performance. The inclusion of the experience variable is consistent with the univariate analysis above. It is also consistent with the literature [Chry 79] and with intuition. The entry of the Math Degree Held? variable is somewhat surprising and did not appear in any prior literature. Since Math Degree Held? did not emerge as differential via the t-test, it may not be strongly correlated with exceptional performance. Rather it may be one of many variables capable of explaining a significant amount of variance at that particular point in the stepwise discriminant analysis. This variable may be a stand-in for any variable related to education.

The Total Languages Used Professionally variable entered the Canonical Discriminant Function with a different sign indicating that it is related more to non-exceptional performance. This is inconsistent with prior studies which show a correlation between the number of languages known and performance. In this dissertation the subjects were asked to identify those languages they had actually used professionally. Other studies asked which languages the subject was familiar with. This dissertation result suggests that in-depth knowledge and use of fewer languages is related to higher performance.

The seven competencies which entered the limited discriminant analysis function of 10 variables are highlighted in bold in Table 5.1. The Discrim column indicates XP or NXP to indicate if the competency is related to exceptional or non-exceptional performance in the Canonical Discriminant Function. The competencies which entered the Canonical Discriminant Function related to exceptional performance are *Driven by a Bias for Action and a Sense of Urgency, Driven by a Sense of Mission, Exhibits and Articulates Strong Beliefs and Convictions,* and *Helps Others.* The competencies which entered the Canonical Discriminant Function related to non-exceptional performance are *Perseverance, Responds to Schedule Pressure by Sacrificing Parts of the Design Process,* and *Willingness to Confront Others.*

The results of this discriminant analysis could be used for software engineer selection. The biographical variables of experience, languages used, and math degree are easily obtained from the individual. Behavior based interviewing can be used to extract a score on each of the seven competencies used in the analysis. The Canonical Discriminant Function can then be used to predict the individual's performance.

Perhaps a better use of this result is to use it as a tool to improve performance of existing software engineers. Once engineers know what differentiates performance they can modify their behaviors in order to achieve better results. The assumption here is that most of the identified competencies are exhibited by all engineers to some degree. By emphasizing and reinforcing the competencies exhibited by the exceptional performers, the performance of all software engineers can be improved.

Four of the competencies selected by discriminant analysis were also among the nine competencies identified as differential via the t-test. This is one indication of the consistency of the results. The cluster of competencies for exceptional and non-exceptional performance by the discriminant analysis continue to reinforce the *internal/external focus* of the competencies on a differential basis. The competencies related to exceptional performance continue to relate to overall project or team results. The competencies associated with non-exceptional performance still relate to the individual and the individual's performance.

One further insight can be gleaned from Table 5.1. With two exceptions, all of the competencies selected as differential by the t-test and the discriminant analysis fall into the broad categories of *Personal Attributes* and *Interpersonal Skills*. This is a very significant result. It indicates that the most important characteristics leading to exceptional performance are not skills associated with the task. Rather, personal characteristics and interpersonal interactions are the things that differentiate performance. Thus skills unique to the person prove to be the most important. One study participant notes, "I have not seen one project fail because the engineers lacked technical knowledge." Another participant says, "I'm a lot more willing not to question somebody's judgement or talent and just let them do it because I have to understand that they're a talented person too, and they're going to do the right thing as opposed

to before, thinking my way." A further study participant notes, "It's interesting, you could get into the attributes of the personal, because I think that is the major key, actually, ... that is basically one of the major keys as to whether something's going to work, partly the interaction personalities, but partly the mindset, the attitude about which we approach the project with." These quotes indicate the understanding on the part of study participants of the importance of personal and interpersonal skills and attributes.

The primacy of personal and interpersonal skills and attributes is further reinforced by reviewing the competencies suggested by managers as representing their exceptional performers. Here competencies like *Breadth of View and Influence, Leadership, People Skills and Teamwork,* and *Positive Attitude* reinforce the notion that the external focus of the engineer is critically important to achieve exceptional performance. One manager described these critical attributes as, "Big picture seen, overall architect, expertise that is sought, excellent technical judgement, self-starter, reliable, accurate schedules, pleasant surprises." These are the sorts of characteristics that managers use to rate an engineer's performance.

This discussion indicates that the important competencies for software engineers may be largely task and situation independent. If this is true, the results contained within this dissertation would be very generalizable. The results could apply to other software engineers not part of this study. The results may apply to other engineering or even non-engineering disciplines. Further study is required to determine if this is indeed the case.

The multivariate analysis shows a relationship to the univariate analysis. The experience factor continues to be as important in the multivariate analysis it is in the univariate analysis. Two additional biographical competencies enter the 10 variable

Canonical Discriminant Function: *Total Languages Used Professionally* and *Math Degree Held?*. Four of the seven competencies which enter the discriminant function are also differential on a univariate analysis. The differential competencies from univariate and multivariate analysis cluster in the *Personal Attributes* and *Interpersonal Skills* categories indicating that performance is not differentially related to task or situational skills.

The earlier discussion on the purpose of the biographical questionnaire posed three hypotheses to be considered in this dissertation. The first hypothesis was that exceptional engineers would make broader use of various tools or methods, especially acting as early adopters of new tools and techniques. The data in this research failed to support this hypothesis. The Phase 1 survey of tools and methods was inconclusive. Further, although exceptional engineers did differentially exhibit *the Mastery of Skills and Techniques* competency, they did not differentially exhibit the *Uses New Tools or Methods* competency. This indicates that exceptional engineers are masterful with the tools they do use, but they are not more likely to be early adopters of new tools.

The second hypothesis was that academic and demographic factors would not emerge as predictors of performance. This proved to be true in this research. No demographic factor other than experience was differentially related to performance. The third hypothesis was that age and experience would correlate with performance, but would emerge as a threshold effect where they would not be significant beyond a certain point. This also proved true. The cluster of experience variables were differentially related to exceptional performance. However, experience alone was not

an extremely good predictor of performance. Thus experience is important to exceptional performance, but although it is necessary for exceptional performance it is not sufficient.

5.3 Dynamic System Model of Performance

Figure 5.1 presents a dynamic system model of performance[15]. The model is based on the notion that competencies, experience, and performance are all related by a set of cause and effect relationships. The model also accounts for the feedback inherent in the system and helps to explain the interrelationship of experience and the competencies.

The left side of the model lists the competency categories associated with software engineers. The categories *of Personal Attributes and Interpersonal Skills* are highlighted since they represent the differential competencies for software engineers. The center of the model defines the broad category of experience. It is composed of the differential experience variables of *Years at Company in Software, Total Years in Software,* and *Total Years Worked.* The right side of the model defines the measured output to the system. It is the performance category assigned to the individual software engineer.

The system starts with a new software engineer at the beginning of the career. This person has set of competencies defined by prior experience. Largely this is the training the person received in school. The competencies are directly related to the behavior of the individual. It is these behaviors that begin the process of gaining experience.

[15]Model proposed by Professor Charles O. Neidt.

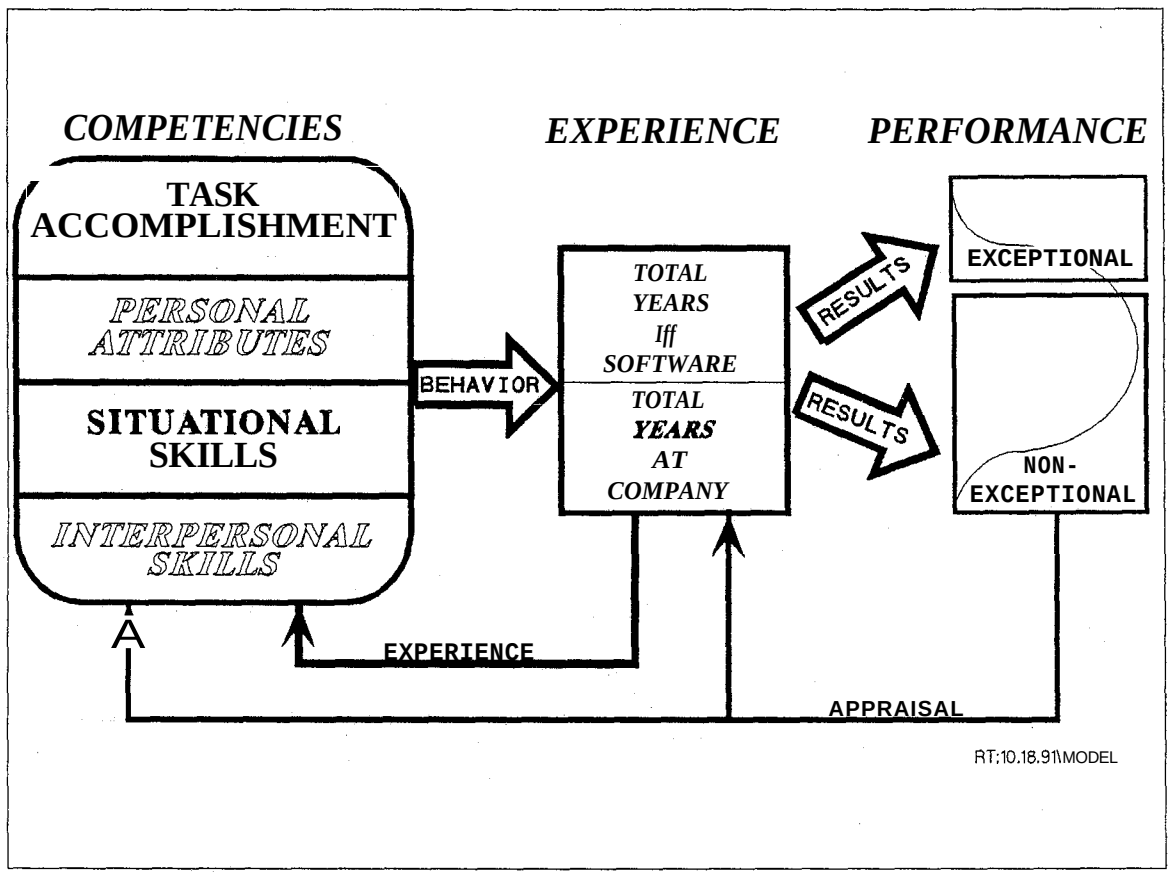

FIGURE 5.1 Dynamic System Model of Performance

Experience corresponds to applying behaviors in order to achieve results. As an engineer gains experience in the employer's company and in the field of software engineering, they begin to achieve their results. The very process of gaining this experience allows for many opportunities to improve upon behavior patterns in an attempt to become more effective. This experience forms the first feedback loop. As an engineer finds behaviors which help to achieve intermediate goals, these are

reinforced and developed. Behaviors which prove to be unsuccessful in achieving intermediate results are modified until the correct results are achieved. Thus the actual experience of attempting to complete tasks forms important, behavior modifying feedback. The significance of this feedback was not lost on one engineer who described his experience.

> *I don't think that all Software engineers are equal, you know, some have more experience, and you definitely have to weigh that. Some people have more experience with tools, you have to weigh that. Some people have more self confidence and can put across an aura of knowing what they're talking about when they don't. And if you do treat everyone equally, then you can fall into the trap of listening, perhaps listening to the wrong person or discounting someone who really has something reasonable to say.*

The right side of the model shows the appraisal portion of the model. Here results are evaluated relative to some measure, and the engineers performance is classified as either exceptional or non-exceptional. This appraisal forms the second important feedback loop. The feedback affects experience since the appraisal is used to select future tasks for the engineer. Very poor performance might mean no more experience at all! Very good performance generally results in even more challenging tasks. The appraisal also provides feedback to the competencies. An engineer will modify their own behaviors as a result of the appraisal feedback. This serves to enhance or deemphasize certain competencies.

This model is valuable in that it allows for discussion of the process by which exceptional performance is created. It also allows for discussion of why individuals with similar years of experience continue to exhibit different levels of performance. In some cases the very nature of the experience may be vastly different. Although two engineers may have similar years in a similar job, the actual work assignment may have been very different. Failure to achieve the expected results on an early assignment

may lead to future assignments which fail to develop the required competencies for exceptional performance. This is because management will not give the more challenging assignments to those who have not achieved exceptional results in the past. This severs the important feedback of experience to competencies. The value of a positive experience is highlighted by one engineer in the study who described the normal method of putting new, inexperienced engineers into mainstream project assignments.

> *Some people might say that that's good, it's trial by fire, personally I think that tends, if they do, you give them the opportunity to succeed or fail If they do succeed, I mean you're going to have a tiger on your hands, and that's great when that happens, but if they fail, you're going to have somebody that's going to be demoralized and not feel good about themselves when the mistake might have been more in just fitting the person with the job.*

Engineers may also fail to develop their competencies if the appraisal feedback loop is broken. This can occur either when management fails to provide adequate performance appraisal or when the individual engineer fails to make proper use of the appraisal data. In either case the engineer fails to develop and reinforce the competencies required for exceptional performance.

The Dynamic System Model of Performance developed in this section provide some additional insight into the relationship between experience factors and competencies in achieving exceptional performance. The model illustrates two important feedback loops. Experience is fed back to the competencies and should be used to reinforce the essential competencies required for exceptional performance. Performance appraisal is fed back in two ways. First, it modifies the individual's assignments and hence alters the experience. Second, the appraisal feedback modifies an individual's competencies and behaviors. Failure to achieve an exceptional level of performance may be explained by failure in these feedback loops.

5.4 Summary

This chapter has described the results of this dissertation. Figure 5.2 presents a summary of the results. The chapter discussed univariate and multivariate analysis of the data. Experience emerged as the sole biographical variable related to exceptional performance. There was no simple predictor of performance. Thirty eight competencies of software engineers were identified. Nine of the competencies are differential with five of them associated with exceptional performance and four of them associated with non-exceptional performance. The competencies are placed into one of four categories: *Task Accomplishment, Personal Attributes, Situational Skills,* and *Interpersonal Skills.* The combination of univariate and multivariate analysis of the competencies led to the insight *that Personal Attributes and Interpersonal Skills* are most closely linked with performance differences. Skills associated with the task or situation did not generally emerge as differential. A practical equation of ten variables was shown to correctly classify 81% of the 123 cases in the study. Finally, a dynamic system model of performance was proposed to explain the linkage between experience and competencies.

