Belbin's Company Worker, The Self-Perception Inventory, and Their Application to Software Engineering Teams

by

Peter Klaus Schoenhoff

Thesis submitted to the faculty of Virginia Polytechnic Institute and State University in partial fulfillment of the requirements of the

MASTER OF SCIENCE

degree in

Computer Science / Computer Science Applications

APPROVED:

_____

Dr. Sallie M. Henry, Chairperson

_____

Dr. Manuel Pérez-Quiñones

_____

Dr. Stephen Edwards

December 11, 2001
Blacksburg, VA

Keywords: software engineering, metrics, teams, team roles, Belbin

Belbin's Company Worker, The Self-Perception Inventory, and Their Application to Software Engineering Teams

by Peter Klaus Schoenhoff

Abstract

Software engineering often requires a team arrangement because of the size and scope of modern projects. Several team structures have been defined and used, but these structures generally define only the tasks and jobs required for the team. Various process and product metrics seek to improve quality, even though it is generally agreed that the greatest potential benefit lies in people issues. This study uses a team-based personality profiling tool, the Belbin Self-Perception Inventory, to explore the characteristics offered by the Company Worker, one of the team roles defined by Belbin.

This work is dedicated to my wife, Holly.

Acknowledgements

I wish to offer sincere appreciation to those made this effort possible:

My Lord, Jesus Christ, without whom nothing was made that was made.

My beautiful wife Holly Schoenhoff, who provided not only support and patience but also was willing to proofread and occasionally dig though my stacks of notes for pieces of information I'd lost but knew were there somewhere. I love you, Holly!

My family: Klaus and Linda Schoenhoff, Bill and Carol Bernard, Katie, Barbi, Robby, and Jennifer, for understanding and support.

My advisor, Dr. Sallie Henry, for guidance and encouragement.

My committee members and other faculty who were always willing to let me bounce questions off of them: Dr. Manuel Pérez-Quiñones, Dr. Steven Edwards, Dr. John Carroll, and Dr. Mary Beth Rosson.

My consultants in the statistics department, Michelle Marini and Kevin Shropshire, without whom I would still be lost in the woods.

Mr. Wes Lloyd for his ongoing collaboration and for serving as an observer for these experiments.

Mr. Eran Hollander for assistance in understanding the psychological perspective.

The students of the Spring 2000 and Fall 2000 undergraduate Software Engineering classes at Virginia Tech, who were gracious enough to participate in such a lengthy study.

The students of the Fall 2000 and Spring 2001 Operating Systems classes at Virginia Tech, who allowed me to steal their class time for the sake of study surveys.

And Todd Stevens who did the initial work in this area, giving me a starting point.

Table of Contents

List of Multimedia Objects

Figures

Tables

Chapter I: Background And Motivation

A) Goals of This Research

Since the late 1960's, several functional team structures have been designed and adopted. These structures define jobs and tasks required for each functional role on the team. The aspect of personality traits as a factor in software development team interactions has been largely neglected, though. If a matching between personality traits and functional roles for software engineering teams could be achieved, each team members' work would become more natural and effective. The benefits for team interaction are easily imaginable.

This study, as part of a larger series of investigations into computer science teams at Virginia Tech, addresses one particular personality type. Meredith Belbin defined this type, known as the "Company Worker," in his 1981 book Management Teams [BELR81]. Known as the implementer of the team, the Company Worker's contributions to a team, according to existing theory, suggest that this type would be of particular benefit to software engineering teams. This study seeks to address the following questions.

1. Do the presence and number of Company Workers on a software engineering team affect the team's success, effectiveness, and viability?
2. Are certain roles drawn to particular stages of the software engineering process?
3. Can Belbin's team roles, applied to software engineers, be defined or described using the Keirsey Temperament Sorter?
4. Can roles thought to be non-existent in software engineering contexts be seen to emerge in Belbin retests? (This question also involves test-retest validity issues.)

This study yielded the following high-level results, which are supported and evidenced in the body of this research.

1. A software engineer's personality has an effect on his team's success;

2. There is a difference in the nature of Belbin's team roles as applied to software engineering teams, as opposed to management teams;
3. Whereas Belbin's Company Worker is useful in a management team setting, the role makes a software engineering team less predictable;
4. In addition to adding unpredictability to a team, the Company Worker role in software engineering is also unpredictable, in that it can not be distinguished from other software engineers with regard to the Keirsey Temperament Sorter;
5. Various roles seem to be drawn to particular stages of the software engineering process;
6. The Belbin Self-Perception Inventory, as published in his 1981 book, is useful, but has not been validated, and should be taken with a grain of salt;
7. The test-retest validity of the Self-Perception Inventory suggests that either the test is problematic or a different testing method is warranted. The test may have to be taken by subjects in pre-existing long-term teams, for example;
8. A person's team role may change over time, or from team to team.
9. Trends between the Belbin Self Perception Inventory and the widely accepted Keirsey Temperament Sorter suggest that the Self Perception Inventory is measuring *something*.

B) A Silver Bullet?

Many researchers have suggested that the greatest potential for improving software quality and process lies in personnel issues, rather than in technological or methodological concepts [BROF87] [KELM91] [CURB91] [BOEB81]. Several prominent industry leaders support this assertion, suggesting that an understanding of personality, identification of key traits, and management issues and techniques are the next potential areas for productivity and quality improvement [YOUE92] [BOEB81]. The uniting theme is that the greatest potential gain for productivity seems to be in people, rather than in tools. Brooks alleged in 1987 that "no silver bullet" exists in methodology or technology [BROF87]. A few researchers seem to suggest that it may exist elsewhere, in people issues [KELM91], including Boehm, who writes, "Personnel attributes and human relations activities provide by far the largest source of opportunity for improving software productivity" [BOEB81]. If such a bullet is to be found, people issues are the next logical areas to search.

As an aside, it should be acknowledged that many significant technological advances *have* been made in software engineering – some since Brooks' classic paper – each of which *could* be considered a "silver bullet" of sorts. Structured programming, object oriented design, fourth generation languages, testing methods, architecture-level concepts, and quality control ideas have all helped to improve the state of the industry. However, none of these ideas *alone* is enough to guarantee quality, or even improvement. Indeed, none of these concepts is *applicable* in every imaginable type of project. Brooks was right. If a quantum leap is to be made in productivity in the industry, it is by exploring and managing the personal aspect of software engineering, rather than developing and improving technological tools.

Before the software industry aspires to attain a profound breakthrough in productivity by engaging the personal characteristics of individual programmers, an idea of management strategy should be developed. Qualitative observations and research are useless unless they can provide guidelines for controlling desired effects to maximize productivity, quality, morale, or other management goals. This is only made possible by providing some tool for measurement of the factors being controlled, and those that are in turn to be affected by the control. As DeMarco said, "You can't control what you can't measure" [DEMT82]. If personality attributes are to be used to affect productivity, then a reliable means of measuring the personalities that are present on a development team must exist, along with a reliable means of measuring team productivity. This necessitates at least peripheral attention to the area of metrics.

C) A Brief History of Metrics

Many metrics devised to aid software engineering measure some attribute of the product or process. Both of these types of metrics have been applied at both the design and code level. For product management, myriad metrics have been designed to describe structural complexity likelihood of errors, productivity,

and many of the "-ilities,"[GLAR91] [BOEB78]. Early proposed product metrics, like LOC/KLOC were easy to measure and understand, but offered little in terms of interpretive value, aside from obvious conclusions (i.e., a longer program is probably more complex). Adaptations of these metrics are common (LOC per function, cost per KLOC), but the value of these simple metrics is widely debated [JONC86].

Albrecht instead proposed function-oriented metrics, like the function point [ALBA79], which measure functionality based on values such as the number of files used and the number of distinct outputs to the user. Jones used the function-point concept to demonstrate that counting LOC is largely language dependent [JONC98].

Halstead proposed a system to measure code complexity utilizing analysis of the language used for the program [HALM77]. (The term "language" here refers to the formal vocabulary and grammar used by the programmer, including variable and function names, rather than simply the programming language itself.) This complexity measure estimates the readability and understandability of the program code. Halstead's system measures features such as vocabulary size, program volume, and "effort" by counting simpler quantities (like number of unique operators and operands) and applying them to prescribed formulae. One poor aspect of this system is the lack of intuitiveness of the formulae.

Good metrics are generally directly aimed at improving programming practices and are backed up both theoretically and statistically. Metrics examining areas such as shared data and global variables are strong examples of this concept. Application of these metrics guide the programmer to more reliable code and coding practices in a way that is both readily understandable to the programmer and validated by experimental research.

Some product metrics aim at predicting problem areas at the design level. An early example is McCabe's Cyclomatic Complexity [MCCT89] [MCCT94], which traces the number of possible paths that can be followed through all the decision statements in a segment of code. Minimizing the Cyclomatic Complexity of a program or function makes it more easily understandable, and thus less likely to introduce errors. Other metrics involve interconnectivity between functions, such as the Fan-In-Fan-Out Metric [HENS81]. This system counts the number of functions calling the measured function and the number of functions called by the measured function. More complex metrics combine and improve existing concepts, like the Henry-Kafura Metric, which combines Fan-In-Fan-Out with number of parameters passed [HENS81].

Kemerer and Chidamber's work on the MOOSE metrics for object oriented systems include such measures as Depth in the Tree, Coupling Between Objects, and Number of Children [CHIS04]. Often these can be determined by a quick visual parsing of a good design structure chart, so that predictive metrics can be used and applied before any code is actually written. This has the obvious advantage of allowing problem areas to be detected at an early stage, minimizing the investment in a problematic design. Because at first the metrics were not statistically validated, their interpretive value was somewhat suspect for a time; however the claims were eventually confirmed [LIW93].

Process management has been addressed with more loosely defined tools, metrics, and guidelines. These systems aim to ensure that the method that a software production group adopts produces quality results and that the adopted method itself is actually followed by the developers. One recent example is the Capability Maturity Model (CMM) designed and adapted by the Department of Defense [PAUM93] [CURB97]. The CMM requires surveys and detailed examination of the process used by the development organization by a highly trained group of investigators, and results in a score on a five-point scale. (See Figure 1.1) Most companies in the United States were operating at level 1,

according to Humphrey [HUMW89]. The Department of Defense currently requires a level 2, and only a small handful of organizations have been honestly assessed and verified as operating at level 5. [CURB97]

Earlier systems included Total Quality Management (TQM), in which organizations move though a series of steps, each of which focuses on a specific set of process areas. An organization's current focus step is a measure of their current process quality. [PRER01] [ARTL92]

Software Quality Assurance (SQA) deserves mention here, but rather than being a process metric itself, it is a group assembled for the duration of a project that prescribes a system tailored to ensure the quality of the process for the specific project. The SQA group may (and usually should) incorporate process metrics as appropriate, which are monitored and used for the purpose of steering the process toward quality. Their recommendations may take the form of Statistical Software Process Improvement (SSPI), which aims to detect problem areas in the development process by analyzing the trends of prior development efforts.

D) Programmer Metrics

Humphrey advocates that the process metrics be aimed at individuals in his personal software process approach [HUMW95], explaining that different workers may require different processes, simply because of personal differences. This work acknowledges the fact that personal differences exist, and can have an effect on the overall process (and hence the product), but the differences between individual engineers are not resolved in a way that can be measured or managed.

It is the personality characteristics of the software development personnel that are not being addressed. No hard-and-fast guidelines or tools exist for maximizing the effectiveness of a team by addressing which *types* of people are

present on the team. Which types of people work best together? Which types are better at which jobs? Most importantly, how can this be predicted and used to the advantage of a development team? If the greatest potential for improving productivity is in the personal attributes of the software developers, then it makes sense to research and determine what metrics for those characteristics may be beneficial for affecting change.

This deficiency (the virtual non-existence of programmer metrics) is easily illustrated in Paulish and Carleton's determinants for software quality [PAU94]. Paulish and Carleton describe Process as a triangle (shown in Figure 1.2), which connects Product, Technology, and People. All of these things are measurable except people factors, which play a role in affecting customer characteristics and the development environment.