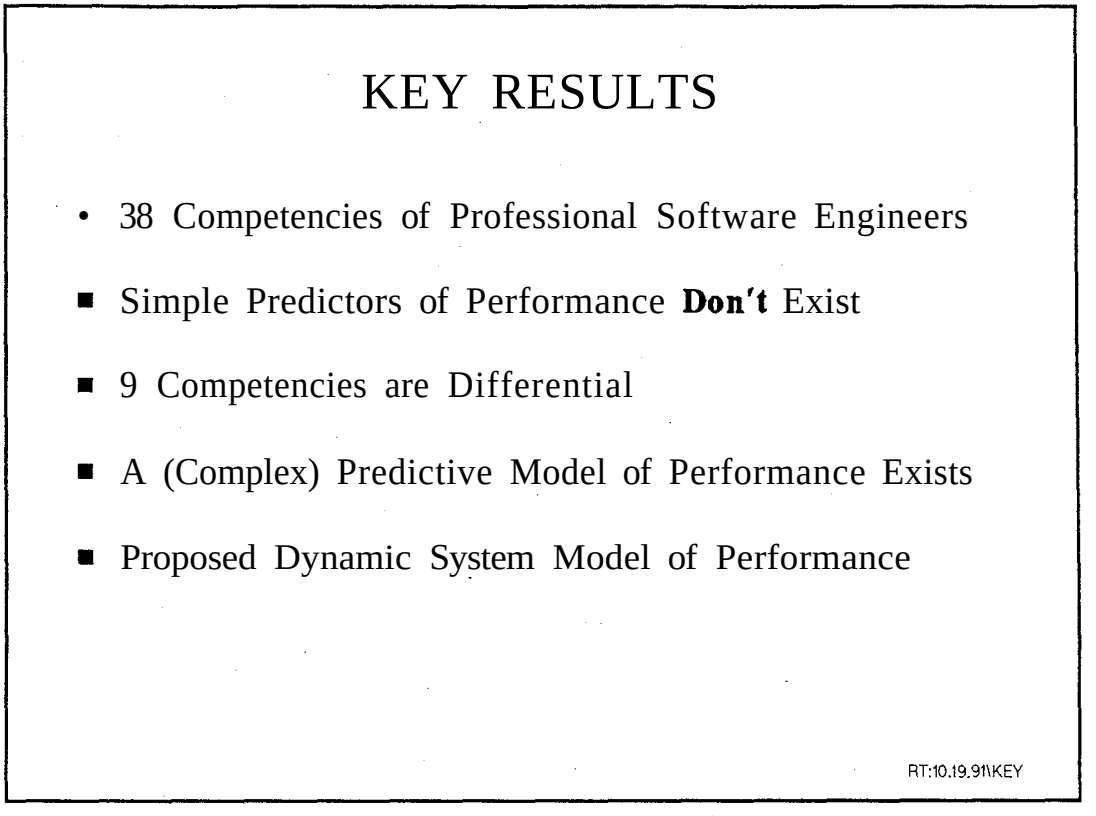

FIGURE 5.2 Key Research Results

CHAPTER 6

FUTURE DIRECTIONS FOR FURTHER STUDY

Get it right or let it alone.
The conclusion you jump to may be your own.[16]

This dissertation presents the results of an investigation of the competencies and demographics which contribute to the performance of professional software engineers. The research finds significant predictors of performance. The dissertation discussed the role of experience in performance. The dissertation also presented the 38 competencies associated with software engineering. Further, the dissertation presents the differential competencies associated with high performance. The research identifies the importance of *Personal Attributes* and *Interpersonal Skills* to exceptional software engineers. Finally, the dissertation presents a practical predictive model of performance capable of correctly classifying over 81% of the 123 cases in this study. All of these results are combined in the creation of the Dynamic System Model of Performance. This final chapter will discuss some of the implications of this research as well as its deficiencies. The chapter will suggest areas of further research.

Figure 6.1 presents a summary of the implications of this research. The dissertation presents implications for further research, education, and practice. The research reported provides a valuable set of 38 competencies which can form the basis of further studies into performance differences for software engineers. These 38 competencies express a broad range of behaviors required of a software engineer.

[16] James Thurber, *Further Fables for Our Time* (New York, 1956)

Since the competencies covered behaviors well beyond simple task accomplishment, it's clear that future research should be careful to consider these non-task skills in the design of future studies. One interesting result from this dissertation is that skilled subjects are able to complete complex research instruments without guidance. The Q-Sort is generally conducted with the researcher acting as a facilitator. In the present research, the Q-Sort was sent to the subjects and they completed, it on their own. Given that only two out of 129 subjects incorrectly completed the Q-Sort, it seems that complex tasks are feasible for research methods with highly skilled subjects. Further, the survey response rate of 47% demonstrates a surprising willingness on the part of the subjects to participate. This could be due to the level of interest that the subjects had in the area and their interest in the survey task itself.

In the area of education, this dissertation research concludes that educators need to stress the Interpersonal Skills and Personal Attributes as part of an engineer's preparation. Generally, education focuses on the tasks associated with the discipline under study. Although this task training is important, the differential skills are not task oriented. The educational process should support the development of interpersonal skills and personal attributes through the creation of learning situations which stress these competencies. Team-oriented assignments may be useful for developing these traits.

This dissertation research bears directly on the interviewing and hiring procedures in industry. This research suggests that behavioral interviewing stressing the differential traits identified may be the best way to locate exceptional employees. Note, however, that interviewing based on the differential competencies may be

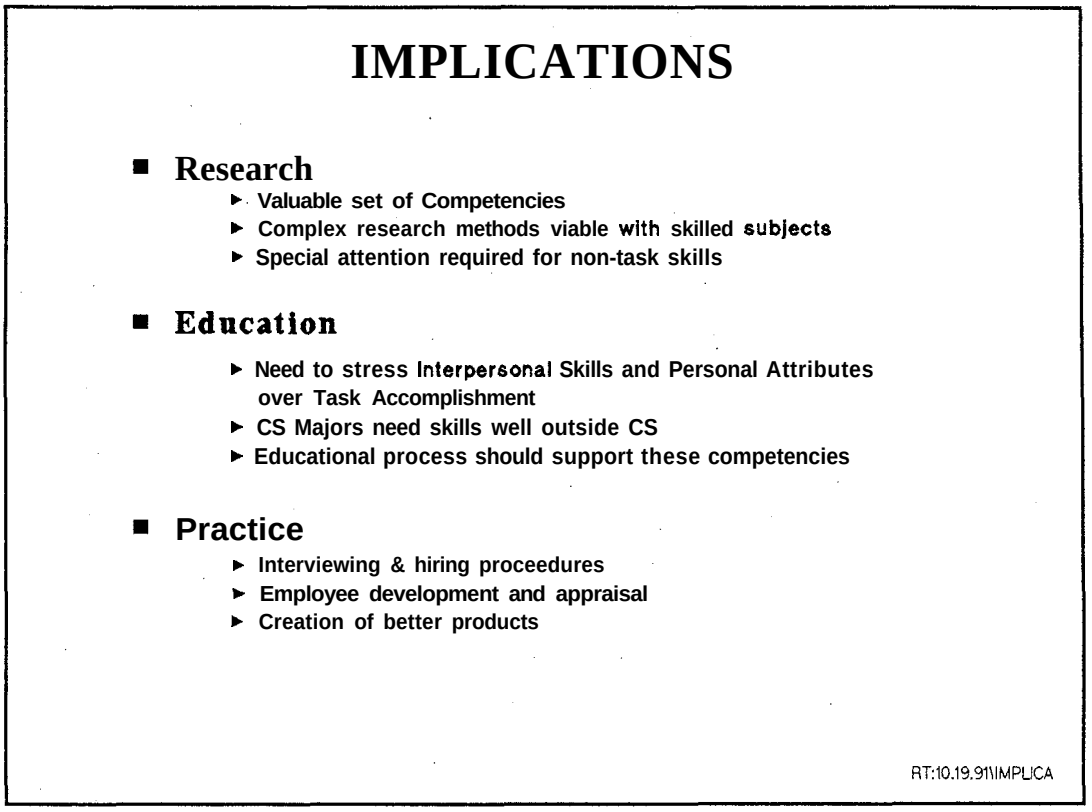

FIGURE 6.1 Research Implications

inappropriate for recruiting recent college graduates. Since the competencies were identified by interviewing experienced engineers, they may be different than the competencies of exceptional recent graduates.

The research results can also be used to improve the employee development and appraisal processes. The Dynamic System Model of Performance suggests the connection between appraisal and performance. One question raised in the discussion of the Dynamic System Model of Performance is the degree to which the feedback loops of *Experience* and *Appraisal* are active for non-exceptional engineers. Perhaps the failure that occurs for non-exceptional engineers is a breakdown of either the

experience or the appraisal feedback loop. If an engineer is unable to learn from their own experience and refine their own competencies based upon this experience, they will fail to make the required improvements in their own competencies. Without the refinement of these competencies, the engineer's behavior will not change. Without the behavior changes there is no change in results and no change in appraised performance.

The performance appraisal is the second feedback loop in the model. If this feedback fails to modify the engineer's emphasis on competencies, again there will be no behavior change. Failure in this feedback loop may be more significant since this also affects the actual assignments an engineer will receive. Management typically will only provide the most challenging assignments to the top performers. If an engineer fails to receive these challenging assignments, the very nature of their experience changes significantly. The engineer is not as challenged and the experience feedback received is not as valuable.

This discussion suggests that the two feedback loops of the Dynamic System Model of Performance are important paths for an engineer's development. Managers can assist in the development of engineers in two important ways. Managers can provide the challenging assignments which will give the engineer the opportunity to enhance their competencies. Managers can also provide appraisal feedback which reinforces the competencies shown to be associated with exceptional performance. If managers stress the value of and encourage the development of *Personal Attributes* and *Interpersonal Skills* the engineer will improve these competencies. By reinforcing the competencies associated with exceptional performance, the manager will also encourage the behaviors and results associated with exceptional performance.

Figure 6.2 summarizes the future directions for further research from this study. To a certain extent the results described in this dissertation may be thought of as a description of the Company and its culture as much as of the individual engineers. The classification method for exceptional and non-exceptional engineers was management selection. Although the selection system has an objective base, any performance appraisal ultimately has a high subjective component. This subjective component is a reflection of the individual doing the appraisal. It is also a reflection of the organization in which the manager resides. In this light it is interesting to note the behaviors associated with non-exceptional performance. In particular, the competencies of *Seeks Help from Others* and *Willingness to Confront Others* could be a reflection of the organization itself and its desire to suppress behaviors associated with these competencies. The culture of the organization may not allow for confrontation and may attempt to suppress it by associating it with low performance. Also, the culture may set the expectation that only non-exceptional performers need help from others. This concept of organizational impact on the observed competencies was not researched since the study population was from a single organization. The impact of the organization and its appraisal system on observed competencies deserves further study.

Future research should consider exploring the key cultural differences in the population. First, there are corporate cultures which encourage and discourage certain behaviors. Some companies value consensus and team approaches to problem solving. Other companies stress the value of individual efforts. Each culture studied could result in somewhat different competencies identified. Second, there are geographic differences even in the same company. Since each local region in the U.S. has certain differences, these differences will be reflected in the workplace. Further, the

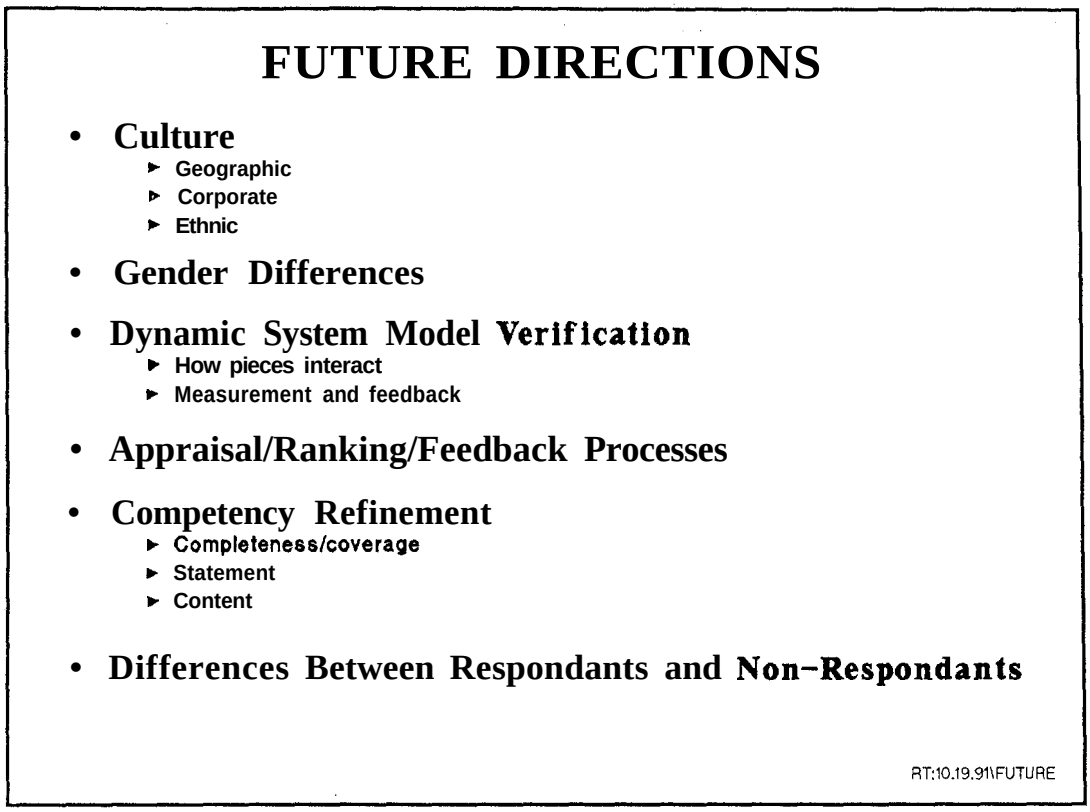

FIGURE 6.2 Future Directions for Further Study

management of each geographic location may develop its own subculture resulting in differences related to geography. Finally, there are vast differences due to ethnic cultures. Certain ethnic groups exhibit markedly different traits from other groups. For example, some groups may value conflict as a problem solving mechanism while other ethnic groups may suppress all conflict in favor of harmony. This will clearly result in different behaviors on the part of individuals.

The research performed in this dissertation considered gender only to the extent that it might be a predictor of performance. The research could be extended to consider differences in gender. One female study participant noted some of the potential gender differences in interacting with other engineers.

> *I have this idea that there's a gender-related trait. ... In general, the guys I work with are very focused on what they do, and if* [my male boss] *or someone comes and wants to solve a problem, wants me to help him solve a problem, he'll start asking me these specific questions that will lead up to the answer to the problem, and it's a very narrow corridor that he can think in, and a woman will tend to say, "Well, this is my problem. What are the things you can offer?" And it's a more broad type of way of looking at things.*

Future research could address gender issues and attempt to determine if competencies are gender related.

All the conclusions reached in this dissertation are based on the behaviors reported by study participants. The original Critical Incident Interviews required participants to describe what they did on their jobs. The Q-Sort exercise asked participants to sort the competencies on a scale from *Least Like My Behavior* to *Most Like My Behavior*. The presumption throughout has been that the subject actually did report upon their behaviors. The danger of this form of research is that participants might actually report competencies which they *value* rather than behaviors they actually exhibit. If this were the case in this study the conclusions would be somewhat different. If subjects reported behaviors they value rather than behaviors they exhibit, the link between exhibited behaviors and performance appraisal would be weakened. This research has relied on careful instruction in an attempt to ensure that subjects did report on behaviors *exhibited* rather than *valued.*

The competencies identified in Phase 1 of this research may not be complete in that they may not cover all the competencies exhibited by software engineers. In particular, a significant list of competencies was dropped from further study after Phase 1 since they had been mentioned by few Phase 1 participants. These and other competencies may be required to complete the set of competencies for software engineers. Also, the precise statement of competencies should be studied to ensure that they express the underlying competency. All competencies in this study were written in a attempt to make them *equally positive.* However, it seems clear that the *Responds to Schedule Pressure by Sacrificing Parts of the Design* competency was "valued" low due to its appearance at the bottom of the rank ordered list of competencies. All competency statements should be refined to ensure that they correctly and neutrally express the core idea. Finally, the content of each competency should be tested to ensure that there is common agreement across study participants as to the meaning of the competency. Each competency should have precisely the correct set of key behaviors associated with it.

As in all studies that allow for the voluntary self-selection of subjects, there is the concern that the subjects who chose to participate are different from those who did not choose to participate in some meaningful way. A future study should consider this issue and attempt to study both groups and look for differences.

This dissertation research suggests a set of replication studies to verify and expand its results. In order to claim any form of generalizability, the study should be replicated with software engineers from more than one company. There is a possibility that the results of this research are closely correlated with the culture and environment of the company studied. Thus replication at other companies, in other industries, with other physical environments are appropriate. The issue of application type also enters into

this discussion. This dissertation covers software engineers developing system software, application software, and embedded microprocessor software. In order to be generalizable, the results must be shown to be independent of application type. Preliminary study of the data from this research seems to indicate that these results are independent of application type. However, the sample sizes for each of the three application types was too small to draw any statistically based conclusions.

Another aspect of generalizability has to do with the phase of development in which the engineer is engaged. Most of the participants in this study were involved in the development of new software. Some were involved in the maintenance of existing products currently on the market. Without studying a broader range of software engineering activities, it will be impossible to tell if the results of this research will apply to them as well.

This dissertation reports significant results in the identification of the competencies of software engineers, in the predictive model for identification of exceptional software engineers, and in the creation of a Dynamic System Model of Performance. This chapter discussed extensions to this study that would provide significant extensions to the result. This dissertation represents a significant personal effort for an extended period of time. As one study participant noted about himself, "it's been five years, no wonder I'm sick of it!" I look forward to seeing the results that others are able to achieve building upon this research.

REFERENCES

[Basi 86] V. R. Basili, R. W. Selby, and D. H. Hutchens, "Experimentation in Software Engineering," *IEEE Transactions on Software Engineering*, Vol., SE-12, No. 7, pp. 733-743, July, 1986.

[Boeh 81] B. W. Boehm, *Software Engineering Economics,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.

[Boeh 83] B. W. Boehm, "Seven Basic Principles of Software Engineering," *The Journal of Systems and Software,* Vol. 3, pp. 3-24, 1983.

[Boeh 88] Barry W. Boehm, "Understanding and Controlling Software Costs," *IEEE Transactions on Software Engineering,* Vol. 14, No. 10, Oct. 1988.

[Brig 83] K. C. Briggs and I. B. Myers, *Myers-Briggs Type Indicator,* Consulting Psychologists Press, Inc., Palo Alto, CA, Seventh Printing, Form G.

[Broo 75] F. P. Brooks, *The Mythical Man-Month: Essays on Software Engineering,* Addison-Wesley Publishing Company, Reading, MA, 1975, *Reprinted with corrections, January 1982.*

[Broo 80] R. E. Brooks, "Studying Programmer Behavior Experimentally: The Problem of Proper Methodology," *Communications ofthe ACM,* Vol. 23, No. 4, pp. 207-213, April 1980.

[Broo 87] F. P. Brooks, Jr., "No Silver Bullet - Essence and Accidents of Software Engineering," *Computer,* Vol. 20, No. 4, pp. 10-19, April 1987.

[Buro 89] Buros Institute of Mental Measurements, *The 10th Annual Mental Measurements Yearbook,* University of Nebraska, Lincoln, 1989.

[Char 82] Charles River Consulting, "Job Competence Assessment: Defining the Attributes of the Top Performer," *American Society for Training and Development Research Series,* Vol. 8, 1982.

[Chry 78] E. Chrysler, "Some Basic Determinants of Computer Programming Productivity," *Communications of the ACM,* Vol. 21, No. 6, pp. 472-483, June 1978.

[Cohe 83] J. Cohen and P. Cohen, *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences,* Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, 1983.

[Curt 79] B. Curtis, S. B. Sheppard, and P. Milliman, "Third Time Charm: Stronger Prediction of Programmer Performance by Software Complexity Metrics," *Proceedings of the 4th International Conference on Software Engineering*, pp. 356-360, 1979.

[Curt 80] B. Curtis, "Measurement and Experimentation in Software Engineering," *Proceedings of the IEEE*, Vol. 68, No. 9, Sept. 1980.

[Curt 81] B. Curtis, "Substantiating Programmer Variability," *Proceedings of the IEEE*, Vol. 69, No. 7, p. 846, July 1981.

[Curt 86a] B. Curtis, "By the Way, Did Anyone Study Any Real Programmers?," in E. Soloway, S. Iyengar (Eds.), *Empirical Studies of Programmers*, Papers Presented at the First Workshop on Empirical Studies of Programmers, June 5-6, 1986, Washington DC, Albex, Norwood, NJ, p. 267,

[Curt 86b] B. Curtis, E. M. Soloway, R. E. Brooks, J. B. Black, K. Ehrlich, and H. R. Ramsey, "Software Psychology: The Need for and Interdisciplinary Program," *Proceedings of the IEEE*, Vol. 74, No. 8, August, 1986.

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

[Evan 89] G. E. Evans and M. G. Simkin, "What Best Predicts Computer Proficiency?," *Communications of the ACM*, Vol. 32, No. 11, pp. 1322-1327, Nov. 1989.

[Fiel 86] N. G. Fielding and J. L. Fielding, *Linking Data*, Sage University Paper Series on Qualitative Research Methods, Vol. 4, Beverly Hills, CA, 1986.

[Flan 54] J. Flanagan, "The Critical Incident Technique," *Psychological Bulletin*, Vol. 51, No. 4, pp. 327-358, July, 1954.

[Frud 87] N. Frude, *A Guide to SPSS/PC+*, Springer-Verlag New York Inc., New York, NY, 1987.

[Guin 87a] R. Guindon and B. Curtis, "Control of Cognitive Processes During Software Design: What Tools Would Support Software Designers?," *MCC Technical Report Number STP-296-87*, Aug. 1987.

[Guin 87b] R. Guindon, B. Curtis, and H. Krasner, "A Model of Cognitive Processes in Software Design: An Analysis of Breakdown in Early Design Activities by Individuals," MCC *Technical Report Number STP-283-87*, Aug. 1987.

[Guin 88] R. Guindon, "A Framework for Building Software Development Environments: System Design as Ill-Structured Problems and as an Opportunistic Process," MCC *Technical Report Number STP-298-88*, Sept. 1988.

[Hori 89] Hewlett-Packard Company, Corporate Training and Development, *The Horizon Project,* October, 1989.

[Hube 89] C. J. Huberty and R. M. Barton, "An Introduction to Discriminant Analysis," *Measurement and Evaluation in Counseling and Development*, Vol. 22, pp. 158-168, October, 1989.

[Isac 88] O. Isachsen and L. Berens, *Working Together - A Personality-Centered Approach to Management,* Neworld Management Press, Coronado, CA, 1988.

[Kaga 85] D. M. Kagan and J. M. Douthat, "Personality and Learning FORTRAN," *International Journal ofMan-Machine Studies*, Vol. 22, pp. 395-402, 1985.

[Klec 80] W. R. Klecka, *Discriminant Analysis,* Sage University Paper series on Quantitative Applications in the Social Sciences, series number 07-019, Beverly Hills and London: Sage Publications, 1980.

[Love 77] L. T. Love, *Relating Individual Differences in Computer Programming Performance to Human Information Processing Abilities,* PhD Thesis, University of Washington, 1977.

[McCl 73] D. C. McClelland, "Testing for Competence Rather Than for 'Intelligence'," *American Psychologist,* Vol. 28, No. 1, pp. 1-14, January, 1973.

[McCr 88] G. McCracken, *The Long Interview,* Sage University Paper Series on Qualitative Research Methods, Vol. 13, Newbury Park, CA, 1988.

[McKe 88] B. McKeown and D. Thomas, *Q Methodology,* Sage University Paper series on Quantitative Analysis in the Social Sciences, series number 07-066, Beverly Hills, 1988.

[Mohe 81] T. Mqher and G. M. Schneider, "Methods for Improving Controlled Experimentation in Software Engineering," *Proceedings of the Fifth International Conference on Software Engineering,* Washington, DC, 1981.

[Mohe 82] T. Moher and G, M. Schneider, "Methodology and Experimental Research in Software Engineering," *International Journal ofMan-Machine Studies,* Vol. 16, No. 1, pp. 65-87, 1982.

[Mora 81] T. P. Moran, "An Applied Psychology of the User," *Computing Surveys,* 13:1, pp. 1-11, 1981.

[Myer 85] I. B. Myers and M. H. McCaulley, *A Guide to the Development and Use of the Myers-Briggs Type Indicator*(R), Consulting Psychologists Press, Palo Alto, CA, 1985.

[Neid 64] C. O. Neidt and R. W. Drebus, "Characteristics Associated with the Creativity of Research Scientists in an Industrial Setting," *Journal of Industrial Psychology*, Vol. 2, No. 4, Dec. 1964.

[Para 90] S. Parasuraman and M. Igbaria, "An examination of gender differences in the determinants of computer anxiety and attitudes toward microcomputers among managers," *International Journal of Man-Machine Studies,* Vol 32., 1990, pp. 327-340.

[Pear 90] A. Pearl, M. Pollack, E. Riskin, B. Thomas, E. Wolf, and A. Wu, "Becoming a Computer Scientist," *Communications of the ACM,* Vol. 33, No. 11, pp. 47-57, November, 1990.

[Shei 81] B. A. Sheil, "The Psychological Study of Programming," *Computing Surveys,* Vol. 13, No. 1, Mar. 1981.

[Shne 76] B. Shneiderman, "Exploratory Experiments in Programmer Behavior," *International Journal of Computer and Information Sciences,* Vol. 5, No. 2, 1976, pp. 123-143.

[Shne 80] B. Shneiderman, *Software Psychology: Human Factors in Computer and Information Systems,* Winthrop Publishers, Inc, Cambridge, MA, 1980.

[Step 53] W. Stephenson, *The Study of Behavior - Q- Technique and Its Methodology,* The University of Chicago Press, Chicago and London, 1953.

[Webe 85] R. P. Weber, *Basic Content Analysis,* Sage University Paper Series on Quantitative Applications in the Social Sciences, Vol. 49, Sage Publications, Newbury Park, CA, 1985.

[Wein 71] Gerald M. Weinberg, *The Psychology of Computer Programming,* Van Nostrand Reinhold Company, New York, 1971.

[Weis 74] L. Weissman, "Psychological Complexity of Computer Programs: An Experimental Methodology," *ACM SIGPLAN Notices,* Vol. 9. No. 3, pp. 25-36, June 1974.

APPENDICES

A Phase 1 Biographical Questionnaire

Interview Date	
Subject ID Number	
Sex	M F
Age	

For each degree that you have completed or begun, please complete the following. Circle the appropriate response or fill in the blank.