What can be found in a programmer's essence that could prove beneficial to software engineering? Is anything there to be found? Sackman *et al*. [SACH68] determined a 10:1 ratio in productivity among programmers with similar training and years of experience. More research estimates a 20:1 ratio in productiveness among computer science professionals in general [CURB91], and Boehm states that 26:1 ratios have been observed [BOEB81]. These observations alone are enough to conclude that some programmers are more productive than others. What do the programmers at the high end of that spectrum have in common? Can they be identified in advance? Just as important is the issue of what desirable qualities these programmers lack. In any case, the difference exists within the person, not within the tools.

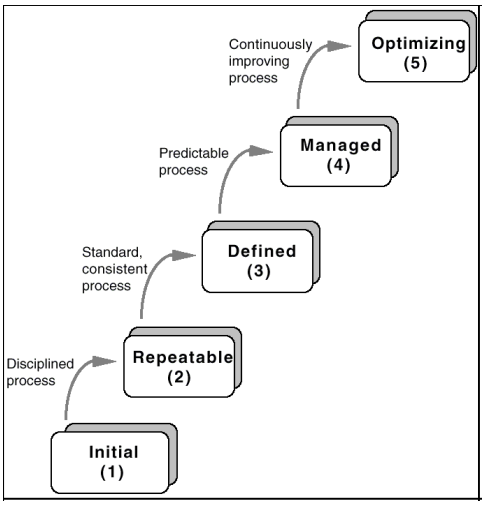


Figure 1.1 The Capability Maturity Model [PAUM93]

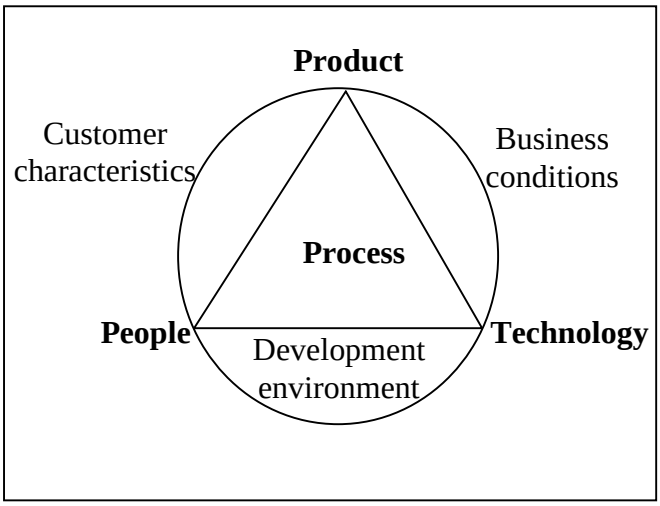


Figure 1.2: Determinants for Software Quality [PAU94]

As Boehm argues, "Personnel attributes and human relations activities provide by far the largest source of opportunity for improving software productivity" [BOEB81]. Better ways must be found to promote individual effectiveness and more productive team interaction. Personality is sure to have an important impact in both of these areas. A programmer's personality certainly plays a part in the ease with which he can learn and understand tasks, interpret needs, and communicate with customers or colleagues, not to mention the internal motivation he feels toward the various activities associated with programming. From a teamwork perspective, personal factors would not only influence the abilities of a programmer, but also the manner in which he cooperates and collaborates with his teammates.

A software development manager or team project leader must understand the effects of his individual programmers' personality traits in the context of the development team if he is to address problems quickly and effectively [CURB86]. Motivational tactics applied by management are also affected by personality, and can even backfire if poorly planned. For example, a manager attempting to motivate an employee may comment on the employee's good work. One employee may take the remark as a compliment, but another employee may

perceive the same words as an insult [BELR81]. Management has much to gain from psychology to understand where and why slowdowns occur. If putting certain combinations of people together affects quality, environment, or project velocity [BECK99], management would do well to take heed.

The psychological aspects of software engineering are not merely of local interest. Curtis points out that automakers consistently experience greater productivity improvement though effective workforce management than though any of the technological improvements in which they invest [CURB91]. Three of Curtis' four suggestions for increasing a software organization's body of collective knowledge are based in management or psychology [CURB90].

Some investigations have explored the technical fields using the Myers-Briggs Type Indicator (MBTI), but have determined little, aside from the fact that the majority of the subjects tested were introverted (rather than extroverted) and thinking oriented (as opposed to feeling oriented) [LYOM85] [BUIE88] [BUSC85] [SITS84]. Williams' recent work notes that approximately half of her sample of computer science students scored high for Introversion and Judgment on the MBTI [WILL00]. Thomsett's work made use of three tools, including the MBTI and Belbin's Team Role Theory, which is of special importance to this current research. (Interestingly, while Bush and Schadke noted that personnel managers often tend to hire employees whose MBTI type matches their own, Belbin noted the same concept with regard to his own research years earlier [BUSC85] [BELR81].)

E) R. Meredith Belbin and Teams

R. Meredith Belbin, a student of management science, published in 1981 his acclaimed book *Management Teams*, which describes personality types from the perspective of interactions between various "team roles" that are represented by the members of a management team [BELR81]. Biddle explains role functions

as the "behavioral repertoire" of a person, based on internal and external influence. The role a person assumes is reflected in his nature and in the external needs of his team [BIDB79]. The application of this concept to software engineering has been endorsed in various works, although actual empirical research in this area is still at an early stage. Yourdon, in *The Decline and Fall of the American Programmer* advocates the use of Belbin's roles, although he attributes the roles to Thomsett [YOUE92] [THOR90]. Stevens' and Henry's work at Virginia Tech began to explore the real feasibility of such a connection [STET98] [HENS99].

Belbin's approach, as contrasted to other personality profiling methods, is of special interest to software engineering for two reasons. First, the measurement of programmer types and analysis of such information is useless unless one plans to *do* something with that information. Any action or decision that makes use of these measurements implies active management, and management is the perspective from which Belbin's research and writing stems. (Belbin's work is key to the current study, and a more detailed overview of team role theory follows in Chapter 2.)

Secondly, software development projects are now large enough that an individual effort is generally not a practical (or possible) option. The context of the team is unavoidable, and it is wise to consider not only an individual person's predisposition to certain behaviors, but also how those behaviors will affect (and be affected by) the other team members' characteristic behaviors. Hackman's definition of a team, which identifies the "interdependence among members" and the "organizational context" is inherent to the nature of a software engineering team [HACJ90].

In the late 1960s and early 1970s, the computer science community began to realize that the most significant software development projects would require a team structure. Some, like Harlan Mills and Fred Brooks, developed team models

to describe the necessary functions on the team, as well as the organizational structure supporting the team. Various names were given to these models: the Chief Programmer Team, the Surgical Team, and the Egoless Programming Team, to name a few [BAKF72] [BROF95] [MILH71] [VONA84] [WEIG71]. Recently, a renewed interest in the Chief Programmer Team model surfaced. The recent advent of the Extreme Programming (XP) movement [BECK99] owes a substantial debt to Mills. XP teams, like Brooks' Surgical Team, are based in part on the Chief Programmer Team model. The merits of Mills' model are evident in the fact that the same concepts are revisited so frequently.

At this point a distinction should be made between the *functional roles* on a software engineering team and the *team roles*. The term *team role* is used in this current work to describe the aspects of the team members associated with personality and personal behavior characteristics, for the sake of consistency with Belbin's team roles. *Functional role* will denote the specific job performed by a team member – a code librarian, for example. Functional roles describe the team needs addressed by activities and tasks, whereas team roles describe the needs addressed by the individual nature of those people on the team. The programming team models mentioned in the preceding paragraph all define functional roles needed in order for the model to succeed. A short overview of the more popular software engineering team models is presented here.

F) Software Engineering Team Models

1) Individual Programmer

This concept is included here as a team model because of its relevance to the history of team models. In the late 1960s, as software projects became larger, management generally responded by attacking the problem with more people. This culminated in Brooks' Law, as described in his seminal work, *The Mythical Man-Month*. Basically, Brooks asserted that adding people to a project that is already late would make the project later. The number of communication

channels that must be maintained increases faster than any productivity speedup that could be gained by the extra manpower. Because of the overhead associated with healthy maintenance and use of those channels, any project must have a cap on practical team size.

Mills seems to have noticed and addressed this issue even before Brooks pinpointed the theory behind the problem. In 1968, Mills published a paper insisting that if greater quality and productivity were to be achieved, a better way to handle large projects was to use an individual programmer, rather than a large team [MILH83]. Mills approached this concept from a perspective of programmer responsibility, but the beginnings of his first formal team model had been planted. Mills apparently had noticed that existing teams had become unmanageable. Even though this first step didn't describe a team model per se (but rather the explicit absence of one), the issue of team manageability was addressed in this concept before any of the team models were formalized.

2) Chief Programmer Teams

Mills and Baker described a basic team of three: the Chief, the Backup, and the Librarian, with auxiliary members as needed. This notion was motivated primarily by the realization that small teams had the potential to be more effective than the large teams that had been previously employed for large projects [BAKF72] [MILH71]. These roles are defined as follows:

2a. The Chief

The Chief is a senior level programmer. He is responsible for direct construction, manipulation, and modification of the code.

2b. The Backup

The Backup has basically the same skills and nearly the same experience as the Chief. His primary purposes include secondary manipulation of code, as

well as serving as a resource through which the Chief can explore and evaluate ideas.

2c. The Librarian

The Librarian's duties include code archival and version control, as well as documentation of system decisions. This role necessarily contains aspects of both a secretary and a programmer.

2d. Others

The team may require other team members, depending on the size, scope and nature of the project. These additional members may fill whatever functions are needed, including coding of smaller functional systems.

Why does the Chief need a backup? The Chief and the Backup are both gifted and knowledgeable. Generally, the primary difference between these two team members is the amount of experience each possesses. (Incidentally, studies show breadth of experience, rather than length, to be an effective predictor of programmer productivity [CURB91][SHES79].) Usually, the Backup is being groomed for a future role of Chief. The Backup serves as a second opinion, check, or devil's advocate to the Chief, and the two generally prevent each other from becoming dull, as iron sharpens iron.

The Librarian serves to keep the Chief and the Backup from being distracted by mundane details and overhead. In real world projects, a very significant portion of time is spent on non-programming overhead that surrounds any large project. The Librarian keeps this overhead away from his teammates, so that they are not distracted (and delayed) by work that could just as easily be done by someone else. This ensures that progress is not brought to a halt in the name of budget and support issues, for example.

Von Mayrhauser suggests that this model lends itself most naturally to hierarchically structured projects that are not overbearing in scope [VONA84].

3) Surgical Teams

Brooks expanded the Chief Programmer Team concept into a model he called the Surgical Team. This metaphor intended to bring to mind the skilled team on which a Surgeon relies, each of which plays a crucial role in the success of the operation. Although ultimately the Surgeon is responsible for the main focus of the work, it is not plausible to expect the Surgeon to be responsible for single-handedly seeing to every detail of the work. With this in mind, Brooks defines the roles as follows:

3a. The Surgeon

The Surgeon is the most gifted programmer on the team. He formulates an attack, plans the solution, and implements the majority of the code, including the more complex sections. He relies heavily on the other members of his team, primarily to prevent his focus from drifting from the current problems and their solutions to activities peripheral to his central purpose.

3b. The Copilot

Although the term “copilot” doesn’t quite fit the medical metaphor, it is quite apt. The Copilot sees to everything that the Surgeon sees to, without getting in the Surgeon’s way. He is a second-in-command, able to take the controls at a moment’s notice, or to tackle whatever tasks the Surgeon needs from moment to moment. He is also a listening ear for the Surgeon to use to help develop or hash out strategies and designs.

3c. The Administrator

This member ensures that resources are available as needed, including (but not limited to) funding, sufficient computer resources (hardware and software),

staffing, and physical space. His responsibilities include financial and legal matters.

3d. The Editor

The Editor is a gentle backup for the Surgeon with regard to documentation. While the Surgeon must be responsible for accurate documentation (as nobody on the team will understand the system as well as he), the Surgeon is not always the most lucid and eloquent member of the team with regard to communication skills. The Editor is responsible for making sure that the Surgeon's documentation is readable by others, and that it remains accurate throughout development and maintenance.

3e. The Program Clerk

This person maintains the archival and versioning of all code and tests. The Clerk must be able to produce, at a moment's notice, any needed portion of the project, past or current, from design through testing. This degree of organization requires substantial overhead, and so this role's separate existence provides a much-needed aid to productivity for the other team members.