Degree	ASSOCIATE BS BA MS MA ENGINEER PhD	OTHER:
Major	CS CE EE MATH PHYSICS	OTHER:
Status	COMPLETED PARTIALLY-COMPLETED	
School		

Degree	ASSOCIATE BS BA MS MA ENGINEER PhD	OTHER:
Major	CS CE EE MATH PHYSICS	OTHER:
Status	COMPLETED PARTIALLY-COMPLETED	
School		

Degree	ASSOCIATE BS BA MS MA ENGINEER PhD	OTHER:
Major	CS CE EE MATH PHYSICS	OTHER:
Status	COMPLETED PARTIALLY-COMPLETED	
School		

Degree	ASSOCIATE BS BA MS MA ENGINEER PhD	OTHER:
Major	CS CE EE MATH PHYSICS	OTHER:
Status	COMPLETED PARTIALLY-COMPLETED	
School		

In the **LAST 2 YEARS, how** much training have you completed relevant to software engineering including any formal study toward a degree? The unit of measure is **'contact hours.'** That is, time spent in a classroom for formal training, and time spent studying for self-study. Hence a semester long college class might be expressed as 3 contact hours a week for 15 weeks for a total of 45 contact hours. Also include corporate training classes, consultant seminars, and self-study.

# CONTACT HOURS	COLLEGE	CORPORATE	SEMINARS	SELF-STUDY	OTHER

TOTAL Training Contact Hours ____________

(Continued on back...)

Complete the following table with the number of years of experience at the Company and elsewhere indicating whether your work was Software Engineering or not. Include any full time employment once beginning your professional career. Hence jobs between degrees count while summer jobs during college don't. The total of all values in the table should sum to your total years of experience.

# YEARS EXPERIENCE	SOFTWARE ENGINEERING	NOT SOFTWARE ENGINEERING
COMPANY		
NOT COMPANY		

TOTAL Years Experience _____

For each programming language used, complete the following:

LANGUAGE	SKILL LEVEL[17]

Use the following table to describe the software engineering methods and tools that you use now or in the past in your job.

METHOD/TOOL[18]	SKILL LEVEL[17]

[17] Self described skill level:
HIGH: expert, lots of experience, well versed
MED: comfortable, some experience, familiar
LOW: novice user, little experience

[18] For example, SA/SD, Yourdon, Jackson, McCabe, Halstead, ..., but be sure to include anything you feel is relevant.

B Phase 1 Standard Ethics Protocol

ETHICS PROTOCOL

(To be read by interviewer before the beginning of the interview. One copy of this form should be left with the respondent, and one copy should be signed by the respondent and kept by the interviewer.)[19]

Hello, my name is Rick Turley. I am a researcher on a project entitled **POSE: The Process of Software Engineering.**

This project is being conducted as a Ph.D. dissertation in the Department of Computer Science at Colorado State University.

I am the principal investigator and may be contacted at (XXX)XXX-XXXX or xxxxxxxxxxxxxxxxxxxx should you have any questions.

Thank you for your willingness to participate in this research project. Your participation is very much appreciated. Just before we start the interview, I would like to reassure you that as a participant in this project you have several very definite rights.

* First, your participation in this interview is entirely voluntary.
* You are free to refuse to answer any question at any time.
* You are free to withdraw from the interview at any time.
* This interview will be kept strictly confidential and will be available only to members of the research team.
* Excerpts of this interview may be made part of the final research report, but under no circumstances will your name or identifying characteristics be included in this report
* As part of this study you will be taking the Myers-Briggs Type Indicator Test. You are free to not answer any of the questions on this test and may decline to take the test at all.

I would be grateful if you would sign this form to show that I have read you its contents.

_____ (signed)

_____ (printed)

_____ (dated)

[19] *adapted from: G.* McCracken, *The Long Interview,* Sage University Paper Series on Qualitative Research Methods, Vol. 13, Newbury Park, CA, p. 69, 1988.

C Phase 1 Critical Incident Interview Outline

The structure and philosophy of the interview is drawn from [Hori 89]. The purpose of the *Critical Behavior Interview* is to find out what people actually do in critical job situations. It is a systematic method for gathering detailed information describing how a job is actually performed and what differentiates exceptional performance from average performance. It identifies the knowledge, skills, abilities, and motivation which contribute to outstanding performance. The technique assumes that past behavior predicts future performance.

In general, the technique focuses on learning about the background, thoughts, feelings, behavior, dialogue, and outcome of a significant example situation. The interview technique proposes key questions to ask:

1. Getting Background
   "Take me back to the beginning of the situation. How did it start?"
   "How did you first come to be involved in that situation?"
2. Getting Thoughts
   "What were you thinking?"
   "What was going through your mind a the tune?"
3. Getting Feelings
   "What were you feeling when that happened to you?"
   "What was your reaction?"
4. Getting Behavior
   - "What did you do?"
   "What was the next thing you did?"
5. Getting Dialogue
   "What did you say to him/her/them?"
   - "Can you tell me what the exact words were?"
6. Getting Outcomes
   - "How did it all turn out?"
   - "What was the final result?"

NOTES

* note team size of described project and program
* note application type
* note industry type
* note user type (end user, internal user, external user, system integrator, ...)
* probe managing or mentoring activities of subject
* to what extent is *domain knowledge* a factor?

Introduction

(5 minutes)

Purpose: To brief the person about the areas to be covered in the interview and to establish a comfortable atmosphere.

1. Establish a comfortable, relaxed tone. Set the stage with small talk.
2. Choose a comfortable seating arrangement.
3. Open the interview by explaining its purpose.

   The history of software engineering is long and varied. Most of the research in this field has focused on software tools and methods. Generally, the industry has looked for a 'silver bullet' in the form of new technology. This has been true despite significant research indicating the significance of the role of the individual in software development. Much of the research into 'software psychology' has emphasized the effect of different tools and techniques on the productivity of individuals. In this research, there have been performance differences of up to 22:1 in completing the task under study. Most research has chosen to view this as 'sample variance' and account for it statistically. I'm attempting to study these differences directly by looking at the individuals doing the job of software engineering.

   You have been selected by your management to participate in a study of software engineering excellence. The purpose of this study is to identify the key skills, behaviors, and knowledge required to be an exceptional software engineer. I will be using the data from this interview and 30-40 more like it to build a profile which will be later validated by surveying a larger population of software engineers. This profile will be used to determine the attributes of exceptional software engineers in order to influence training and development.

   This is a 'research interview' so I need you to be as specific as possible and assume I know nothing about your job. I also may need to interrupt or cut you off at times in order to get the desired level of detail.

4. Briefly outline what ground will be covered in the interview.

   This process will take approximately 2 hours and include a review of the preliminary survey of your background which you have already completed. This will focus on your education and experience in software engineering. This will be followed by an in-depth interview covering some of your particular experiences in software. The interview will follow the critical incidence model used in behavioral interviewing. You have already completed the Myers-Briggs Type Indicator test designed to assess your personality profile.

5. Ask permission to record the interview on tape, and assure the person of the confidentiality of the interview process. Complete the ethics protocol.

   This research is completely voluntary on your part. You may choose to not answer any question and you may terminate the process at any point. I would like to tape record the interview so that it can be transcribed for later analysis. If at any time you feel uncomfortable with the taping, just let me know and I will turn off the recorder. Excerpts of this interview may be part of the final research report, but under no circumstances will your name or identifying characteristics be included in this report. If you are comfortable with continuing at this point, would you please sign the ETHICS PROTOCOL which explains these provisions. THANK YOU!

6. Do you have any questions before we begin?

***** TURN ON TAPE RECORDER *****

Background Information

(10-15 minutes)

Purpose: To get an overview of the person's recent job history and accomplishments.

Mechanism: Complete the Biographical Questionnaire.

Key Situations

(40-120 minutes)

Purpose: To find out exactly what the person did to achieve specific accomplishments and deal with difficulties or frustrations.

Situation #1: High Point

1. Establish a transition from the background information to this next part of the interview.
2. Ask the person fo the first key situation; a *high point.*
   - "Tell me about a time in the last year or two that was a real *high point* for you, a time when you were able to accomplish something you felt really good about, or a time when you felt especially pleased with something you did. First, give me a brief overview of the situation so I know what you're going to be telling me about, and then we'll go back for the details."
3. Let the person think about a situation and begin to tell you about it.
4. When the person has focused on a situation of the appropriate scope indicate that this is what you are looking for and probe for details.
5. When the person has related all the details ask for another replay with emphasis on *what the person did, what they said, what they thought about it,* and *what the outcome was.*
6. When sufficient detail is received, close the discussion on this topic.

Situation #2: High Point

Repeat as above to obtain a second situation description.

Situation **#3: Low Point**

Repeat as above to obtain a low point situation description.

In asking about *low point* situations, it is important not to ask simply for a situation where the person felt frustrated because the person may recall a situation where something happened to him but in which he had little if any direct involvement. Instead, you want a situation where the person felt frustrated as a *consequence* of something he attempted or had an active role in trying to bring about.

Additional Situations

Repeat as above allowing the person to define the situation domain - the job area or experience topic that he/she wants to talk about.

Final Situation

Repeat for a final high point situation to leave the interview on a positive note.

Closing

(10-20 minutes)

Purpose: To get examples of how the person demonstrated qualities that she considers to be strengths and to allow time for her questions.

1. Allow the person to summarize his/her experience by reflecting on some of her strengths.
2. Ask the person to give a list of capabilities, characteristics, and areas of expertise for their job.
3. Ask the person about training or other development opportunities that the person believes were especially beneficial to him during his career. .
4. Close the interview by thanking the subject for time spent and offering to answer any questions.

   Note what phrases the subject considers to be the earmarks of an exceptional programmer.

D Derived Competencies from Transcript Analysis

TEAM ORIENTED		
	# Subjects	# Incidents
Exceptional	9	14
Non-Exceptional	5	10
Fisher's Exact Test (Two Tail)	0.1409	

Definition: Values synergy of group efforts and invests the effort required to create group solutions even at the expense of individual results.

Associated Behaviors:

* Balances the strengths and weaknesses of other team members.
* Recognizes synergy of group efforts and invests personal time and energy to leverage it.
* Promotes constant communication among team members using such techniques as brainstorming sessions, travel, phone calls, e-mail, or just sitting close together.
* Shows concern for the feelings of others. Treats others with respect. Attempts to create *buy-in* for decisions.
* Guides or influences others.
* Promotes the development of shared values on the team.
* Exhibits commitment to the whole project, not just the assigned part.

Illustrative Examples:

The knowledge needed to be there on both ends to pull it all together, but also the synergy, just the two people working together and kicking ideas around and trying things, and when we finally figured out the missing ingredient that made it all come together. ... Fundamentally we needed the knowledge of both of us to pull it off, although there was some synergy just from working together.

It was important to have two of us in that we tended to pick up different things and bounce back and forth the ideas.

We have really good team dynamics. We tend to have lots of interaction; we tend to, uh, to share ideas very fluently, and we tend to think of ideas in the process of conversations with one another.

We spent a lot of time trying to communicate, that was the key thing, I think.

We'd have a problem, we'd sit down and look through the code together or try to figure out what's going on, do SA's, or SD's, you know, structured analysis on the board, white boards, next to each other, and, uh, try to figure out what's the best way to work on something. We'd try to partition the program so that, uh, we had a very defined interface between us.

My first goal was to establish myself, get their respect. ... The first thing was to encourage other people and make sure that you mean no harm to them.

Trying to let everyone get their say in. ... Build some kind of consensus. ... Encourage people. ... Trying to get them to open up and say what they're thinking. ... Letting people be the way they want to be.

Some *NOT* Examples:

I was doing it myself, which tends to make things go quicker when you work with someone. ... I think one of the best things is that I get left alone a lot. ... I tend to hide. ... Everything I've done was by myself, basically.

SEEKS HELP FROM OTHERS		
	# Subjects	# Incidents
Exceptional	4	6
Non-Exceptional	7	12
Fisher's Exact Test (Two Tail)	0.3698	

Definition: Proactively seeks the assistance of others in learning, researching, designing, understanding, debugging, or checking results.

Associated Behaviors:

* Asks previous implementers to explain their designs.
* Asks other engineers to critique or evaluate a design.
* Surveys others to create a list of alternatives.
* Contacts others to help solve problems.

Illustrative Examples:

I had to go to other people for information on things that I didn't know. ... There is an engineer who worked on the original project porting it, that I was able to go ask questions of when I came up with a list and couldn't figure it out instead of banging my head. I have a tendency to really try and figure it out myself for too long before I go and ask people for help. I waste time that way. ... I went and asked one of the designers of the input who did the original port ... also the other designer. ... I was calling people all over the place, in the Kernel lab, strings and notes. And there is no clean way to do it, I determined. ... I ended up calling someone in the Kernel lab who had some ideas and said here's a way to do it. ... I rely pretty heavily on some other people in the lab who I would consider to be hackers to go in and play with this and find out what all the problems are and come up with solutions. ... I'm not afraid of trying to call anyone in HP. ... I tried to give it to people that I consider to be UNIX gurus to go and test out with it, because I knew I didn't have the large knowledge base that some other people did. ... I talked with product marketing and product support to find out what would they, what would people expect. And I also, I called people in X. ... Talking with marketing and finding out what do you think the customers are going to expect.

HELPS OTHERS		
	# Subjects	# Incidents
Exceptional	2	7
Non-Exceptional	0	0
Fisher's Exact Test (Two Tail)	0.4737	

Definition: Spends a significant amount of time assisting others in the completion of their tasks or influencing broad organizational direction.

Associated Behaviors:

* Acts as a lab-wide consultant for process or product issues.
* Reviews, directs, or influences the work of other engineers.
* Takes over tasks from other engineers in an effort to complete the project.
* Teaches engineering skills.

Illustrative Examples:

[I] tended to be likely to get done early and either pick up other portions of the code, go help other people on the project, or occasionally get drawn into outside things such as management consulting or other investigations. ... I still get asked to go out and go for a walk and let's talk about something that's bothering me, and consulted in terms of code reviewing, sometimes people will come over and say I wrote an algorithm that does this, and could you glance at it and see if you find any bugs, so consulting in terms of both coding and people issues.

USE OF PROTOTYPES		
	# Subjects	# Incidents
Exceptional	10	10
Non-Exceptional	4	4
Fisher's Exact Test (Two Tail)	0.0108	

Definition: Uses a prototyping method to assess key system parameters before designing the final product. Avoids using prototype as final implementation.

Associated Behaviors:

* Uses prototypes as a mechanism for incremental development of a product.
* Attempts to not allow prototype to become the final product.
* Uses prototype to assess critical areas like performance and time estimates.
* Prototypes in parallel with the detailed design phase.

Illustrative Examples:

Subject created a prototype of a vertical slice partially implementing the key areas in order to assess the likely system performance and estimate effort.

[The] strategy was to try and prove feasibility and investigate schedule by doing a portion of the work as a prototype.

It was simply for performance measurements and to see what it would take to start and do some of this from scratch. ... I felt very strongly that what I had done was not meant to be evolved into the real product. ... they took the prototype that I had worked on and just basically turned it into the real thing, and that piece of code, that one segment is the biggest nightmare of this whole project and one of the reasons that we kept having major schedule problems on it.

The purpose of the prototype was to communicate it to management and to other people.

I explicitly did it in LISP so that I could throw it away and start over.

By doing a little bit of prototyping, it really drove some decisions there.

I guess the idea was to prototype at least a few of them to get a little better perspective on performance.

WRITES/AUTOMATES TESTS WITH CODE		
	# Subjects	# Incidents
Exceptional	9	9
Non-Exceptional	4	4
Fisher's Exact Test (Two Tail)	0.0573	

Definition: Applies incremental testing techniques during code development such that given module achieves a high degree of reliability by the time it's completed.

Associated Behaviors:

* Creates tests in parallel with the creation of code.
* Automates tests so they can be run frequently throughput design.
* Tests frequently during development to ensure reliability at the module level.

Illustrative Examples:

I'm one that will code a piece and then will probably take the time to write some kind of shell around to test.

The test phase on our project was actually, I think, overlapped immediately from the very [first] algorithm I developed. So let's say my algorithm, let's say there's 30 modules in my algorithm, I would have mechanisms for testing every piece in between there or try to. Uh, one of the things we did, and these were gradually put in, but we collected over a thousand different pages, some of them from customers and things like that, we put 'em all into optical media and then we wrote automatic, uh, test scripts that would go and generate the intermediate data for each one of those things if you wanted to, so I could go on a thousand different pages and look how well this first algorithm worked, or at any place in the middle I could print out the results.

I tended to do a lot of testing during the implementation phase, rather than waiting for QA phase.

We already had a set up about 500 automated tests in our test suite, and we were adding to that as we went along, so as a new area would be developed we'd try and write a new test for it that could be automated.

I'd try to test each object by itself as I implemented it. ... I didn't keep any fancy test scaffolds for each one individually. It's just that I'd try to run a few tests of the object by itself where it was feasible to verify or to ease the integration process later on.

Most of the code that I write I usually develop a test run, a regression test for each module, so I can actually do a module, run a little harness for it, and then do actual testing on it, and that way I can see some result.

I think the thing that we did the best ... is we thoroughly tested our modules, our portion of the code. ... We wrote shells that would just test our portions of the code and ran it, ran millions of images through it by the time we were done, just repeating the same set over and over and over and could work out defects.

I'll code it, and then compile it and try it, you know rather than taking a big project and coding the whole thing and then see if it all works, and beginning to work like that, I take the chunks, and each chunk as I build it, I try to wring it out at that time, so that as I add things to it, and begin to put the big project together or program, whatever it is, I try to have each of the individual pieces tested pretty well already when I put it together.

I wrote sets of tests that I would run every now and then on components whose values I knew, ... whenever I made major changes, or just every time I got real antsy about it, you know, every so often I would go in and I would run this test set that covered a lot of different components.

KNOWLEDGE		
	# Subjects	# Incidents
Exceptional	7	7
Non-Exceptional	6	6
Fisher's Exact Test (Two Tail)	1.0000	

Definition: At the time of assignment, possesses the unique skills or knowledge required to accomplish the task at hand.

Associated Behaviors:

* Possesses the necessary domain knowledge and skills required for the job.
* Previous to assignment, and on own initiative, becomes an expert in a given area.

Illustrative Examples:

I tended to understand the objectness of object-oriented design to a little more extent. ... I understood the theory.

The success thing for me was having the right mixture of hardware and software experience so I could really understand what this hardware was doing. ... I had had a lot of experience in assembly before and knew a lot of tricks.

I mean you had to learn to understand ECGs. The medical content was much more challenging and interesting than the computer science part of it.

I knew more about the host systems than they ever did, because of my background on other stuff that I had done.

One of my former assignments was, I wrote the majority of that diagnostics program in the past, the tests themselves. ... I was very familiar with this program and how it ran.

OBTAINS THE NECESSARY TRAINING		
	# Subjects	# Incidents
Exceptional	6	7
Non-Exceptional	6	8
Fisher's Exact Test (Two Tail)	1.0000	

Definition: Actively seeks the necessary training required to complete the assigned task.

Associated Behaviors:

* Seeks out documentation required to understand current assignment.
* Takes classes which will directly help in the completion of the current assignment.
* Keeps current by reading trade or technical journals.
* Improves skills and awareness by attending conferences.

Illustrative Examples:

I'm one that learns through that kind of actually textbook approach in some sense. So just I read through a lot of the documentation.

Took an object-oriented programming class.
Hired a consultant to teach a course in structured analysis and design.
Brought in a testing consultant toward the end of the project.
Kept up with trade publications, took courses from universities.

I sat down and read the manuals for a week ... and then looking over some old code.

I spent a few weeks just sort of going over manuals and going over documentation I was able to get on previous generations of the compiler to see what some of the issues really were.

I did a lot of reading of all the documentation that had been generated to date during the original investigation.

I've had to gain more depth in the operating system structure, in constructs and facilities that are available of windows, presentation manager, Macintosh, and even a little bit on the UNIX machine with X and Motif.

I'm a firm believer that if you have attitude you can force the skills because I certainly didn't have the skills on Objective C and I picked it up pretty quick.

I gathered the IRS's and the schematics for the boards that were interesting, the ones I would be calibrating and making measurements on. And I started getting familiar with the hardware. I had a general idea of what they could do from the system ERS, and then I started looking at the individual boards and the different groups of components, the different functional areas on those boards, just to find out specifically what they could do.

LEVERAGES/REUSES CODE		
	# Subjects	# Incidents
Exceptional	5	6
Non-Exceptional	5	6
Fisher's Exact Test (Two Tail)	1.0000	

Definition: Proactively attempts to leverage other engineer's efforts by using their code or designs and attempts to leverage own effort by making their code reusable.

Associated Behaviors:

* Looks for code or code fragments which can be reused.
* Designs and codes so that effort can be reused.

Illustrative Examples:

"No way are we going to start from scratch on this. Let's go with the, uh, let's look into this MAC APP."

The other, I think, big boon of this project was it was true reuse, true reuse across platforms.

I'm up for stealing anything I can, so it's the throw out the 'not invented here' attitude. ... There are other places where I can make contributions rather than rewriting someone else's stuff.

I then went and looked and talked to a friend in California who was in technical support of the 1000 computers and just got from their goodies tape some utilities they had written to do — not identical but similar sorts of things.

I'm a big fan on stealing things that are appropriate.

One of the biggest things that I did, and this was my own decision, was I said that I'm not going to make a separate driver for the microchannel; I'm going to make a driver that works on both the microchannel and the AT backplane, and it'll be self-configurable so that we can have the same disks.

What I did was I found someone in the building was working on user interface, and I took his user interface; it was just really easy to take the interface, the interface being a bunch of procedures that would get certain things up on screen, and then I took the, someone else was working on the scanner library, stuff that would make the scanner do different things, and I just basically put them together.

We needed a way to tie in the standard in and standard out types to shell processes, basically we just stole that out of one of the manuals.

USES STRUCTURED TECHNIQUES FOR COMMUNICATION		
	# Subjects	# Incidents
Exceptional	6	7
Non-Exceptional	3	3
Fisher's Exact Test (Two Tail)	0.3698	

Definition: Takes advantage of the tools and techniques of structured design in order to understand and communicate designs, but does not follow the complete formalism of the approach.

Associated Behaviors:

* Uses structured techniques (such as Structured Analysis and Design, Hierarchy Charts, ...) as a means of joint development and communication.
* Uses structured techniques as a mechanism for passing off a design or part of a design to another engineer for implementation.
* Views structured techniques as just another tool which can be applied to certain problems, rather than as a panacea.

Illustrative Examples:

They call me the 'bubble lady.'

We started using these bubbles to just explain the different, the main, the two main screens that would show up and then you could use the SA approach in that you can bubble down then, you know, and expand it down, and we did some of that in aspect with the user interface and the flow through it.

Structured design wasn't the panacea, the cure all things, partly due to inexperience, partly due to lack of tools to help enforce some of the things that we wanted to do. ... It got results quicker and the results were what we wanted for the product.

Just doing that top context diagram to me is a real important thing, because then you can really start talking about it and wrapping your mind into it pretty good. ... It was a good communication tool.

We did do hierarchy charts and flow diagrams. We did not maintain them through the life of the program; they were basically a jump start tool. We did review those with the rest of the team.

I took some of the loosest concepts of structured analysis and just drew diagrams on the white board and asked them if they were right. ... I felt like I could be confident that all the boundary conditions were easily understandable and were confined and simplified.

I'm sure that I kind of remember going back and pulling out the diagrams we'd drawn, and that provided our means of communication again for us to discuss and understand each other meant.

METHODICAL PROBLEM SOLVING		
	# Subjects	# Incidents
Exceptional	6	6
Non-Exceptional	3	4
Fisher's Exact Test (Two Tail)	0.3698	

Definition: Uses methodical approach in understanding and solving problems.

Associated Behaviors:

* Builds mental or physical system models to enhance understanding and visualization.
* Designs well controlled experiments to efficiently resolve problems.
* Invests in the development of test tools to solve problems.

Illustrative Examples:

We drew several partial successes, each of which gave us enough confidence that we were on the right track, but each of which introduced additional problems that needed to be solved.

I figured if I understood the system — the problem would be apparent.

It helped that we kind of worked from the ground up. We were not taking large things and trying to break them apart; we were building them. ... So we'd build up the biggest things we could, and then we'd look at special cases.

Some *NOT* Examples:

That was a lot of trial and error. I would come up with what I thought would work and then I had a lot of write statements, you know, print statements in my code so that when I press a character I would see what would come out on the other end and whether RMB was going to get screwed up or not.

USE OF NEW METHODS OR TOOLS		
	# Subjects	# Incidents
Exceptional	4	4
Non-Exceptional	1	1
Fisher's **Exact** Test (Two **Tail**)	0.3034	

Definition: Seeks to improve performance or results through the use of new tools or methods.

Associated Behaviors:

* Recognizes value in new tools or techniques.
* Proactively seeks out new tools or methods to solve problems.
* Uses work assignment as a way to learn new tools or methods.

Illustrative Examples:

I got to update it in terms of the kinds of implementation approaches, the technology that it was built on, went to full object oriented implementation instead of the very ad hoc Pascal version of it before. ...I've been doing object oriented programming since before it was a popular thing. I've probably been doing it for seven or eight years in various forms, and to me it clearly has so many benefits and so many different ways, especially in terms of the things I did on this project. Because you start out modeling the problem domain instead of taking the traditional structured analysis design approach, and that has had tremendous benefits in terms of the maintainability and modifiability of this compiler.

SCHEDULES AND ESTIMATES WELL		
	# Subjects	# Incidents
Exceptional	4	4
Non-Exceptional	0	0
Fisher's Exact Test (Two Tail)	0.6285	

Definition: Shows a strong concern for schedules and estimates schedules well.

Associated Behaviors:

* Maintains personal "rules of thumb" for schedule estimation.
* Refines schedule estimates based on measured progress.
* Meets schedules.
* Schedules via task breakdown and successive refinement.

Illustrative Examples:

We de-rated it by about 50% or something, to take into account learning code and some of the other engineers that we knew we were going to get. ... We were about 50% short. ... We used the information gained in actually doing some porting and turning on to estimate more closely what we thought it would take, and we refined these estimates constantly as we went through the project.

The overall project had a schedule problem, personally I didn't. Each one of my milestones I was meeting, I met my 50% schedules on all of the portions that I was working on and throughout the project I wound up taking on more of other people's. ... I know how to set schedules and I know how to get my work done. ... Experience, you know, I've been working here for 12 and a half years and I used to mess up a lot, and now I'm getting to where I don't mess up.

USES CODE READING		
	# Subjects	# Incidents
Exceptional	3	3
Non-Exceptional	1	1
Fisher's Exact Test (Two Tail)	0.5820	

Definition: Uses code reading and other group development techniques to ensure final code quality.

Associated Behaviors:

* Asks for others to code read their work.
* Participates in the code reading of other engineer's work.

Illustrative Examples:

I ended up reading all of my code twice and I helped read two other people's code in non-author code reading, and that was very beneficial.

When I finished that, I had what I thought was a reasonable amount of code, I went to this other person that I work with, and I said, would you code read this, and rather than sitting down in a formal code review, which would have been a waste of time for both of us, he just took it home one night [and] read it.