3f. The Toolsmith

This role may seem outdated with the modern existence of myriad IDEs, CASE tools, and other resources, but their very existence actually confirms the need for tools to relieve the programmer of various burdens. Generically produced tools are certainly useful, but cannot predict the more specific needs of the project team. As a Surgeon trying to test an idea thinks to himself, "I could fix this entire problem easily if I could…," the Toolsmith is responsible for providing those micro-solutions.

3g. The Tester

This member is responsible for writing and checking all test cases according to system design and specification, comparing output with expected results, and writing stubs and/or drivers as warranted by the situation.

3h. The Language Lawyer

Brooks writes that "most computer installations have one or two people who delight in mastery of the intricacies of a programming language" [BROF95]. As the Toolsmith provides original external aids for development and testing, the Language Lawyer points out built-in solutions for those cases where the Surgeon's musings are more along the lines of "I wish the language would let me do…."

3i. The Secretaries

For clerical work external and internal to the code and/or product, the Administrator and the Editor, respectively, have a Secretary.

Some of Brooks' listed roles may be shared between teams. Brooks suggests that the Administrator and the Language Lawyer be shared if the team situation allows. [BROF95]

Although Brooks' focus was on functional roles rather than anything with a specific psychological aspect, it is worth considering the team implications from this standpoint. If roles working closely together, such as the Surgeon and Copilot, have trouble working together because of personality conflicts, this will certainly affect the team as a whole. Brooks also intended to set a bound on effective team size.

4) Egoless Teams

Egoless Programming is somewhat of a periphery where programming team models are concerned. Advocates of the Egoless Team are quick to point out that a special type or set of people is needed to make this team work [WEIG71]. Schneiderman asserts that the Egoless Team is a temporary phase that some fortunate and healthy teams go though, but one that is very hard to sustain [SCHB80].

The basic concept is that no member of the team is specifically and inherently tied to any specific part of the project: everyone is responsible for everything. This separation of person and product allows programmers to collaborate without the negative aspects commonly associated with comparing a stronger teammate with a weaker one.

In this model, no specific roles exist, but team members are certain to still incline to those tasks that take better advantage of their individual strengths and gifts. (The advice of Belbin, Myers, etc. regarding the need for diversity and variety becomes apparent here, even when considered in the context of an egoless environment.)

Von Mayrhauser, however, argues that Egoless Teams may be appropriate in cases where functional roles are well defined (stating auto production as an example), but that programming is not such a clear cut set of rules and processes [VONA84]. She continues to assert that although the model may promote morale, it does not improve the product. Essentially, Egoless Programming has great potential to become chaotic if the team members are not well managed and dedicated to the concept and the team itself.

5) Extreme Programming

The concepts of Agile Methodologies in general [AGIL01], and Extreme Programming in particular [BECK99] are of particular relevance here. When Beck described the formalized process of Extreme Programming, the Chief Programmer model was once again brought to use. XP requires that two programmers work together at a single terminal [BECK99]. As one programmer enters code, the second programmer simply observes, points out flaws, and suggests options. In this sense, “two heads are better than one.” Auer reiterates the point that while XP has proven invaluable for some types of projects, it is still not the long sought after silver bullet [AUEK01].

Extreme Programming also adopts the better aspects of Egoless Programming. One of the rules of XP is that when any member of the team finds an opportunity to improve any code, he is responsible for making those improvements, even if he did not write the code. This policy is tempered with a requirement for frequent builds, with all test cases rerun after each new build. When a change is made, a rebuild is done. If all previous test cases still function, the change is kept; otherwise, it is thrown out, and the previous version of the code remains current [BECK99].

In spite of the subtle differences among these four types of teams, there are some very strong principles that cannot easily be refuted:

- a small number of main programmers make the team (and the project) easier to manage;
- two programmers working together is better than larger numbers, or a single programmer;
- there exists a substantial amount of project overhead from which the two programmers should be protected;
- despite the gross differences in the roles, each role is crucial to the team's success.

The important questions to ask at this point are those regarding the necessary team roles for a software development team, and the matching of the functional roles described by the various team models with appropriate effective team roles. Which personality types work well together? Which are more apt to readily satisfy which functional roles? Do certain combinations work better than others?

Before these questions can be answered, a battery of tests and observations are necessary to determine what effects various personality types have on generic software engineering teams. This is one of the primary goals of this research. In order for any results to be useful in generating a management scheme of any value, the measurement tool must be predictable. The reliability of the any

personality measurement devices must be checked while research is being conducted to determine the value of the tools, and if better ways exist to use the tools.

The eventual hope of this line of research is that team model structures reflect the nature of the team members themselves. If the needed roles which computer scientists play on development teams become a more natural and comfortable extension of their innate personality, the team should expect to run much more smoothly, efficiently, and with greater motivation. The eventual payoff would be in quality and productivity levels. Existing tools have substantially raised the bar with respect to conceivable project scope over the years. How much potential could exist in this new area of management?

Chapter II: Belbin's Team Roles Theory

A) Background

A team-based approach to software engineering requires that analysis from the perspectives of personality, psychology, or management also be team-oriented. Meredith Belbin's team roles are a practical choice for such a perspective. Belbin's work stemmed from a fixation on the basic question, "Why do some management teams succeed and others fail?" [BELR81] Stevens identified the relevance of this line of thought to the software engineering domain [STET98], but his observation is not unique. At a recent workshop at the ACM-OOPSLA conference, a lead software engineer noted that he had worked with very successful and productive teams, but complained that he currently has a team problem which he describes as "two backseat drivers," both correcting each other's problems, but neither willing to take the lead [BRUT01]. Software processes author and consultant Joseph Perline notes, "I've had those who I've reached a sort of 'Vulcan Mind Meld' with when pair programming, and then I've had people with whom I simply cannot pair program. There are also those who *will not* pair program – 'cowboys' if you will" [PELJ01].

Meredith Belbin began his nine years of research in 1969 at the Administrative Staff College at Henley, England [DULV95] [BELR81]. With a focus on management science, his early work included the development of a system of tools (computer-aided simulations and games) that objectively measured management teams' success. With these tools in place, one could then conduct studies to identify the attributes of an optimal team. Belbin's interest turned soon to the perceived types of people who comprise successful and unsuccessful teams, the combinations of people on successful teams, and the contributions of various team members.

With the new focus on team member types, Belbin developed a scheme for identifying the types. This work made use of several psychometric tools, most notably the Cattell Personality Inventory, or "16PF", the Critical Thinking Appraisal, the Occupational Personality Questionnaire (OPQ), and the Personal Preference Questionnaire (PPQ). The most notable of these, the 16PF test, measures a person's ranking on each of sixteen scales. The scales each include a pair of polar opposites, such as "shy/bold", or "relaxed/tense." These scales are then combined through various prescribed formulae (which Cattell provided) to produce "second-order" factors, including "introversion/extroversion" and "creative disposition." Other tests used in Belbin's research included the Critical Thinking Appraisal (CTA) [WATG80] and the Personal Preference Questionnaire (PPQ), developed by the Industrial Training Research Unit at Cambridge University [IND], and Hans Eysenck's Eysenck Personality Inventory (EPI) [EYSH64] [BELR81].

In Belbin's early experiments, a phenomenon was observed, which was subsequently termed the *Apollo Syndrome*. The Apollo Syndrome is a trend whereby extremely gifted subjects grouped into a "super-team" under-perform teams comprised of more average subjects. Belbin defines the *Apollo Company* as "one which hoards mental ability so that it has ample resources and talents for dealing with the most complex problems, but suffers from extreme difficulty in being able to use what it has," [BELR81]. (This, incidentally, shadows the communication and direction problems experienced by many technical teams.) In the Apollo team experiments, team members often inadvertently undermined each other's strategies while pursuing their own solutions to the perceived team problems. It became clear that intelligence, talent, and understanding were not the only factors underlying a team's ability (even probability) to be effective and successful.

Using group observation, tools to measure team success, and psychometric measurement techniques, Belbin gradually defined eight team roles, which he

claimed could accurately describe the needs and abilities of a team and its members. Belbin explains that a full complement of roles is the best chance for a team's success. Furthermore, certain types of situations and problems are best addressed by particular roles. A summary description of the roles (taken from Belbin's 1981 book) appears in Table 2.1 for convenience.

B) Team Roles and Role Theory

1) The Chairman:

The Chairman is the first of Belbin's leadership roles. Although he may not be as technically minded as other roles, he is the positive motivation for the team. He is generally in a more permanent leadership position, such as a vice president, for example.

2) The Shaper:

The Shaper is the other leadership role Belbin describes. He is often a more abrasive motivator, but gets the job done. He is generally better suited to temporary positions, such as project team leadership, which ends when the project is completed. Research suggests that software development professionals tend more toward this leadership style than that of the Chairman [STET98] [HENS99].

3) The Plant:

The Plant is the source of creative and original ideas for the team. He is known for a tendency to ignore the obvious and embrace the less conventional.

4) The Monitor-Evaluator:

The Monitor-Evaluator is a fairly detached role. He keeps the team from making conceptual mistakes, and points out flaws in the plan before they become commitments.

5) The Resource Investigator:

In the same sense that the Plant provides ideas and resources from inside the team, the Resource Investigator provides external resources to the team. He is known for the contacts he has, and for an extroverted nature.

6) The Company Worker:

This role is often called the Implementer. (In fact, Belbin has now renamed him as such.) It is this member who turns concepts into working procedures. This member also has a tendency to seek out and find the unfilled needs of the company, and tackle those projects that others would rather avoid.

7) The Team Worker:

This team member provides a type of lubrication for the team. He resolves conflicts, promotes spirit, and serves as a sort of peacemaker. If any one member is responsible for a team's sense of cohesiveness, it is he.

8) The Completer-Finisher:

The person in this role sees to it that the plans that are begun are carried through to fruition. This role is known primarily for attention to detail.

Role	Typical Features	Positive Qualities	Allowable Weakness
CH	Calm, self-confident, controlled.	A capacity for treating and welcoming all potential contributors on their merits and without prejudice. A strong sense of objectives.	No more than ordinary in terms of intellect or creative ability.
SH	Highly-strung, outgoing, dynamic.	Drive and readiness to challenge inertia, ineffectiveness, complacency or self-deception.	Proneness to provocation, irritation and impatience.
PL	Individualistic, serious-minded, unorthodox.	Genius, imagination, intellect, knowledge.	Up in the clouds, inclined to disregard practical details or protocol.
ME	Sober, unemotional, prudent.	Judgment discretion, hard-headedness.	Lacks inspiration or the ability to motivate others.
RI	Extroverted, enthusiastic, curious, communicative.	A capacity for contacting people and exploring anything new. An ability to respond to challenge.	Liable to lose interest once the initial fascination has passed.
CW	Conservative, dutiful, predictable.	Organizing ability, practical common sense, hard working, self-discipline.	Lack of flexibility, unresponsiveness to unproven ideas.
TW	Socially oriented, rather mild, sensitive.	An ability to respond to people and to situations, and to promote team spirit.	Indecisiveness at moments of crisis.
CF	Painstaking, orderly, conscientious, anxious.	A capacity for follow-through. Perfectionism.	A tendency to worry about small things. A reluctance to "let go".

Table 2.1: Belbin Team Role Summary, CH=Chairman, SH=Shaper, PL=Plant, ME=Monitor-Evaluator, RI=Resource Investigator, CW=Company Worker, TW=Team Worker, and CF=Completer-Finisher. [BELR81]

The roles can be divided into pairs bearing some similarity. The Chairman and Shaper roles, for example, both focus on leadership attributes, although their leadership styles and purposes are very different. The Plant and Monitor-Evaluator both supply the intellect of the team, although the Plant's contributions are generally creative (not bound by the limitations of conventionality), while the Monitor-Evaluator's contributions tend to be scrutinous, logical assessments of what will or will not work, according to cold facts [BELR81].

For the most part, Belbin's roles have been applied only to business management teams. Belbin's research observations make note of the fact that the management of various industries often has drastically different pools of roles present, with the culture behind the industry rewarding, if not dictating, the behaviors associated with specific roles. Despite Belbin's attempt to cover a broad range of industries, he limited his research to the management groups themselves, without attempting a foray into teams of other disciplines. Yourdon suggested the application of these roles to software engineering teams, and the

work of Stevens and Henry began to explore the validity of such an association [YOUE93] [HENS99].