DESIGN STYLE		
	# Subjects	# Incidents
Exceptional	7	9
Non-Exceptional	9	10
Fisher's Exact Test (Two Tail)	0.5820	

Definition: Uses decomposition design techniques relying on visual representation of designs. Creates structured designs, generally without using formal techniques.

Associated Behaviors:

* Keeps design specification in sync with the actual implementation.
* Enumerates and prioritizes (design alternatives, defects, ...)
* Follows a *top-down* design method using decomposition to successively refine the design.
* Recognizes value of up-front design in leading to a successful implementation.
* Takes modular approach in order to reuse as much of the design as possible.
* Uses pictures to communicate and understand designs.

Illustrative Examples:

One of the few times I've seen a design kept up to spec with the actual implementation, and I think we did a fairly good job of that.

We just took a real modular approach. Instead of designing a dual-channel algorithm one more time, we're going to take two existing copies of the single-channel algorithm and feed their information to a resolver, which will combine the information.

There was a long period of drawing up lists of a half dozen or so candidates for things that would be profitable to go investigate and looking at risk benefit sorts of things on each of them and choosing which one had the highest expectation of a payoff, and which one was worth going and doing.

Just try to think of the smallest natural procedures and functions that makes sense. There's no formal structured design or anything like that. ... Particularly small ones where the programs ... usually I feel quite free to go ahead and use global variables ... as long as I document thoroughly.

We didn't spend lots and lots of hours doing structured design or structured analysis or DFDs or anything like that. At the same time, those techniques were involved in, at least in my opinion, in any intelligent assessment of software task.

I kind of had this idea of breaking the project up in chunks, thinking of the different things that needed to be done, you know, breaking it off in tasks, and taking one task at a time. So I kind of went to the heart of what would make it work or not, you know, getting the interrupts in properly, counting them doing all the interrupt routines to make sure I didn't lose pulses, you know, not get too many, and that kind of thing, so I started there. ... I really like to work on small tasks, if you will, and iterate on it. I'll code it, and then compile it and try it, you know rather than taking a big project and coding the whole thing and then see if it all works, and beginning to work like that, I take the chunks, and each chunk as I build it, I try to wring it out at that time, so that as I add things to it, and begin to put the big project together or program, whatever it is, I try to have each of the individual pieces tested pretty well already when I put it together.

I think in pictures, so I typically, on any problem, I sit down and I start drawing.

I took each card, broke it down to its functionality, functional blocks, and then looked at each of those and described then that basically would turn into an Auto Cal set, and then I would describe what I envisioned doing for that functional block, and then I'd go talk to the hardware guys about does this make sense, is this how you think it works.

FOCUS ON USER OR CUSTOMER NEEDS		
	# Subjects	# Incidents
Exceptional	6	6
Non-Exceptional	5	6
Fisher's Exact Test (Two Tail)	1.0000	

Definition: Considers customer or user input and feedback to be an essential ingredient in the design of products.

Associated Behaviors:

* Proactively attempts to obtain customer and user input and feedback for products.
* Measures project success in terms of customer satisfaction.

Illustrative Examples:

It was well received by the customers. Because that point of view carried up into doing a lot of things that were very beneficial to people. There were some capabilities that it had in that system that haven't been matched in our next two generations of systems that we've done. ... Taking these fuzzy requirements that we were formulating and coming up with a solution for our customers was a real fun part of that project, because there was no precedent.

As we went through the process of interviewing these people, I learned a lot about Macintosh developers. I learned a lot about how they thought, what their issues were, how they got to where they were, what the kind of things were that they valued, and how they went about creating product.

And then when I got everything exactly right — just for the heck of it, I brought in a couple of people and say, you know, in a user testing type of situation, uh, different types of people, I brought in about three, I think, and say, "Here, do it." And I wanted to sit back and watch them just breeze through it, and of course, they didn't; you know, they got caught in different places.

RESPONSE TO SCHEDULE PRESSURE		
	# Subjects	# Incidents
Exceptional	4	4
Non-Exceptional	5	5
Fisher's **Exact** Test (**Two** Tail)	1.0000	

Definition: In response to schedule pressure, sacrifices important parts of the design process.

Associated Behaviors:

In response to schedule pressure:

* provides incomplete documentation.
* does not adequately inspect or test product.
* does not prototype or adequately design risky parts of product.
* hands off parts of design or test to others.

Illustrative Examples:

We were so pushed all the time that we never really had a lot of time to do some of the things that we wanted to do like walkthroughs and things like that.

*We had a schedule pressure and we never **did,** even **thought** the schedule kept slipping **out,** we never did have a block of time in which I could go in and do it right.*

*In hindsight, the overall thing slipped **out,** and there was enough time to have done that part right, and I wish I had pushed harder for it.*

Subject's comments about continually being asked to pull up a schedule: *No matter how good your intentions are you **can't** resist the pressure to come up with the right answer to that.*

We didn't have all the automated tests done that we had really committed to.

*I think it's one of these things that we didn't **spend** the time up front, and that was because of schedule or perceived pressure of schedule that we did **not** feel we had the time to prototype and play with some of these things, and we thought well, we thought it would be **rule-based,** but we had not clue, I had no clue, how complex the whole thing would turn out to be. And that's just naive expectations, I guess.*

Didn't have a lot of design documentation because I was under a real gun at the time.

EMPHASIZES ELEGANT AND SIMPLE SOLUTIONS		
	# Subjects	# Incidents
Exceptional	5	5
Non-Exceptional	3	3
Fisher's Exact Test (Two Tail)	0.6499	

Definition: Creates solutions which are elegant and simple and allow for easy extension to future needs.

Associated Behaviors:

* Values simple solutions.
* Designs general solutions which will be easily extendible, even if it's not currently needed.
* Applies structure to ill defined problems and problem domains.

Illustrative Examples:

I guess that's why I thought it was kind of neat because the elegance in it is that it was so simple to get it down on a piece of paper

I built the ASSEMBLER to have a concept of modules and exporting names and importing names to try and do encapsulation of ideas, so I was trying to build kind of a structured ASSEMBLER.

I couldn't wait to get back to the hierarchy charts and clean them up. I just want everything to be just nice and neat and real easy to understand and real easy to communicate and really fits into the realities of the hardware also.

This allowed us to do more than what we needed to do. It was more flexible than what we really needed, which I liked, because that means in the future, not knowing what's coming up next, this is a lot more flexible, I'm not constrained, I'm not up against a hard limit already, I have a lot more room, so there's a lot of flexibility in this of stuff we probably don't need to use, but it fits all of our cases.

PRIDE IN QUALITY AND PRODUCTIVITY		
	# Subjects	# incidents
Exceptional	6	7
Non-Exceptional	6	7
Fisher's Exact Test (Two Tail)	1.0000	

Definition: Takes pride in producing defect free products on schedule in minimum time.

Associated Behaviors:

* Takes pride in meeting or beating schedules.
* Takes pride in achieving low defect counts.
* Takes pride in achieving high productivity and accomplishing significant amounts of work over short periods of time.

Illustrative Examples:

I felt pretty good because as a result of using that I was able to increase I think the quality of my code a lot, I ended up with one fourth the bug rate of the next closest person, so I felt pretty good about that.

That code has been very, very good. I don't think we found any defects in it, or maybe one. It's probably, in terms of defects, the best code I've ever written.

Typically this compiler that I'm working on, it would have taken anywhere from three to five people for two years to make. And I did the bulk of it in a year by myself.

The mind set that most people had was that you developed this code and then bum it in ROM and you don't ever get to change it. Once it's in ROM, you don't get to go out and replace all the ROMs in people's instruments, so it had to be right. The legacy of that was still carrying over to us when we did this. We would never have admitted to ourselves that it was okay to have defects in the software.

I think probably the thing that I look back on about that is there's been very few defects against this part of the code. Not to brag or anything like that but ...

[We] sold a boatload of them and just didn't have significant defect issues.

We beat it to death and we've gotten very few complaints from the field, very few problems have come up, and so the now the product's been stable for a couple years.

PROACTIVE/INITIATOR/DRIVER		
	# Subjects	# Incidents
Exceptional	6	11
Non-Exceptional	5	9
Fisher's Exact Test (Two Tail)	1.0000	

Definition: Takes the initiative to identify ways of completing important tasks. Influences others to consider alternative approaches.

Associated Behaviors:

* Proactively completes projects or tasks that they consider important.
* Influences others in design, organizational structure, ...
* Identifies ways to surmount barriers and remove obstacles.

Illustrative Examples:

I was not satisfied with it because I kept making these various proposals, not do it, go do a driver, you know, both of those I thought would have been a better alternative than doing it partially (?) and when it looked like I was going to lose that case, then I went off and did something anyway to make it better, and that is that on my own, I didn't even talk to management about this, I just went and extended it to support the MUX card.

I tend to be proactive in solving these, I would go over to someone's desk, or say let's go take a walk, or sometimes I would go to management and say, let's go take a walk.

We were the two that had the most free time and so we would either individually or jointly go in and pick bugs up out of there and track them down.

Subject describes a significant, successful project which was started by two engineers who had a vision of what should be done and just did it - even when they should have been doing a different job.

I would publish memos, and I published a decision since the steering committee did come to a decision, I published the decision, and everyone went, "No, no, no, you can't publish a decision; who are you to do this?"

Some of the equipment I couldn't get. I went up to Fort Collins; I went up to their lab, and I brought my tests and spent an evening up there and tried stuff. ...I can go to Fort Collins, you know, to test that.

I decided ... we better do this. ... I went back and said we don't have time in the schedule to do it, but I'll just make time. I did it because I felt it was important.

PROACTIVE WITH MANAGEMENT		
	# Subjects	# Incidents
Exceptional	5	9
Non-Exceptional	5	7
Fisher's Exact Test (Two Tail)	1.0000	

Definition: Proactively attempts to influence project direction by influencing management.

Associated Behaviors:

* Discusses issues concerning other engineers with manager.
* Attempts to set project direction and make project decisions by influencing manager.
* Make specific resource or assignment recommendations to management.
* Promotes product ideas through demos or selling of ideas.

Illustrative Examples:

Eventually [my managers] came up with a project strategy that looked very similar to the one that I had worked out with him prior to that, so I was really proud to be able to somehow, even at the very lower level, prod people to come up with a product strategy, and I did that only because I had to know how to specify my product, and I wasn't going to just do it the way someone had attempted, had told me to.

The thing that was actually one of the neater parts of it is selling the ideas. ... It did require doing a data sheet, getting all the project manager in the section turned on about it so that when the section manager kind of finds out about it, you know it sort of got the green light to go ahead. ... The power of the demo can never be underestimated. ... We already had something running once we got the section manager into it. Of course, they just start doing back flips when they see that. So it's pretty important around here to have a demo of some sort to really get things going.

DRIVEN BY DESIRE TO CONTRIBUTE		
	# Subjects	# Incidents
Exceptional	5	6
Non-Exceptional	3	3
Fisher's Exact Test (Two Tail)	0.6499	

Definition: Values the sense of accomplishment which comes from making a direct contribution.

Associated Behaviors:

* Seeks assignments where they can contribute.
* Feels rewarded by the chance to contribute.

Illustrative Examples:

I felt like I was really making a contribution. ... The part that was really satisfying about that job is we could make impact on a large number of projects, and we could make impact on sort of the culture of HP.

I just felt really good because I had a major part to play in that thing, both in the hardware and software.

I was able to make a contribution in this area, at least by putting something in that was not even in the original plan.

I was able to introduce significant new technologies in the lab, and I really feel good about the results that we came out of it with.

I find it much more satisfying if I feel like I'm accomplishing something occasionally, you know, rather than doing a lot of work that it doesn't seem like you're getting much done.

At the time it seemed like a contribution.

SENSE OF FUN		
	# Subjects	# Incidents
Exceptional	5	9
Non-Exceptional	2	2
Fisher's Exact Test (Two Tail)	0.3498	

Definition: Enjoys the challenge of the assignment and the sense of accomplishment from completing it. Just plain has fun at work.

Associated Behaviors:

* Looks forward to going to work.
* Derives a sense of accomplishment from work.
* Enjoys the challenge of a tough assignment.
* Driven by the reward of doing something new and different.

Illustrative Examples:

*The fun was the challenge of testing something that nobody had any idea of how to **test**. ... In this case, nobody had ever done this before. Nobody knew how to do **it**. ... It's fun to **just** trying to think of ways to try to trick the software.*

It was for me personally a tremendously fruitful kind of a project, and it was very, very cool to do the first thing that the company had ever done on this specific platform.

*It was **a** fun one, it's rare **that** you get to do something maybe in the course of a month like that from start to finish and write some new code in and deal with the whole algorithm and go through a complete, mini design cycle.*

SENSE OF MISSION		
	# Subjects	# Incidents
Exceptional	4	4
Non-Exceptional	2	2
Fisher's Exact Test (Two Tail)	0.6285	

Definition: Driven by a sense of mission and clearly articulated goals to achieve a specific result.

Associated Behaviors:

* Creates and articulates clear and specific goal statements.
* Drives project effort to achieve specific goals.

Illustrative Examples:

We were very focused on the problem we were trying to solve rather than the process of how we were trying to solve it. ... We knew that we had to develop that test system. We became, let's say so focused on solving this problem — and the team was very small, so the communications and things weren't a problem. ... I think a lot of it is just that focus on the goal of what we were doing rather than becoming enamored of the processes. ... We had a vision in the sense that we believed that this would turn into something big.

We really knew what we were doing, ... we knew when we were going to be done, so that kept us focused on what we had to do.

We weren't creating a single product, we created product strategy. We understood very clearly at that point where we were going for at least five years. We understood what the generations needed to be.

LACK OF EGO		
	# Subjects	# Incidents
Exceptional	3	4
Non-Exceptional	1	1
Fisher's Exact Test (Two Tail)	0.5820	

Definition: Stresses the solution over the source of the solution. Doesn't care where a good idea comes from and doesn't feel the need to promote their own ideas.