The theory behind the roles makes various generalizations regarding the strengths and weaknesses of each role and the common stereotypical interactions between roles. These facets are of particular interest to software engineering. If the expected interactions between management team roles can be applied reliably to software engineering teams, then management can use this understanding to fix team shortcomings, maximize individual effectiveness, and prevent probable sources of antagonism within the team.

A person's contribution to a team is expressed as a primary role and secondary role. The primary role is the behavior to which a person defaults in normal circumstances. A person may switch to his secondary role (which is sort of an auxiliary) in cases for example, where the team has a special need for a role which does not match any one member's primary role, or two members both default to a single role in such a way that conflict arises. This second case is perhaps best illustrated in the context of the leadership roles. Two primary Shapers are likely to compete for the role, a competition that may introduce serious problems if they have different agendas.

After defining the team roles, Belbin toyed with various conceptual combinations, including "pure" teams (teams comprised entirely of a single primary role), effective combinations, and combinations likely to cause problems in team dynamic. One example of a problematic combination is a Monitor-Evaluator with Team Workers and Company Workers but no Plant. The Company Workers move the company along, while the Team Workers ease the team dynamic, preventing conflicts between the Company Workers. This results in a very smooth-running team, albeit without the innovations that could be provided by a Plant or Resource Investigator. In this case, the Monitor-Evaluator is left with no ideas to critique, no problems to fix, and no contribution to make.

Conversely, if the Monitor-Evaluator is paired with a Completer-Finisher, and the two are given some degree of prominence, the lack of the innovation provided by a Plant or Resource Investigator, or leadership of a Shaper will predispose the pair to situations where real progress in inhibited while the team focuses on small details. Other problematic combinations include multiple strong Shapers, especially if combined with a weak Chairman.

Belbin recommends some interesting combinations, claiming that the roles complement each other well for certain situations. Team Workers and Resource Investigators seem to fare well together if they possess above average intelligence, with the Resource Investigators providing ideas (from the outside), Team Workers ensuring the smooth operation of the team, and both providing secondary Company Worker roles. The addition of a Shaper then prevents the complacency that can occur with the above roles. While this combination seems appropriate for management, it seems ill applied to software development, because of the apparent relative lack of Resource Investigators in the field and the lack of focus on development and scrutiny of original designs.

Belbin describes various Plant combinations that seem more suited to computer science: pairing a Plant with a Chairman, a Plant and a Resource Investigator, or a Plant with a Monitor-Evaluator. The creativity and intellect underlying the Plant are more relevant to software design and development. A Chairman has the potential to add direction without abrasion. The Resource Investigator brings in outside ideas while the Plant provides inside innovations, and the Monitor-Evaluator effectively critiques and points out problem issues within the Plant's formulated models. Choosing the appropriate pairing strategy listed above may help problems experienced by a specific Plant in a specific situation. Belbin also explains the Plant's unsuitability for team leadership and need for a partner, saying, "A brilliantly clever and creative Plant is an asset to a company, but only if ultimate responsibility lies with another," [BELR81].

C) Team Role Measurement: The Belbin SPI

To determine a person's role, Belbin produced a measurement device, known as the Interplace System, which is comprised of two tools: the Self Perception Inventory (SPI) and the Observer's Assessment (OA). The SPI first appeared in Belbin's 1981 book *Management Teams*, as a tool with which a person could discover the team role for which they were best suited. The OA is used to obtain a peer-based estimation of a person's role, and is generally completed by a close colleague of the person being tested. The OA does not appear in *Management Teams* and is available only through Belbin Associates. The two Interplace tools have been refined over the years, and are now intended to be parallel metrics, used in conjunction to verify each other. Belbin does include the SPI, along with a set of directions and a detailed mechanism for scoring the test, in *Management Teams*. This Interplace System has undergone revision over the years, although the current SPI differs from the original by less than 50%. This tool has seen a fair amount of controversy, enjoying widespread use among management, but publicly evaluated, criticized, and defended as a valid psychometric tool by psychology and management professionals.

D) The SPI: Criticisms

Furnham, Steele, and Pendleton published the first major public criticism of the SPI, demonstrating that the tool lacked internal reliability and consistency [FURA93a]. This testing used both original and updated forms of the SPI. Empirical tests suggested the existence of various factors distinguishing the roles, but did not confirm Belbin's theory behind the roles. Belbin himself countered that the tool was never meant to be a standalone means of assessing a person's team role, and that the system used by his company included a battery of tools, of which the SPI was only a part. Furnham *et al.* then countered that even if the SPI was not intended for definitive use, it was the SPI that was being used by managers all over the world (especially in Europe, where Belbin's book had gained wider appeal) [HOGC90]. Furnham argued that because the SPI was the

tool being employed most commonly, the SPI should be scrutinized for its potential value as a psychometric tool.

One of the complaints voiced in this research concerns the ipsative nature of the test [FURA93a], and Johnson et al. published a manifesto enumerating the inadequacies associated with ipsative tests in general [JOHC88]. (*Ipsative* denotes the quality of a test in which a change in one measured dimension affects another dimension.) In the case of the SPI, the test is interpreted by calculating a score for each of eight possible team role categories. The test itself is comprised of seven general questions, each with eight possible answers. For each question, the subject must distribute exactly ten points among the eight possible answers, such that an answer which more aptly describes the subject's own character receives more points, and an answer which does not accurately describe the subject receives fewer (or no) points. The result of this scheme is that in order for a point to be added to one category (the Plant category, for example) that point must be taken from one of the other seven categories, to ensure that the ten-point total is preserved.

In an ipsative test such as the SPI, the measured categories are prevented from being independent. A person who tests with a large number of points in the Shaper category, for example, is unlikely to also have a large number of points in a second category, because the abundance of Shaper points necessitates a deficit in the number of points which can be spread over the remaining categories.

This introduces yet another confounding side effect into the interpretation of SPI scores: the scores of two people cannot be directly compared against one another. One person who is naturally extremely gifted in *two* categories (e.g., Shaper and Company Worker) will have to split points between those two categories, whereas another person whose primary strength is in only *one* category (e.g. Company Worker) can devote a large number of points to that category. In such a case, even though the first person may naturally be a stronger

Company Worker, the second person is likely to test with a higher score in that category on the SPI. The SPI, as it appears in Belbin's *Management Teams*, is included in Appendix A.

Furnham et al. developed a normative (non-ipsative) form of the revised SPI, in which there are ten possibilities for each of seven categories [FURA93a]. The normative version of the test was identical to the original SPI with one exception: instead of distributing ten points among the options for each category, subjects rated each option independently from one to nine, through a Likert scale [LIKR32]. In this environment, a subject's rating for one option has no effect on his possible rating for any other option. Results using the normative version fared no better or worse than results of the ipsative SPI.

Subsequent studies criticized the construct validity of the SPI and its Interplace counterpart, the Observer's Assessment, described above. Broucek and Randall demonstrated poor construct equivalency between the SPI, the OA, and the Five Factor Model of personality measure, claiming that the SPI and OA cannot be considered parallel measures of a person's team role. Broucek *et al.* also detail the mathematical difficulties relating to analysis of ipsative tests [BROW96]. One empirical study defended the SPI, confirming the construct equivalency of the SPI and the OA [DULV95].

E) The SPI: A Defense

Belbin's critics may suggest to some that study using this tool is unwarranted, but this is far from the case. Several valid reasons exist for continuing research with this tool.

First, the Belbin SPI is free, and therefore accessible to any manager who wishes to try it. This instrument is included in Belbin's book [BELR81], which also describes how to interpret the test. Secondly (and in part, for the reason

above), it has been widely accepted and used in industry management [FURA93b]. In other words, the industry already uses the tool, so it certainly makes sense to study how to better apply it in the specific area of software engineering teams.

Thirdly, some significant results have been obtained in previous studies using the tool. Criticisms aside, if the tool can be shown to be effective and useful, it behooves managers to make use of it. Theory must be driven partially by results, and if results exist, they deserve a second look.

Fourth, although Belbin mentions that he did not intend for the SPI to be used as a primary tool, he continues to make use of it. (Note: Belbin claims that the test was written as an incentive within his popular book, as an informal guide, which can suggest to a reader what his role might be [BELR93].) A somewhat revised version is the primary part of a set of online tests, which are available on his company's website, for a price of twenty-five British Pounds Sterling, as of this writing [BELA00].

With regard to the criticisms of the test's ipsativity, some studies have defended ipsative tests in general, claiming that ipsative tests and normative tests may score equally well [SAVP91]. Finally, some of the test's biggest critics, Furnham et al., admit that their own critique is "far from damning," and suggest that more research is warranted [FURA93b]. Furthermore, Dulewicz offers empirical evidence supporting the reliability and validity [DULV95]. Yourdon also suggests the tie between Belbin's roles to software engineering, although he attributes the roles to Thomsett [THOR90] [YOUE93].

If any special directions can be found which can aid in making the test more reliable, those directions should be documented for the sake of future accuracy, along with a rationale that can then guide management decisions in how to administer, interpret, and subsequently use the test.

F) Relevance For Computer Science

1) Functional Roles vs. Team Roles

Belbin advocates the matching of *functional roles* and *team roles*. A functional role embodies the literal work, physical actions, intellectual activity, and specific goals related to a person's job. The team role describes the nature of the contributions a person makes to the group effort, which are often unnoticed and generally not directly tied to job requirements. The team models of Mills, Baker, Brooks, Weinberg, and Beck, described earlier contain examples of functional roles. If the nature of the jobs associated with these functional roles can be matched with a personality predisposed to a matching team role, the corresponding work is expected to flow much more easily and successfully.

For example, both Mills/Baker and Brooks advocate the pairing of a single, skilled, gifted, most important programmer with a secondary, skilled, gifted assistant. The people who enter these roles must be willing and able to communicate easily on the same level, without conflict or personal agendas entering the equation. Combinations of Belbin roles known to have problems in this respect may lead to team difficulties. This difficulty may be circumvented by the avoidance of certain roles (or combinations) or the inclusion of certain roles (or combinations.) In this case, dual Shapers would probably be disastrous, as the Shaper has a tendency to be abrasive and unyielding [STET98].

Beck continues the pair-programming concept, describing the nature not only of a single coder, but of the partner with whom he works, as well. His explanation and justification for the pair-programmer requirement of Extreme Programming neatly parallel the description of the Plant and Monitor-Evaluator team roles, suggesting their possible special utility for this team model.

> There are two roles in each pair. One partner, the one with the keyboard and the mouse, is thinking about the best way to implement this method right here. The other partner is thinking more strategically:
>
> - Is this whole approach going to work?
> - What are some other test cases that might not work yet?
> - Is there some way to simplify the whole system so the current problem just disappears? [BECK99]

Compare this with excerpts from Belbin's description of the Plant and the Monitor-Evaluator:

> [The Plant]…specifies advancing new ideas and strategies with special attention to major issues… looking for possible breaks in approach to the problems with which the group is confronted.

> Intellectually, the [Monitor-Evaluator] was the only person who could hold his own in debate with a [Plant]; and could cause the [Plant] to change his standpoint and retain his respect in so doing… His real asset was a capacity for making shrewd judgments that took all factors into account. [BELR81]

Note that this does not suggest that others cannot effectively fill the pair programming functions. The Extreme Programming movement has already enjoyed a notable success without the help of Belbin's Roles; however, what is suggested here is that the Plant and Monitor-Evaluator may be more comfortable in these functional roles, and may interact more smoothly, easily, and effectively than might others, and in those special cases where other roles get stuck, a Plant and Monitor-Evaluator may be the solution to the problem. In cases where two people have trouble getting along, if one member of the pair is a Plant, replacing his partner with a Monitor-Evaluator may put life back into the project. Belbin also hints that having one Plant aid another Plant experiencing problems is unlikely to solve the problem.

With regard to Egoless Teams, it would behoove managers to avoid mistakes whereby a team comprised of Belbin roles which are inherently poor combinations are asked to "make it work" for an Egoless Team. Conflicts and individual differences are certain to come into play on a team, regardless of efforts to help a team function in an egoless sense. Teams with all Company Workers or multiple Shapers, for example, are bound to perform badly. Team Workers may aid a team like this for the sake of morale, but introverted roles may

be unable to easily cope with the open-critique concepts associated with this model.

2) Recent Work Within Computer Science: Shaper, Plant, and Monitor-Evaluator

A study completed at Virginia Tech in 1998 took the first steps toward exploration of the feasibility of applying Belbin's roles to software engineering teams. This work, conducted by Stevens and Henry, sought to understand the value of three specific roles on a team. The study ran empirical experiments using advanced student programmer teams, which were balanced with regard to experience, grade point average, and presence of various Belbin roles, with the exception of a single role, which was used as a control variable. Teams were observed designing and implementing a solution to a given problem. Team success was then measured by time to completion and a survey regarding the "viability" of the team, or the willingness of the team members to work together on future programming endeavors.

The three roles chosen for study were the Shaper, the Plant, and the Monitor-Evaluator. These roles are of specific interest to software engineering teams because of the implications of their characteristics. Software engineering as a discipline requires that creative, working solutions be applied to understood problems. This requires three elements: leadership and direction for the team; intelligence and creativity by which ideas can be generated to solve more unorthodox problems; and a source of rational decisions for points where the team gets stuck or can choose between many options. These qualities embody the Shaper, Plant, and Monitor-Evaluator, respectively.

The first of Stevens' experiments focused on team leadership. Belbin defined two leadership roles, the Chairman and the Shaper. Shapers seem to be more prevalent in software development than are Chairmen. Stevens' work

showed that teams with a single identifiable leader (Shaper) perform better than other types of teams (those with multiple leaders or no leader). Figure 2.1 demonstrates this trend, showing that in controlled experiments, teams with a single identifiable leader (Shaper) completed problems in less time than did leaderless teams. In this study, a team with two Shaper roles was described as having "no leader", because a decisive single leader could not be identified. The observation described in this figure matches Belbin's theories of leadership combinations. Belbin observed that teams comprised entirely of Shapers fared poorly, and sometimes even turned to unethical means to remain competitive in order to save embarrassment and vent frustration. Teams without any leadership role tended, somewhat obviously, to lack drive and direction.

Broucek and Randall claimed that the Shaper role theoretically should be consistent with a high degree of neuroticism [BROW96], and Belbin teaches that the Shaper has a tendency to "lead from the front" (a concept consistent with the fast-paced nature of project leadership). However, Belbin also observed that Shapers tend to perform well when the Chairman role is thrust upon them. Another noteworthy point is that adding a Shaper may aid a team that has become lethargic and lacking drive, although the Shaper may upset a team which is already productive, balanced, and running smoothly.

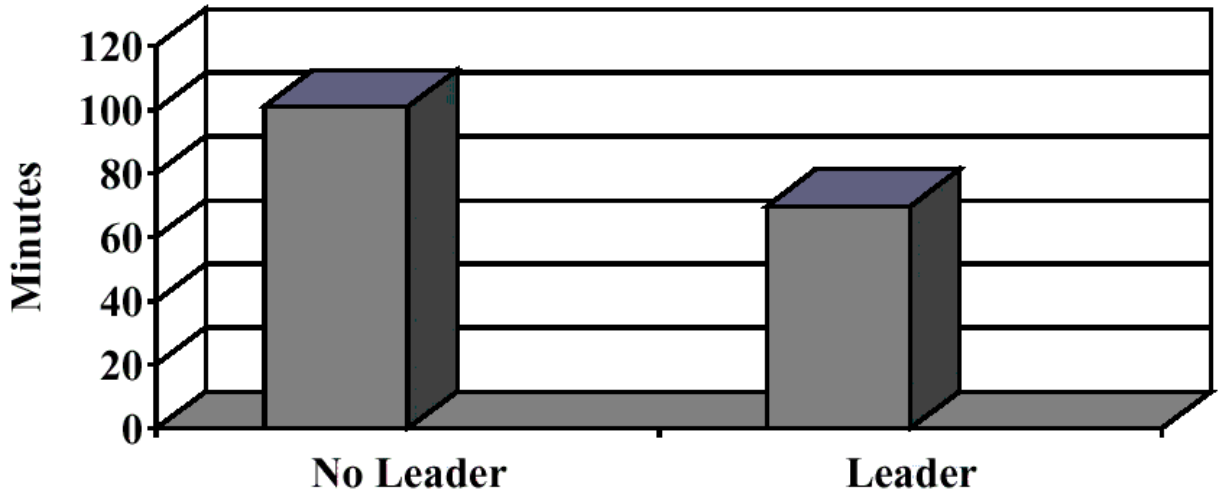


Figure 2.1: The effect of the Shaper on team completion time [STET98]

The Plant was the second focus of Stevens' study [STET98] [HENS99]. This study concluded that the number of Plants on a team and the success of the team are not independent. Stevens observed that teams comprised entirely of Plants performed better than teams with some Plants or with no Plants (Figure 2.2), but this interpretation may be suspect, because of improper interpretation of the Belbin scores associated with those team members, and the fact that the results do not conform to role theory as described by Belbin himself. Stevens' observation of teams comprised entirely of Plants outperforming all other combinations is at ends with Belbin's observation that multiple Plants without clear leadership produce poor results [BELR81]. However, it must be conceded that this could simply reflect a difference between the natures of software development teams and management teams. Stevens also conjectured that the pure Plant teams may "operate on the same wavelength".

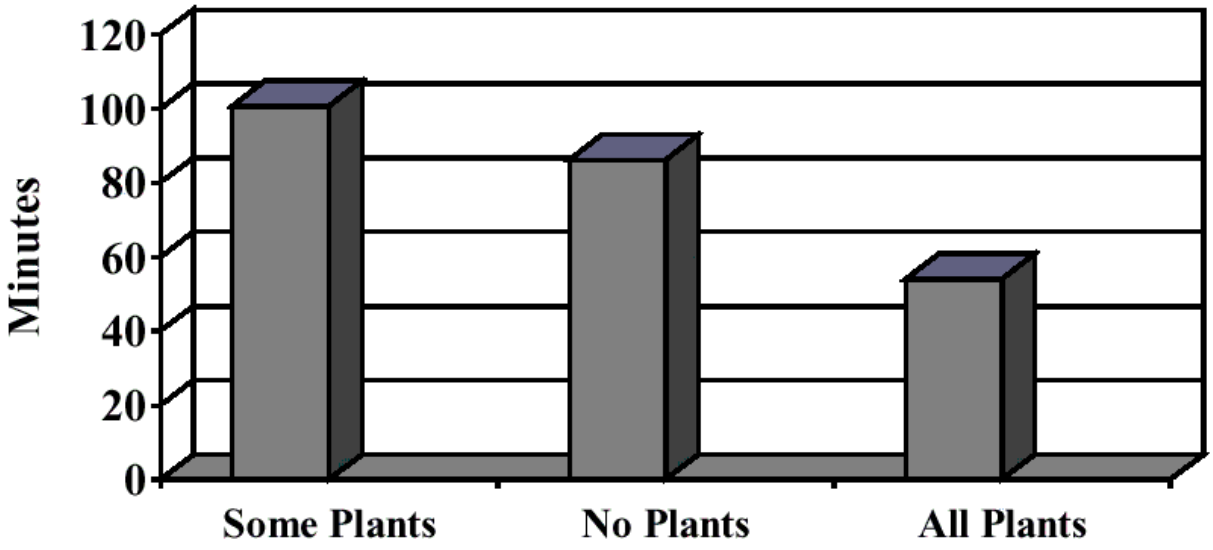


Figure 2.2: The effect of the Plant on team completion time [STET98]

Finally, Steven's investigation into the contributions of Monitor-Evaluators produced unexpected results. Monitor-Evaluator was dubbed the "decision maker" role [STET98], and the assumption made that a software

development team required decision-making capability, which would be provided by the Monitor-Evaluator. (This premise is now considered faulty and too simplistic, for reasons that are enumerated in Chapter 6, under ***Ascertainment of Previous Study***.) Stevens concluded that, within the bounds of his sample, the impact of a Monitor-Evaluator on a software engineering team could not be statistically proven to a significant degree, and conceded that the Monitor-Evaluator may not be beneficial or necessary for software engineering.

G) The Next Step: The Company Worker

A logical next step taken by this current work is to investigate the effects of the presence of Company Workers on development teams. The decision to study the Company Worker was based on the role theory concepts describing the Company Worker as the "implementer" of a team. (In fact, Belbin's more recent publications rename the Company Worker as the "Implementer" [BELA00]. Certainly, a software development group requires a source of leadership and ideas, and perhaps a source of decision-making. Team role theory suggests the need for an implementer. Does this facet of team role theory (originally applied to management science) apply in the same way to software engineering teams?

After Belbin investigated and identified the roles reflecting the most obvious needs of a successful team (leadership and sources of innovation and creativity), the Company Worker became the first role to completely emerge in his studies without having been explicitly sought. It became apparent to Belbin that a particular type of team member was generally responsible for actually getting the work of the team done. Whereas the Chairman provided a destination and morale, the Plant offered a workable route, and the Shaper provided fuel and fire, it was the Company Worker who ended up actually moving the team toward the destination.

Another interesting facet of the Company Worker is his desire to seek out those aspects of a job that must be completed, and that nobody else on the team aspires to do. Belbin gives the strong example of the necessary firing of a liked employee who simply inhibits the unit's progress [BELR81]. No manager with any shred of compassion would look forward to the task of terminating a colleague, but it is the Company Worker who not only realizes that the pruning is necessary for the good of the plant, but who also values the company's well-being and goals strongly enough to make its priorities more important than his own. (This does not in any way suggest that the Company Worker is either a workaholic or a fanatic, but rather that his loyalty is to the good of the whole, rather than to the good of the individuals.)

That loyalty to the good of the outfit is of particular interest to computer science. Industry's focus on quality reflects a deeper need, a need for people who *care* about the real goals, needs, and values of the company. Often, the software process includes a lot of extra time-consuming activities and process requirements that end up doing more harm than good [BECK99]. If role theory applies similarly to management teams and software teams, the Company Worker's conscience should prevent poor quality products or practices from escaping his attention. He is likely to trim programs and projects that are inherently counterproductive, but instead should be expected to focus energies on programs that either produce strong results or have the potential to yield value, *if* he offers software engineering the same gifts that he offers management teams.

This notion (carried a step further) may suggest that the software engineering Company Worker focuses on tasks from which others developers shy. It has been proffered that the psychological implications behind testing one's own code are counter to an individual's own goals. A successful test is described as one in which an error is discovered [MEYG79]. If a programmer first writes a module, and then writes a test case intended to break it, he suddenly is in a no-win situation: if he has a successful test, he proves himself to be an unsuccessful

programmer, and vice-versa. In some cases, programmers shy from the issue by doing a half-hearted or weak job of testing their code – again, resulting in poorer quality. Seemingly, the Company Worker role as described by Belbin would zoom in on the company's need for quality testing, and prioritize this over his own self-gratification (e.g. writing code which is unbroken in testing).

Finally, both Stevens' (and the current work's) experiment data suggests that the Company Worker role is quite prevalent in computer science. According to previous research, a role's absence in the software engineering field suggests that it is unnecessary to the domain. If the converse is true (the presence of a role suggests the need for that role), then the Company worker should be shown to have a positive effect on a development team's effectiveness.

The relationship between the Company Worker and team success in the software engineering domain should be studied and analyzed. What does the Company Worker contribute to computer science? For what tasks does he prefer to contribute? Which of his qualities affect the team's progress and effectiveness? These qualities should be identified, tested, and recorded to further the growing body of knowledge in this area.

Chapter III: Experimental Design

A) Investigation Summary

This goal of this research is to examine five? main aspects of the Company Worker and other team roles, and the Self-Perception Inventory itself, as they relate to software engineering teams. To further analyze results, this entails an examination of the Self-Perception Inventory and a sample of student computer scientists from the perspective of a more widely recognized and accepted tool, the Keirsey Temperament Sorter.