Associated Behaviors:

* Allows others to re-write an idea they've created.
* Allows use of a *discovery process* for others to come to see the value of their ideas.
* Discusses ideas, not positions.
* Focuses on the end result, regardless of who creates the solution.

Illustrative Examples:

So both being skeptical and very argumentative, we had some fairly heated discussions about how we thought it ought to be, but it was very little ego involved in it. I mean it was trying to get the algorithm right, and we really didn't care whose idea it was that worked, as long as we got the right structure.

The ability to put aside your own ego and say, "You know, we are working for the same thing, and you and I need each other; you need me as much as I need you and try and remember that, and I'm not here to hurt you; I'm here to help you, and you're here to help me."

STRENGTH OF CONVICTIONS		
	# Subjects	# Incidents
Exceptional	2	2
Non-Exceptional	1	1
Fisher's Exact Test (Two Tail)	1.0000	

Definition: Exhibits and articulates strong beliefs and convictions. Acts in accordance with these beliefs, even when it is counter to specific management direction.

Associated Behaviors:

* Argues forcefully for specific point of view.
* Risks performance ranking in an effort to secure the best solution.
* Acts in accordance with beliefs rather than solely based on assignment.

Illustrative Examples:

As a matter of principle, subject objected to handing off a prototype since it was not originally intended to be product quality. At first I objected, and second I objected, and third I objected, and finally fourth, I was being called uncooperative.

There was even consideration of dropping the functionality of that out, but I felt pretty strongly about it, so that's why I actually, that's another reason why I ended up tackling it, because I thought it was really important that we did that part of it.

E Self-Described Competencies

Competency	Definition	Examples	#XP	#NXP	Fisher's Exact Test (Two Tail)
Perseverance	Extra work. Keeping at a problem until it is solved. Stubbornness. Keeps plugging away at stuff. Discipline. Thoroughness. Compulsiveness. Willingness to work hard. Followthrough. Dedication. Perfectionism. Keeping things moving. Commitment. Motivation / Self-Motivation. Staying motivated in spite of tedium.	"I think it's more a matter of persistence than any particular brilliance." "I guess it required somebody who was going to test thoroughly enough and be disciplined enough to write code that was high on reliability." "I'm a perfectionist and I like to see things done cleanly and correctly, none of this spaghetti code or any of that stuff." "I am very disciplined." "I will keep up my hierarchy charts." "Just some tenacity, just sticking with it even though it was apparently not getting anywhere, although realistically, in retrospect, when you don't think you're getting anywhere, you typically are because you're at least narrowing down possibilities, you're eliminating things, even though you haven't found anything yet, you're at least eliminating things and getting the problem smaller." "I just pursued it until it converged." Driving to do tedious work quickly. Finding the work challenging. "Testing is tedious, but if you're disciplined and have a procedure you get through it. It took 60 hours per week."	6	7	1.0000
Knowledge	Having a background in software. Ability to write code in a particular language. Knowledge and background for particular job. Technical expertise.	Assembly language and speed-up technique background. Skills at programming the PC. Knowing the existing compiler. Knowing about object-oriented approaches. Knew about drivers. A base knowledge of HP-UX and RMB. Knowing X-Windows. Having the knowledge of the different relays. Domain knowledge, the file system, etc. A EE-like background knowing how bits and bytes really work. "I think I was the best person suited for this mainly because I knew the most about the overall operation of the program to begin with, the diagnostics program, as far as how it ran and all the intricacies of it, because I had done most of the work on it earlier. ... I didn't have to learn something new to begin to know what to do to implement it. ... I already had the knowledge of how it worked, so that was a real plus." "My job was a job that a true software person wouldn't have done, couldn't have don, because it was so tied in with the hardware."	6	6	1.0000

Teamwork	Working with someone else that had different strengths and being able to feed off each other's. Ability to work with people. Relationship with other people. Team worked well. Ability to work with all of these people, especially through all the problems. Ability to put aside your own ego. Keeping people happy and motivated in a really tense situation. Being sensitive to users and service people. Working with people - personal interaction. Joint ownership. Being able to deal with diversely motivated organizations, real time. Understanding of the global issues of rolling a product and how people and players fit together. Paying attention to more than just the technical aspects of a project.	Recognizing that in some cases the strengths of the other members of your team are going to be stronger than your own. Recognizing that it's best to not try to get your idea adopted for the sake of that it's your idea. Being the initiator to work together. "I think a lot of times we overvalue technical skills and undervalue the people skills for people that are in technical positions." Being able to work well together with other people. Being able to work in a positive way with people. "I'm a lot more willing not to question somebody's judgement or talent and just let them do because I have to understand that they're a talented person too, and they're going to do the right things as opposed to before thinking, my way." "The ability to work with different people from different areas that have different concerns from mine and yet be able to work with them towards a common end. "I tend to work best in groups with discussion about things."	6	6	1.0000
Skills / Techniques	Comfort with multiple structured techniques. Debugging. Certain technology choices. Technical and software development background.	Ability to write quick utilities — Led to quick tests. Skill in using xdb. The object-oriented approach helped here - we could add people more easily." Troubleshooting ability. Comfortable with the tool environment. "There probably aren't that many people ... who can run through assembly language these days and do it with any kind of panache." Code reading - being able to read another engineer's code and understand it. Reuse.	6	5	1.0000

Thinking	Thinking algorithmically and structuredly. Being able to see the basic theory and basic structure. Ability to build models in your mind. Ability to structure problems. Analytical skills. Ability to go in and identify what the problem is. Ability to find solutions to problems. Ability to visualize what's going on in the whole system. Ability to understand the design. Ability to look at a problem and produce a solution matched to the problem. Spends time thinking. Being able to think on your feet. Thinking of alternative solutions. Skills in terms of breaking a design into smaller chunks and approaching it in a systematic way. Being able to follow structure and protocol.	"I think that structuring problems is related to the building of models." "I was able to understand the design, the communications, the protocols, and the data interchanges." "I've been thinking about that at various, different levels, you know, hacking on it myself, looking at competitors' things, other people in the company what they have done, and things like that. So I felt pretty well versed on the problem." "I like to get the problem down to something that I can get in my head and munch it over in my head." "I'm very good at really working through it, the detail and boundary conditions and what makes it work, what makes it break, and that kind of thing." "I'm a fairly good problem solver, I think I can gather information and extract the part of the information that's relevant to my problem and then apply it to a solution." "To me, engineering is problem solving." "Being able to pick up on the protocol or standard and conform to that and getting a quick understanding of it."	5	4	1.0000
Communication	Making people understand that I had heard them, and rather than just letting them walk away, we would talk through some of the issues. Constant communication with other engineers. Receptive. Open-mindedness. Responsive to new ideas. Communication one-on-one and within groups. Interpersonal skills. People skills. Inter-organization communication. Being open. Reasonable diplomacy or sensitivity.	I would try to make them see some of the points that maybe they hadn't thought about. They were more comfortable with the fact that I listened when a decision came out and it didn't mesh with the input they had given. "Being able to sit down with the marketing people and play marketing guy for a while." "Communication is paramount, technology has been less important."	4	4	1.0000
Learning	Being able to pick up new techniques quickly. Specific training. Willingness to learn and train yourself. A belief that you can pick up knowledge in any new area. Focus on improving skills.	Advanced course in microprocessors. "It's so interdisciplinary — you've got to be willing to do anything and believe you can do it with training as well as the people who are already doing it." Taking software classes - fitting classes together. "I can start from nothing and accumulate the databases that I need and sift from that the information that really pertains to the problem, and then organize that data in a way that then I can solve the problem."	4	3	1.0000
Desire to do	Bias for action. Sense of urgency. Desire to do the job. Results oriented. Drive. Get going on something. Try things.	"If I hadn't pushed, I would not have done this particular project that quick." "I'm here because I want to be." "I want to see results at the end of the day." "Just jump in and start working - you develop the capability."	2	3	0.5820

Attention to Detail	Ability to deal with complexity. Detail oriented. Writes everything down.	"I can keep track of a lot of that kind of stuff in my head, as far as mentally integrating or synthesizing a system view, say of the design, and eventually coming out then with realizing that on paper." "I also write in my lab notebook any little thing that goes wrong or any little anomaly or anything, I write that down in such a way that I can go back. ... I'm very conscious of 'where there's smoke, there's fire.'" Look at corner cases. I'm "good at taking a lot of detail from several different things and getting it into my mind all the same time and kind of working on it and munching it around in my head."	2	2	1.0000
Thoroughness	Making sure all paths are covered. Being methodical. Being organized. Being overcautious.	"I just had a real fear of ever being in that situation [lots of defects in code]. ... I just went slowly and real carefully and overdid if anything."	2	2	1.0000
Innovation	Creative ideas.	"I like to think of alternatives, being creative and ... practical a the same time."	2	2	1.0000
Conviction	Belief in the project or product. Doing the right thing. Selling projects.	"We had to go out and sell this to other divisions, we even had to convince one division ... to drop a proprietary system they were working on in favor of this." "I have to stick to my convictions." "The biggest thing there is being aware of just what for your own given organization, how does that work?"	2	2	1.0000
Experience	Prior experience with similar project.		1	2	1.0000
Prototyping	Approach of demonstrating feasibility	Start to implement the most feasible alternative.	2	1	1.0000
Seeks help	Values encouragement by other people.	"Coming up with questions that I couldn't find the answers to and then going and asking other people." The ability to know when you don't have enough knowledge and to go get help. Allow people to criticize or give new inputs. "I just look at what other people have done and ask them why."	0	4	0.0867
Desire to improve things	Not being satisfied with the status quo. Setting high personal expectations and goals.	Constantly looking for better tools, better approaches, and better technologies. Giving yourself time for improvement.	3	0	0.2105
Challenge	Enjoying working in new areas. Curiosity.	"The customers would have more functionality or better performance and that it would be actually challenging and interesting job."	1	3	0.4737
Scheduling	Planning ahead. Schedule setting ability.		2	0	0.4737
Simplicity	Simplifying things as much as possible. Not letting it get too complex. Trying to make it simple. Aversion to complexity.	"I was trying to purposely stay away from very complicated solutions." "A little bit of vision that we could do things by rethinking things we could do things simpler and more elegantly."	0	2	0.4737

Quality	Making sure the result is readable Concern for reliability. Commitment to high quality.		2	0	0.4737

F Survey Instructions for Participants

DIRECTIONS

1. Biographical Questionnaire

   The objective of this questionnaire is to determine if any demographic, educational, or experience variables best characterize the Company's Software Engineering population.

   Please complete the questionnaire indicating the information requested. In all cases, make your best attempt to answer the questions. If you get stuck on any questions, please get in touch with me. Save the **RESULTS OF THE SORTING EXERCISE** portion until you complete step 2.

2. Competency Sorting Exercise

   The objective of this exercise is to determine which job competencies identified in the Phase 1 research best characterize the Company's Software Engineering population. You will sort these competencies based on how well they describe your behaviors on the job, especially when you're performing at your best. Try to think of the best software experience you've had and use that to guide selection of which attributes best describe your behavior on the job.

   - Be sure that you have a clear desk or table to work on before you start. You will be placing 3 x 5 cards in one of 7 piles so you need space to spread these out. Find the supplied pile markers in the envelope and lay these out on your table in order from number *6* on your leu to number *0* on your right. These pile markers are annotated to remind you that column *6* represents those competencies that are most like your behavior and column *0* represents those competencies that are least like your behavior.
   - Read through all 38 competency cards to become familiar with them.
   - Sort all of the cards into 3 piles of any number of cards. Place to the left the cards which include the competencies which best describe your behavior in the process of software engineering. Place to the right those cards which include competencies which least describe your behavior in the process of software engineering. Place those cards with competencies about which you are unsure in the middle pile.
   - During the sorting you will spread the items in piles under the pile markers, while maintaining the general left-center-right relationships.
   - Select the 2 items which most strongly relate to your behavior on the job as a software engineer. Think in particular about those tunes which have been a personal best for you. Place these two cards under the column marker labelled *6*. The order of these cards under the marker is not important. All will receive the same score.
   - Now select the 2 items which least reflect your behavior on the job as a software engineer. Place these under the column marker labelled *0*.
   - Continue in this way, alternating between the left and right sides of the distribution, placing the indicated number of cards below each column marker. Feel free to move any card at any time should you change your mind about which competencies are most closely related to your actual behavior. All that matters is that the right number of cards eventually are found beneath each column marker. Try not to take too long agonizing over the placement of any one card. Your first impulse for placing the card is probably the best. If it helps, you can jot a short phrase that captures the essence of the competency directly onto the card as a prompt to use in sorting.
   - Review your groupings to be sure that they accurately reflect your behavior while completing your software engineering assignments. Move any cards you wish to better reflect which competencies most apply to you doing your job. Now record the item identification numbers found in the lower right hand corner of each card in the appropriate column on the back of the **BIOGRAPHICAL QUESTIONNAIRE.**
   - If you have any questions, don't hesitate to give me a call at T-229-2340 to ask for help.

3. Mail the completed **BIOGRAPHICAL QUESTIONNAIRE** and all survey materials to me via interoffice mail in the pre-addressed envelope provided.

THANK YOU FOR YOUR PARTICIPATION!

G Phase 2 Biographical Questionnaire

Sex	M F
Age	
HP Division	

For each degree that you have completed or begun, please complete the following. Circle the appropriate response or fill in the blank.