Goals of this research are as follows:

1. Observe and define the effect of Company Workers on a software engineering team success, effectiveness, and viability;
2. Determine if certain team roles are drawn to particular stages of the software engineering process, or to particular team needs, such as idea generation;
3. Describe the study sample of computer science students using the Keirsey Temperament Sorter. Look for trends and correlations between results of the Keirsey Temperament Sorter and the Belbin Self-Perception Inventory;
4. Retest sample participants with the Belbin Self-Perception Inventory in the context of experiment teams. Use retest results to explore whether new roles emerge because of team need. Also use these results to analyze the test-retest validity of the SPI.

Eventual results supported the following eventual conclusions:

1. A software engineer's personality has an effect on his team's success;
2. There is a difference in the nature of Belbin's team roles as applied to software engineering teams, as opposed to management teams;
3. Whereas Belbin's Company Worker is useful in a management team setting, the role makes a software engineering team less predictable;
4. In addition to adding unpredictability to a team, the Company Worker role in software engineering is also unpredictable, in that it can not be distinguished from other software engineers with regard to the Keirsey Temperament Sorter;

5. Various roles seem to be drawn to particular stages of the software engineering process;
6. The Belbin Self-Perception Inventory, as published in his 1981 book, is useful, but has not been validated, and should be taken with a grain of salt;
7. The test-retest validity of the Self-Perception Inventory suggests that either the test is problematic or a different testing method is warranted. The test may have to be taken by subjects in pre-existing long-term teams, for example;
8. A person's team role may change over time, or from team to team.
9. Trends between the Belbin Self Perception Inventory and the widely accepted Keirsey Temperament Sorter suggest that the Self Perception Inventory is measuring *something*.

B) Process

To investigate the effect of the presence of the Company Worker on the effectiveness of software engineering teams, several experiments were simultaneously conducted. Although previous work on the Plant, Shaper, and Monitor-Evaluator outlined a process for studying the value contributed by a role [STET98], it was decided that more detailed results than such an experiment generated were desirable. Thus, the investigation sought not only to determine the value of the Company Worker, but also to question the specific arena in which his contributions exist. Secondly, a measure of the viability of sustaining a team, with respect to the presence of Company Workers, was performed. In other words, are teams with varying numbers of Company Workers more or less likely to be willing to work together?

Thirdly, an examination of the contributions of each role with respect to the most basic elements of the waterfall model of the software engineering process was done. Specifically, three phases of general software engineering (design, implementation, and testing/debugging) were considered with respect to the Company Worker. The goal was to answer questions akin to, "How much does the Company Worker contribute to the design phase?"

A fourth area of interest involved the reliability and stability of a person's team role. Would a person's teammates be able to recognize his role? To what degree is a person's team role a constant? Does a person's team role change from team to team, or is one generally "born" to a certain role?

These questions are of particular value to managers assembling, organizing, and revising development teams. Of obvious interest is the value added to a team by a specific role, but just as relevant are questions about what kind of teams benefited most from the presence of which roles. Are Company Workers better suited to testing and debugging teams rather than to design teams, for example?

In some cases, the benefit added to a team by a management technique may not offset the cost of the technique itself. No manager wants to spend ten dollars to reap a five-dollar payoff, and if the viability of sustaining a team – the ease, comfort, and efficiency with which the members work together – provides a deficit when coupled with the productivity of the team, the team members should not be grouped together for sustained projects.

C) The Study Participants

The participant subjects for this study were the students of a senior-level software engineering course at Virginia Tech. Due to the unconventional nature of the course, a brief high-level overview of the class structure is warranted. The class was comprised of 27 computer science students. This course is a senior level elective, and has a reputation for attracting extremely gifted student programmers who have had previous internship or co-op experience.

The course is project-centered. For the major course project, students are grouped into design teams of three students each. Each design team proposes and designs a project with a reasonable degree of difficulty. This covers all phases of

design from architecture-level to detailed specification documentation. Designs are modularized to support some degree of parallel implementation. Other classmates are then contracted to produce code to realize the design specifications. No fewer than five programmers (each working on a separate module) are hired to implement each team's design, and no student may write production software for his own team's design.

D) The Value of the Company Worker

1) Team Formation

In order to study the basic value added by the Company Worker to a generic software engineering team, some of the method for this study was aligned with previous work relating Belbin Team Roles to software engineering. [STET98] To this end, each of 27 software engineering students was given the Belbin Self-Perception Inventory (SPI) and the Kiersey Temperament Sorter. Then, nine experimental teams were formulated with the following constraints:

1. Each experimental team is comprised of exactly three members;
2. The sum of the grade point averages for each team's three members is balanced, inasmuch as possible, across all teams for the experiment;
3. The *presence* of a Belbin role within a subject is determined by his/her score for that role on the Self-Perception Inventory exceeding a preset threshold;
4. The number of Company Workers on each team is varied. Some teams have two Company Workers, some have one, and others have no Company Workers;
5. The presence of all other roles on each team is balanced and made constant, inasmuch as that is possible;
6. No two students working together on a semester design project team may work together on an experimental team.

A set of experiments, dubbed "Phase I", were run (explained in the next section of this chapter), and then nine new teams were formulated for a "Phase II", using the same constraints listed above, plus a seventh, additional constraint:

7. No two students working together on a Phase I experimental team may work together on a Phase II experimental team.

Ironically, just before the study began, two student participants in the class became engaged to each other. In the interests of having this relationship not affect results, these classmates were prohibited from working on the same team in any of the experiments.

2) The Experiments

The Company Worker *value experiments* for both Phase I and Phase II followed the same format. Four 90-minute sessions were conducted. For each session, each team was given a single computer to work with. The computers and software provided to each team were identical. Each team was given a copy of a problem, in a format similar to that used by the annual International Programming Competition sponsored by the Association of Computing Machinery. (See Appendices E and F for example problems) Teams then were directed to design and implement a solution to the problem and submit it to an impartial judge who monitored the room. Any team questions were directed to the judge. Each team was allowed 90 minutes to complete the program.

Before being counted as a correct solution, each submitted program was exercised against a set of test data, which had been prepared in advance. The same test data was used for all teams. If the output test results were correct, the submission time for the team was recorded and the team was deemed "successful" for that experiment. For each of the two phases of the experiment, this process was conducted four times, each time with a different problem. For each run of the experiment, all teams worked on the same problem at the same time.

3) Measurement of Role by Threshold

As stated above in the Team Formation section (see the third team formation constraint), we chose to measure the presence of each team role by comparing a person's score for that role against a given threshold. Previous studies [STET98] set this threshold at 12 points. The current work decided to conform to that threshold level for theoretical reasons. If a subject could distribute available points evenly across all eight roles, each role would receive 8.5 points. For this reason, any score below ten is considered insignificant, as such a score is within the bounds of random chance. In fact, when scoring the SPI, scores between 10 and 19 are recorded using a single lower-case letter representing the measured role, followed by the measured score less ten. For example, a score of 12 in the Plant category would be recorded as "p2". Scores above 19 are recorded using a single upper-case letter representing the measured role, followed by the measured score less 20. Scores below 10 for a role are simply not recorded. Scores below 12 are close enough to the average to be deemed untrustworthy.

The possible team role measurement options were:

1. Primary roles considered only;
2. Primary and secondary roles considered;
3. Threshold method.

Role theory explains that a person may fill multiple roles. Belbin's work specifically talks about a person's *primary* and *secondary* roles. Measurement by each of the above methods poses its own specific problems.

For the sake of *forming* teams, consideration of participants by only their primary team role was rejected as a method. A person's primary role is simply the single role for which he scores the highest on the SPI. The sample was heavy in both Company Workers and Shapers, and consideration of only a person's primary role would have skewed the experiment data to suggest that the other

roles he may play (for practical purposes) were nonexistent. This could easily have resulted in poorly balanced teams with respect to team role.

The secondary role is the one for which a person scores second highest. When considering secondary role as a factor for team construction, another problem occurs: several participant team members scored a tie for secondary role. Consideration of a single secondary team role would have been deceiving in this case. Although there are some analysis situations where a tie can be statistically resolved (a method we used elsewhere for lack of any better alternative), that method is inappropriate for the initial team formation process.

Finally, previous work used the threshold method for team formulation. A comparison of number of Company Workers for each team using the three methods resulted in almost identical results, suggesting that the threshold method is reliable enough for initial setup.

E) Measuring Facets Other Than Team Effectiveness

While the threshold measurement scheme seemed a reasonable basis for the initial formation of the experiment teams, so that the effectiveness / relative success of each team could be quantitatively studied, other facets of the team interaction have enough impact on a team's effectiveness to warrant inclusion in this study. For these purposes, a role measurement determination system other than threshold may be more appropriate. This decision reflects an understanding of the ipsative problems of the SPI, and of the basis for Belbin's role theory itself.

Belbin writes of primary and secondary roles, and identifies them as the roles for which a subject measures highest and second highest, respectively. It was decided that these roles would be used for the next phases of this study, which sought to determine a team's viability, given the presence of various roles.

The determination of primary and secondary roles presented a further problem. For statistical study, definite primary/secondary roles were needed, but as the Belbin SPI is designed, it is possible for a subject to score a tie between two roles. One option in such a situation would be simply to throw out the difficult data, but this approach would have discounted over 10% of the study – a reasonably large portion. Instead, two parallel approaches were used.

With the first method, in the event of a tie between two roles, a "winning" role was chosen via random chance. If the tie was for a primary role, the "losing" role was recorded as the subject's secondary role. If the tie was for a secondary role, the loser was simply discarded. This approach allows the ability to guess as closely as possible given the sample data, while leaving the small number of unknown factors to chance, in hopes that random chance will balance out the results. In this method, in the event of a three-way tie, the data is discarded.

The second method considers all of the tied roles together, and requires specific questions to be asked of the data. If a question seeks a correlation between the viability of a team and the presence of a team member who was either a primary Shaper, any team would grouped into the "has a Shaper" category who had a member who had a tie for primary role (if "Shaper" was one of the tied roles). Three-way ties are included in this method. For experiments dealing with smaller data samples, the first method was used. In large samples, viz., the Belbin-Keirsey mapping described later in this Chapter, the later system was applied.

1) Viability and Appraisal of Activity

This research involved two secondary studies, each of which made use of the aforementioned multiple-analysis technique. The first of these two studies includes basic questions related to viability, such as "Did your team function well

together?" and "Were there any conflicts within the group that inhibited your progress?" For questions like this, rather than simply noting a conflict between 5 and 22, for example, analysis could note trends for likelihood of conflicts in teams of two primary Company Workers.

In the second of these peripheral studies, the multiple-analysis strategy was employed to study which team roles were more apt to be involved in which software engineering activities. After each experiment phase, participants were given a viability survey (Appendix C), which asked them (in addition to basic viability questions) to identify the contributions of each team member. The predefined software engineering contribution areas included design, coding/implementation, and testing/debugging. Subjects were allowed to attribute multiple activities to each person, and multiple people to each activity. This system made the resulting breakdown completely orthogonal.

F) Experiments With the SPI Itself

Finally, questions regarding the ability of one subject to identify his teammate's role were contemplated. Is a person "born" to play a specific role? Are some roles more likely to remain constant than others? Can a person identify his teammates' roles using the SPI? Will the test's validity generally be confirmed thought test-retest trials?

To this end, subjects were asked to retake the Belbin SPI after each phase of the experiments, and also to predict the answers their teammates would give to the same test. This provided not only for a means of checking role consistency for individuals, but provided the ability to discern whether a person was perceived by his teammates as playing the role he claimed to play. Along a similar vein, it allowed a potential examination into software engineering's "absent roles." Some of Belbin's roles had shown poorly in the data and in previous related studies. [STET98] The emergence of these roles could be observed though such a retest

scheme. This allows one to consider whether the absent roles are simply not *present*, or if they are actually not *needed* for software engineering.

In addition to a focus on the Company Worker, each of the viability and engineering activity-related experiments were designed so that trends for all of the present roles might be observed and recorded.

G) Mapping the Belbin SPI to the Keirsey Temperament Sorter

The Keirsey Temperament Sorter, described in Chapter 1, offers a means of defining personality based on four orthogonal scales. Similar in nature to the Myers-Briggs Type Indicator, it offers a fairly consistent picture of a person's overall personality characteristics. Prior research of software engineers had attempted to map programmers' Keirsey results to their Belbin SPI results. This earlier study, however, suffered from poor sample size. A first step in an acceptable study must be to identify the trends of each role with respect to each scale.

For the current study, several mapping possibilities were explored, but in the interest of obtaining observable correlations, each Keirsey scale was polarized. This was done so that a person could be characterized simply as an Introvert or Extrovert, rather than assigned a value identifying him as "65% introverted," for example. A study then sought to determine if team role had any effect on each Kiersey scale.

Why map the Belbin Roles to the Keirsey Temperament Sorter in the first place? The obvious benefit to such a pursuit, if it proved feasible, would be the ability to organize a team optimized with respect to Belbin team role theory, having only Keirsey data to use as input. If a person's Keirsey records are already available, perhaps reasonably accurate estimates can often be made of his team

role. Also, if Keirsey scales are shown to be more stable than Belbin roles, and if a significant correlation between the two can be drawn, the Keirsey scales may prove to offer a more reliable means of predicting team success when coupled with Belbin role theory.

Chapter IV: The Controlled Experiments

A) The Company Worker and Team Productivity, Effectiveness, and Success

To determine the general effect of the Company Worker on team productivity, the software engineering class experiment teams (described in Chapter 3) were observed, and records were taken of their activity. Observers also recorded measures of the teams' success, using two scales. For individual experiments, time to completion was used. Also, because not all teams completed every problem successfully, a count was kept of the number of problems successfully completed by each team within the 90-minute experiment periods. Each team worked together for four experiments, with a new problem for each experiment.

The number of Company Workers on each team ranged from zero to two. Because the Company Worker is the "Implementer" of the team, he was expected to have a positive effect on team success. More specifically, it was hypothesized that the presence of more Company Workers would make a team more successful.

B) Basic Description of Study Sample

Twenty-seven participants comprised the sample. A count of Primary Roles (determined as described by Belbin) included 6 Chairmen, 13 Shapers, 5 Plants, 0 Resource Investigators, 10 Monitor Evaluators, 12 Company Workers, 6 Team Workers, and 6 Completer-Finishers. (Note: In cases where a participant had two roles tied for primary, both roles were counted here. Table 4.1 shows specific breakdowns.) A count of roles determined by threshold method (where role existence is determined by a score of at least 12 for that role scale on the Belbin Self-Perception Inventory) included 2 Chairmen, 9 Shapers, 3 Plants, 0

Resource Investigators, 4 Monitor Evaluators, 9 Company Workers, 1 Team Worker, and 1 Completer-Finisher. The threshold method, described in detail in Chapter 3, was used for team formation, in light of the fact that a person may actually fill more than one role, and in order to address the tie problem that occurs in the Primary Role method. The threshold method acknowledges that a single person may assume multiple team roles. The Primary/Secondary role method, prescribed by Belbin, usually narrows results to a single primary and secondary role for each person, although it is possible in this method for a person to have a "tie" between two or more roles for Primary or Secondary. In Table 4.1, data on the study sample is shown with and without the tie cases. This data reflects two ties for primary role and six ties for secondary role (including one three-way secondary role tie.)

Table 4.1: Breakdown of the study sample by role, using various determination techniques.

Role	Threshold Method	Primary Roles, (without ties)	Primary Role Ties	Primary Roles, (with ties)	Secondary Roles (without ties)	Secondary Role Ties	Secondary Roles, (with ties)
Chairman	6	2	--	2	4	--	4
Shaper	13	8	1	9	5	1	6
Plant	5	2	1	3	1	--	1
Resource Investigator	0	0	--	0	0	--	0
Monitor-Evaluator	10	4	--	4	1	4	5
Company Worker	12	9	--	9	3		3
Team Worker	6	0	1	1	4	2	6
Completer-Finisher	6	0	1	1	3	2	5

As an initial observation, it is interesting to note the lack of Resource Investigators. Previous research seems to suggest that they are simply not needed in software engineering teams, because that for practical purposes, the role does not exist among the computer science community [STET98]. This argument logic is further explored in Chapter 6.