Degree	ASSOCIATE BS BA MS MA PhD	OTHER:
Major	CS CE EE MATH PHYSICS	OTHER:
Status	COMPLETED PARTIALLY-COMPLETED	
School		

Degree	ASSOCIATE BS BA MS MA PhD	OTHER:
Major	CS CE EE MATH PHYSICS	OTHER:
Status	COMPLETED PARTIALLY-COMPLETED	
School		

Degree	ASSOCIATE BS BA MS MA PhD	OTHER:
Major	CS CE EE MATH PHYSICS	OTHER:
Status	COMPLETED PARTIALLY-COMPLETED	
School		

Degree	ASSOCIATE BS BA MS MA PhD	OTHER:
Major	CS CE EE MATH PHYSICS	OTHER:
Status	COMPLETED PARTIALLY-COMPLETED	
School		

In the **LAST 2 YEARS,** how much training have you completed relevant to software engineering including any formal study toward a degree? The unit of measure is 'contact hours.' That is, time spent in a classroom for formal training, and time spent studying for self-study. Hence a semester long college class might be expressed as 3 contact hours a week for 15 weeks for a total of 45 contact hours. Also include corporate training classes, consultant seminars, and self-study. The table below is designed to jog your memory about classes taken. The only number to be used in the study is the **TOTAL Training Contact Hours** below.

# CONTACT HOURS	COLLEGE	CORPORATE	SEMINARS	SELF-STUDY	OTHER

TOTAL Training Contact Hours [ ]

(Continued on back...)

Complete the following table with the number of years of experience at the Company and elsewhere indicating whether your work was Software Engineering or not. Include any full time employment once beginning your professional career. Do not include part-time jobs while in school, co-ops, SEED positions, research assistantships, or teaching assistantships. Hence jobs between degrees count while summer jobs during college don't. If you were managing an activity during this period, also include this tune in the appropriate cell. The total of all values in the table should sum to your total years of experience.

# YEARS EXPERIENCE	SOFTWARE ENGINEERING	NOT SOFTWARE ENGINEERING
COMPANY		
NOT COMPANY		

TOTAL Years Experience ____

For each programming language used professionally, complete the following: *(Count language variants as a single language. For example, C and C++ are considered as one language. Also, various forms of Assembly language are considered as one language.)*

LANGUAGE	SKILL LEVEL
	HIGH LOW
	HIGH LOW
	HIGH LOW
	HIGH LOW
	HIGH LOW
	HIGH LOW
	HIGH LOW
	HIGH LOW

Self-described skill level:

HIGH: expert, lots of experience, well versed, used on many projects

LOW: novice user, little experience, used on few projects

Please fill in the following table after completing the attached statement sorting exercise. Place the appropriate item numbers in each column of the table, in the same category into which you sorted your cards.

RESULTS OF SORTING EXERCISE

Most Like My Behavior						*Least Like My Behavior*
6	**5**	**4**	**3**	**2**	**1**	**0**
(2)	(4)	(7)	(12)	(7)	(4)	(2)
____	____	____	____	____	____	____
____	____	____	____	____	____	____
	____	____	____	____	____	
	____	____	____	____	____	
		____	____	____		
		____	____	____		
		____	____	____		
			____			
			____			
			____			
			____			
			____			

H Phase 2 Competency Statements

DEFINITION

I value the synergy of group efforts and invest the effort required to create group solutions, even at the expense of my individual results.

KEY BEHAVIORS

- I balance the strengths and weaknesses of other team members.
- I promote constant communication among team members using techniques such as brainstorming sessions, travel, phone calls, e-mail, or just being physically close to the rest of the team.
- I recognize synergy of group efforts and invest personal time and energy to leverage it.

Item #1

DEFINITION

I proactively seek the assistance of others in learning, researching, designing, understanding, debugging, or checking results.

KEY BEHAVIORS

- I ask previous implementers to explain their designs.
- I ask other engineers to critique or evaluate my designs.
- I survey others to create lists of alternatives.

Item #2

DEFINITION

I spend a significant amount of tune assisting others in the completion of their tasks or influencing broad organizational direction.

KEY BEHAVIORS

- I act as a lab-wide consultant for process or product issues.
- I review, direct, or influence the work of other engineers.
- I assist other engineers with their tasks in an effort to complete the project.
- I teach engineering skills to other engineers.

Item #3

DEFINITION

I use a prototyping method to assess key system parameters before designing the final product. I avoid using my prototypes as final implementations.

KEY BEHAVIORS

- I use prototypes as a mechanism for incremental development of a product.
- I attempt to not allow my prototypes to become the final product.
- I use prototypes to assess critical areas like performance and tune estimates.
- I prototype in parallel with the detailed design phase.

Item #4

DEFINITION

I apply incremental testing techniques during code development such that a given module achieves a high degree of reliability by the time it's completed.

KEY BEHAVIORS

- I create tests in parallel with the creation of code.
- I automate tests so they can be run frequently throughout design.
- I test frequently during development to ensure reliability at the module level.

Item #5

DEFINITION

At the time of assignment, I possess the unique skills or knowledge required to accomplish the task at hand.

KEY BEHAVIORS

- I possess the necessary domain knowledge and skills required for the job.
- Prior to an assignment, and on my own initiative, I become an expert in a given area.

Item #6

DEFINITION

I actively seek the training needed to complete the assigned task.

KEY BEHAVIORS

- I seek out documentation required to understand my current assignment.
- I take classes which will directly help in the completion of the current assignment.
- I keep current by reading trade or technical journals.
- I improve my skills and awareness by attending conferences.

Item #7

DEFINITION

I proactively attempt to leverage other engineers' effort by using their code or designs. I attempt to leverage my own effort by making my code reusable.

KEY BEHAVIORS

- I look for code or code fragments which can be reused.
- I design and code so that my effort can be reused.

Item #8

DEFINITION

I take advantage of the tools and techniques of structured design in order to understand and communicate designs, but do not necessarily follow the complete formalism of the approach.

KEY BEHAVIORS

- I use structured techniques (such as Structured Analysis and Design, Hierarchy Charts, ...) as a means of joint development and communication.
- I use structured techniques as a mechanism for passing off a design or part of a design to another engineer for implementation.
- I view structured techniques as just another tool which can be applied to certain problems, rather than as a panacea.

Item #9

DEFINITION

I use a methodical approach in understanding and solving problems.

KEY BEHAVIORS

- I build mental or physical system models to enhance my understanding and visualization of the problem.
- I design well controlled experiments to efficiently resolve problems.
- I invest in the development of test tools to solve problems.

Item#10

DEFINITION

I seek to improve performance or results through the use of new tools or methods.

KEY BEHAVIORS

- I proactively seek out new tools or methods to solve problems.
- I use my work assignment as a way to learn new tools or methods.
- I recognize value in new tools or techniques.

Item#11

DEFINITION

I show a strong concern for schedules and I estimate schedules well.

KEY BEHAVIORS

- I maintain personal "rules of thumb" for schedule estimation.
- I refine schedule estimates based on my measured progress.
- I schedule via task breakdown and successive refinement.
- I meet schedules.

Item #12

DEFINITION

I use code reading and other group development techniques to ensure final code quality.

KEY BEHAVIORS

oI participate in the code reading of other engineers' work.

oI ask for others to code read my work.

oI participate in brainstorming and other group development techniques.

Item#13

DEFINITION

I use decomposition design techniques relying on visual representation of designs. I create structured designs, generally without using formal techniques.

KEY BEHAVIORS

oI keep design specification in sync with the actual implementation.

oI follow a *top-down* design method using decomposition to successively refine the design.

oI take a modular approach in order to reuse as much of the design as possible.

oI use pictures to communicate and understand designs.

Item#14

DEFINITION

I consider customer or user input and feedback to be an essential ingredient in the design of products.

KEY BEHAVIORS

oI proactively attempt to obtain customer and user input and feedback for products.

oI measure project success in terms of customer satisfaction.

Item #15

DEFINITION

In response to schedule pressure, I am forced to sacrifice important parts of the design process.

KEY BEHAVIORS

o In response to schedule pressure I am forced to provide incomplete documentation.

o When schedules slip, I do not have time to adequately inspect or test the product.

o When pushed to pull up a schedule, I will not prototype or adequately design risky parts of product.

Item #16

DEFINITION

I create solutions which are elegant and simple and allow for easy extension to future needs.

KEY BEHAVIORS

oI value simple solutions.

oI design general solutions which will be easily extended, even if it's not currently needed.

oI apply structure to ill defined problems and problem domains.

Item#17

DEFINITION

I take pride in producing defect free products on schedule in minimum time.

KEY BEHAVIORS

oI take pride in meeting or beating schedules.

oI take pride in achieving low defect counts.

oI take pride in achieving high productivity and accomplishing significant amounts of work over short periods of time.

Item #18

DEFINITION

I take the initiative to identify ways of completing important tasks. I influence others to consider alternative approaches.

KEY BEHAVIORS

- I proactively complete projects or tasks that I consider important.
- I influence others in design, organizational structure, ...
- I identify ways to surmount barriers and remove obstacles.

Item#19

DEFINITION

I proactively attempt to influence project direction by influencing management.

KEY BEHAVIORS

- I discuss issues concerning other engineers with my manager.
- I attempt to set project direction and make project decisions by influencing my manager.
- I make specific resource or assignment recommendations to management.
- I promote product ideas through demos or selling of ideas to management.

Item #20

DEFINITION

I value the sense of accomplishment which conies from making a direct contribution.

KEY BEHAVIORS

- I seek assignments where I can contribute.
- I feel rewarded by the chance to contribute.

Item#21

DEFINITION

I enjoy the challenge of the assignment and the sense of accomplishment from completing it. I just plain have fun at work.

KEY BEHAVIORS

- I look forward to going to work.
- I derive a sense of accomplishment from work.
- I enjoy the challenge of a tough assignment.
- I am driven by the reward of doing something new and different.

Item #22

DEFINITION

I am driven by a sense of mission and clearly articulated goals to achieve a specific result.

KEY BEHAVIORS

- I create and articulate clear and specific goal statements.
- I drive the project to achieve specific goals.

Item#23

DEFINITION

I stress the solution over the source of the solution. I don't care where a good idea comes from and don't feel the need to promote my own ideas.

KEY BEHAVIORS

- I focus on the end result, regardless of who creates the solution.
- I allow use of a *discovery process* for others to come to see the value of my ideas.
- I discuss ideas, not positions.
- I allow others to re-write an idea I've created.

Item#24

DEFINITION

I exhibit and articulate strong beliefs and convictions. I act in accordance with these beliefs, even when it is counter to specific management direction.

KEY BEHAVIORS

- I act in accordance with my beliefs rather than solely based on my assignment.
- I risk my performance ranking in an effort to secure the best solution.
- I argue forcefully for a specific point of view.

Item #25

DEFINITION

I mix my personal and work goals by seeking or tailoring assignments to my professional interests.

KEY BEHAVIORS

- I identify positions I would like to have and lobby to receive them.
- I seek assignments which will further my professional development.
- I identify technical areas which I'd like to develop and find ways to apply them to the project at hand.

Item #26

DEFINITION

I confront others when necessary to ensure a good design or product solution.

KEY BEHAVIORS

- Rather than letting a conflict simmer, I will openly confront another person in an effort to resolve it.
- I will raise a tough issue of conflict with another engineer to my manager in an effort to have it resolved.

Item#27

DEFINITION

I am thorough in my assignment and persevere until it's completed.

KEY BEHAVIORS

- I drive hard to complete even the tedious parts of my assignment.
- I like to see things done cleanly and correctly.
- I stick with assignments even when it's not clear that they're going anywhere. At least I'm eliminating things and making the problem smaller.

Item#28

DEFINITION

I have mastered the skills and techniques necessary for good software design and implementation.

KEY BEHAVIORS

- I have a strong technical and software development background.
- I am comfortable with multiple software design and implementation techniques.
- I have very strong software development skills.

Item#29

DEFINITION

I posses strong analytic skills that allow me to visualize a complex problem and create alternative solutions

KEY BEHAVIORS

- I am able to see basic theory and basic structure in a problem.
- I am able to visualize what's going on inside a complex system.
- I am able to break a large, complex problem into smaller, more manageable chunks.

Item #30

DEFINITION

I am driven by a strong bias for action and sense of urgency in completing my assignments.

KEY BEHAVIORS

- When faced with a tough problem, I don't hesitate to get started. I develop the required capability as I go.
- I am results oriented and want to make progress on a regular basis.
- I push myself to achieve results quickly.

Item#31

DEFINITION

I am detail oriented and able to deal with very complex problems.

KEY BEHAVIORS

- I'm good at taking details from a lot of different sources and determining a good solution to a problem.
- I keep track of lots of detail, either in my head or on paper.
- I concern myself with "corner cases" and other seemingly insignificant data, since this is often where the breakthrough comes from.

Item #32

DEFINITION

I am very methodical, organized, and cautious in my work.

KEY BEHAVIORS

- I make sure that all paths are covered in my design and problem solving.
- I work slowly and carefully to avoid making mistakes.

Item#33

DEFINITION

I am innovative in my solutions to problems.

KEY BEHAVIORS

- I like to create alternatives which are both creative and practical.
- I have creative ideas and solutions to problems.

Item #34

DEFINITION

My prior experience with similar projects leads to my high performance on current projects.

KEY BEHAVIORS

- I use the experience gained on one project to improve my solutions on subsequent projects.
- I find that the skills I learn on one project are directly applicable to my next project.

Item#35

DEFINITION

I set high personal expectations and goals. I am not satisfied with the status quo.

KEY BEHAVIORS

- I am constantly looking for better tools, better technologies, or better problem solving approaches.
- I give myself time to improve.

Item#36

DEFINITION

I have a high concern for reliability and a strong commitment to quality.

KEY BEHAVIORS

- I make sure that my products have few, if any, defects.
- I work hard to be sure that my results are clear and readable to others.

Item#37

DEFINITION

I maintain a broad "big picture" view of my projects in an attempt to influence the project direction.

KEY BEHAVIORS

- I remain aware of what other engineers are doing and suggest ways to better achieve project objectives.
- I try to be sure that project goals make sense, and work to change them if they don't.
- I try to fit my project into the broader scheme of division programs.

Item #38