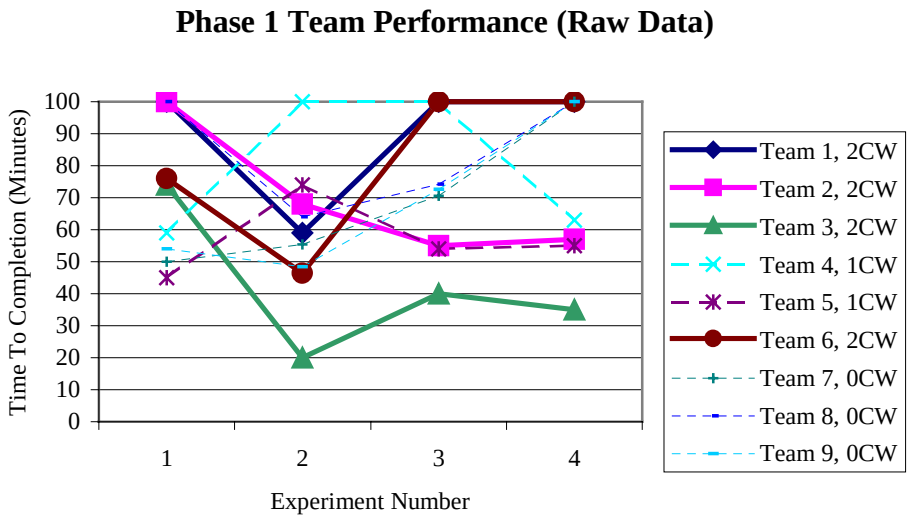


Figure 4.1: Team performance for phase 1, raw data, team perspective

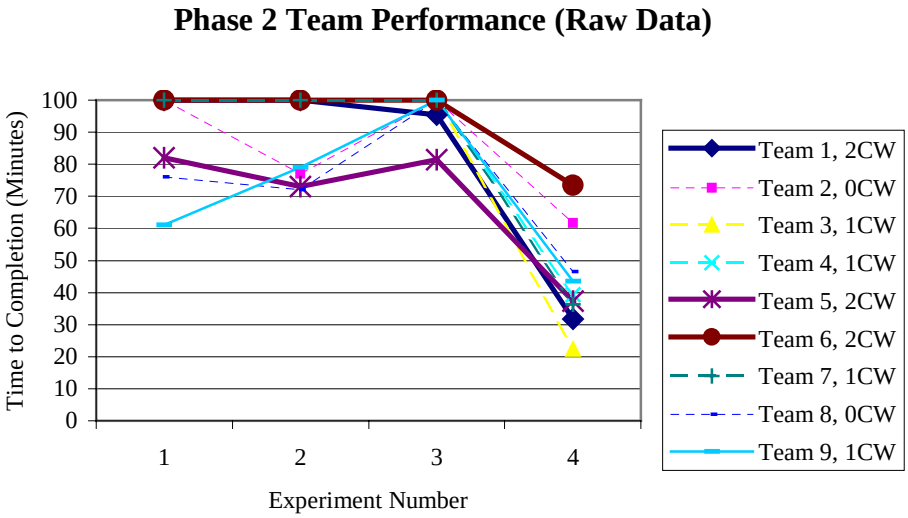


Figure 4.2: Team performance for phase 2, raw data, team perspective

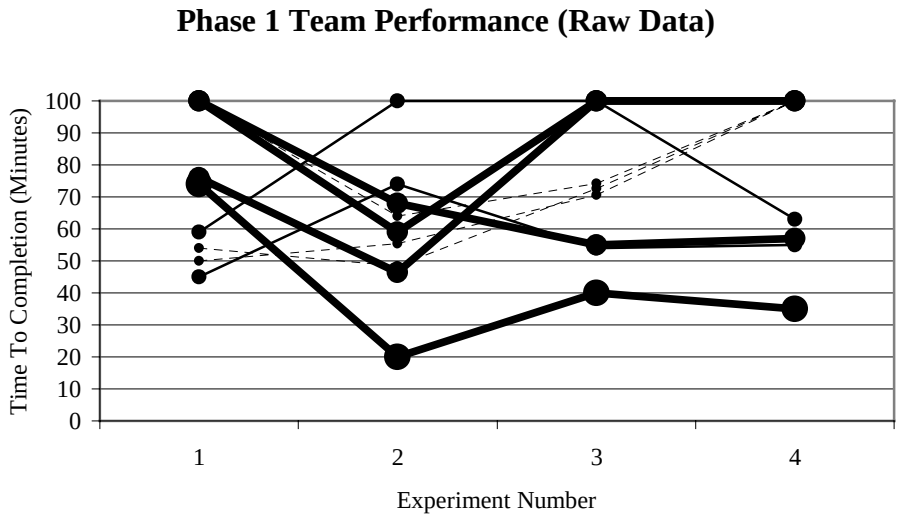


Figure 4.3: Team performance for phase 1, raw data, Company Worker perspective

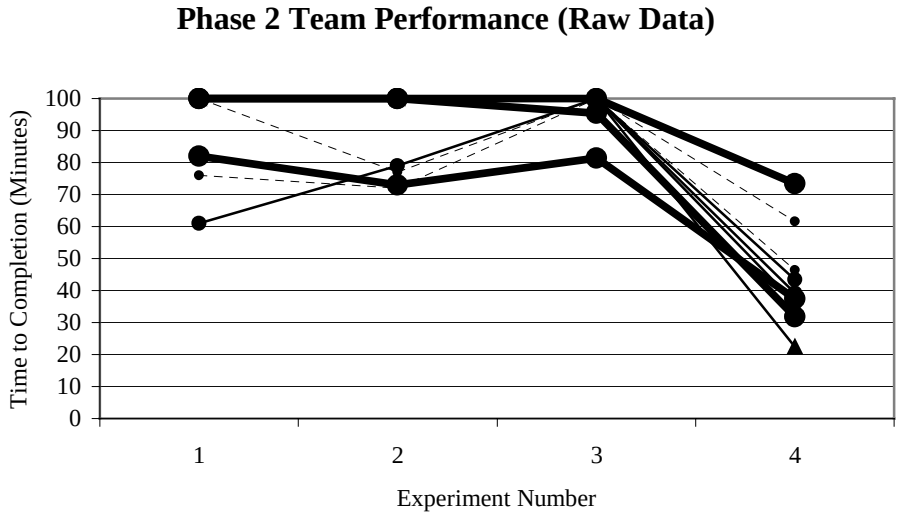


Figure 4.4: Team performance for phase 2, raw data, Company Worker perspective

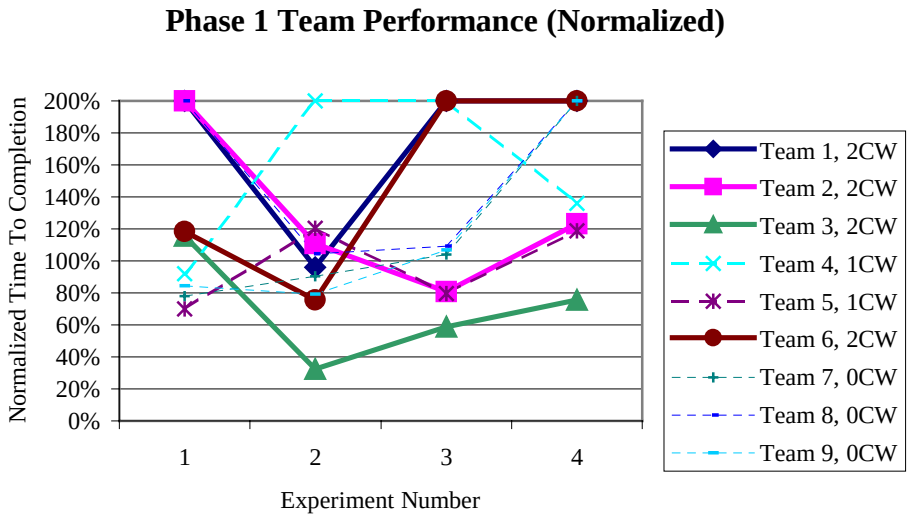


Figure 4.5: Team performance for phase 1, normalized, team perspective

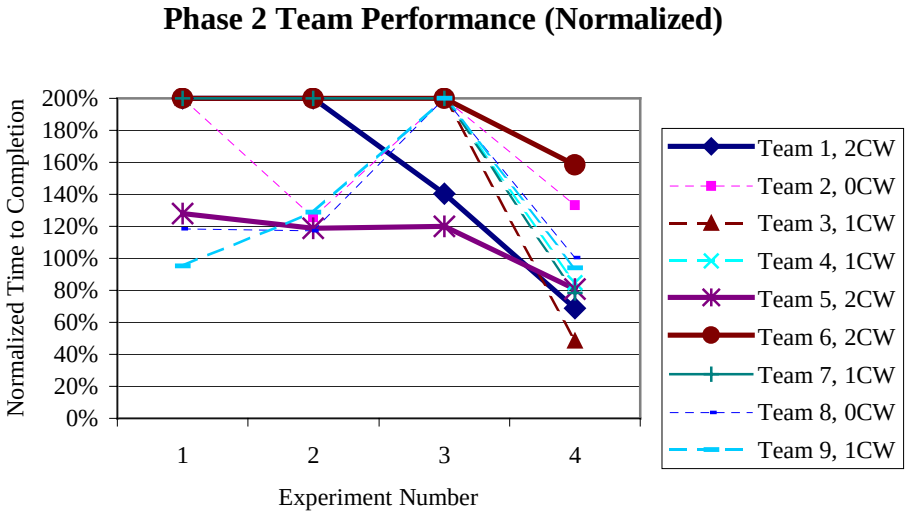


Figure 4.6: Team performance for phase 2, normalized, team perspective

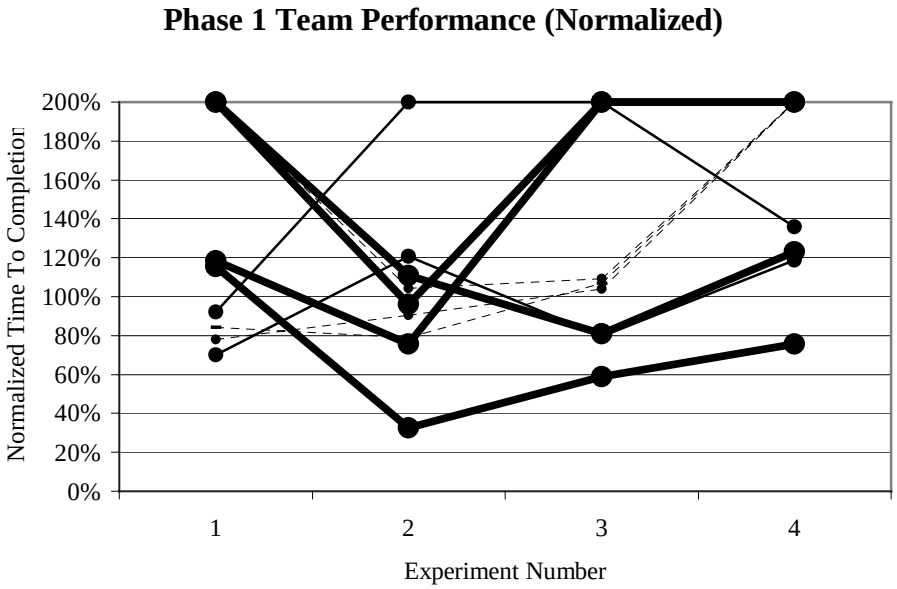


Figure 4.7: Team performance for phase 1, normalized, Company Worker perspective

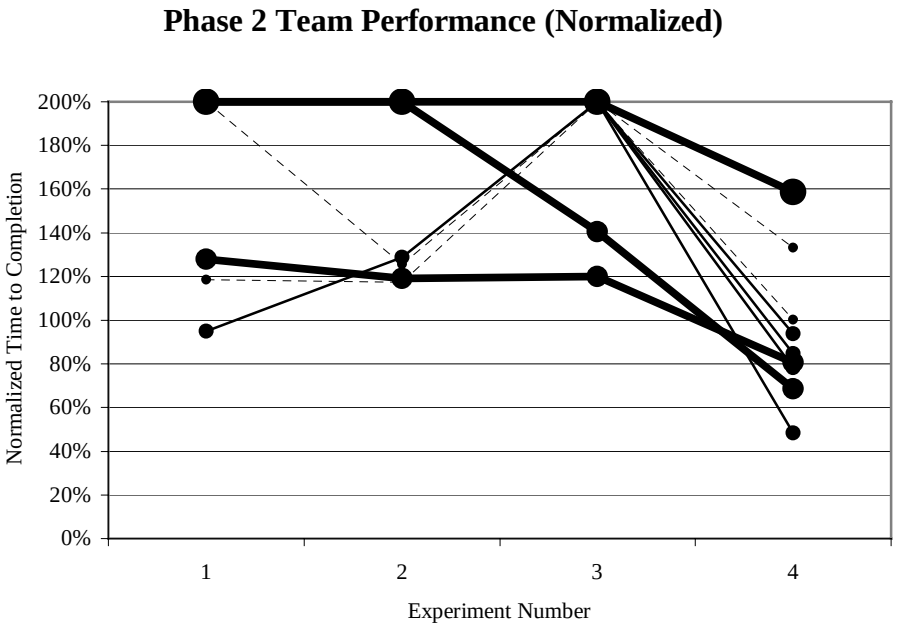


Figure 4.8: Team performance for phase 2, normalized, Company Worker perspective

C) Experiment 1: Company Worker vs. Success

It was hypothesized that teams containing *one or two Company Workers would perform better than teams with no Company Workers*, with time to completion and number of successes used as measures of performance. An analysis of the correlation between number of Company Workers on a team and team effectiveness (as measured by number of successes and time to completion), using Pearson's Product-Moment Correlation [SAS90], via **PROC CORR** in ***SAS*** produced insignificant results. The performance of teams with Company Workers, on average, could not be shown to be significantly different than teams without Company Workers.

The overall performance of each team over the course of experiments in each phase is shown in Figures 4.1 – 4.8. These figures represent three dichotomies. First, phase one and phase two of the experiments are graphed separately for clarity. Secondly it is desirable to have both raw representations of results (for the sake of records and accuracy) and normalized visualizations (for the sake of comparisons between problems of varying difficulty). Thirdly, different graphs offer the ability to either track performance by team number or by number of Company Workers. These combinations require 2^3 or 8 figures. Phase 1 graphs are shown in Figures 4.1, 4.3, 4.5, and 4.7. Phase 2 graphs are shown in Figures 4.2, 4.4, 4.6, and 4.8.

Ninety minutes were allotted for completion of each problem. In raw (non-normalized) representations, a completion time of 100 minutes denotes that the corresponding team did not complete the problem. In normalized representations, a normalized completion time of 200% denotes that the corresponding team did not complete the problem. For normalized graphs, 100% represents the average completion time for that problem. Raw data representations are shown in Figures 4.1 – 4.4, and normalized representations are shown in Figures 4.5 – 4.8. In all of these figures (Figures 4.1 – 4.8), each plotted line represents a unique team; each horizontal increment represents one

experiment, and the vertical scale denotes the time to completion for each problem. In Figures 4.1, 4.2, 4.5, and 4.6, each team is represented by a unique plotted line style. In Figures 4.3, 4.4, 4.7, and 4.8, heavy plotted lines designate teams with two Company Workers, light solid lines designate teams with one Company Worker, and dotted lines designate teams with no Company Workers.

When comparing teams' times to completion against the number of Company Workers on the teams, no significant correlation was found (**r** = -.02; **p** = .88). Using number of successes as a measure of effectiveness did not offer different results (**r** = -.10; **p** = .70).

```
PROC CORR DATA = SASDIR.COMPLETIONTIME_FILE;
VAR TIME_TO_COMPLETION;
WITH NUM_CWS;
RUN;
```

Figure 4.9: SAS code for correlating completion time against number of Company Workers on a team.

A scatter plot of the results inspired an interesting new theory. No upward or downward trend is visually detectable. It does seem, however, that the results are clustered close together for teams with no Company Workers, but more spread out along the y-axis for teams with more Company Workers. This suggests a new possibility for consideration: the Company Workers' presence may add variance to a team's performance. In simpler terms, adding a Company Worker may make a team less predictable. Figures 4.10 and 4.11 show bubble plots of the problem completion times for each successful team in each experiment. The appearance of increasing variance in time as number of Company Workers on a team increases may suggests that the presence of a Company Worker may make a team less predictable. The area of a bubble reflects the number of sample items at that discrete point. While Figure 4.10 shows raw completion times, Figure 4.11 displays the same data using a normalized time axis.

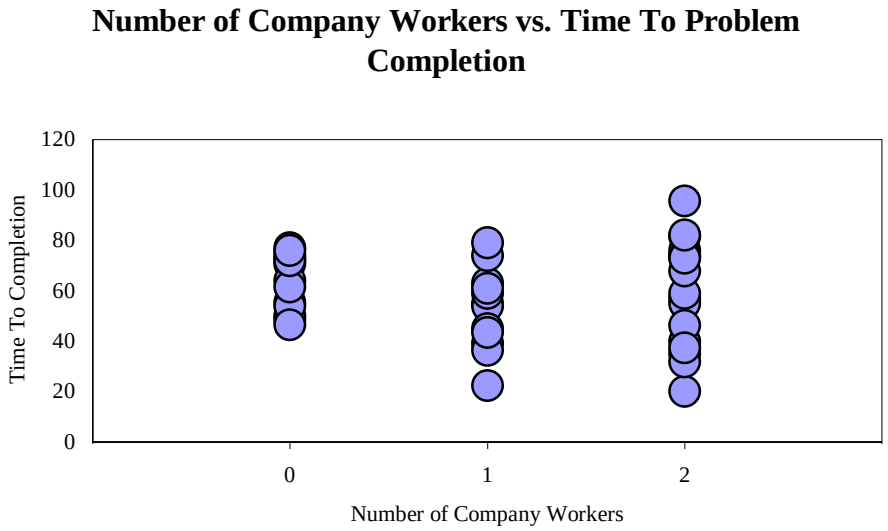


Figure 4.10: Number of Company Workers vs. problem completion time

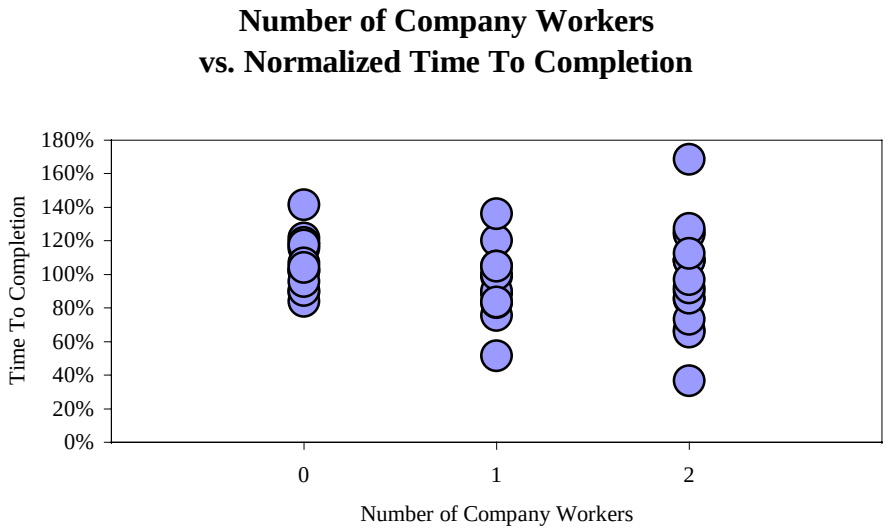


Figure 4.11: Number of Company Workers vs. normalized problem completion time

Expectations at this point were that as the number of Company Workers on a team increased, the team would become less predictable, and that this trend would manifest itself in increasing variance with respect to the predefined measures of team success: time to problem completion and number of successful problem completions. An *F* test for equality of variances produced statistically significant results [SNEG80]. Completion times were normalized among individual programming experiments to address issues related to varying program difficulty from one experiment to another. Variance for each group was determined using **PROC MEANS** in ***SAS***. (A "group" means all teams with the same number of Company Workers.) The ratio of variances between different groups in the sample was then compared against the critical *F*-value for confidence. When comparing zero-Company-Worker teams against two-Company-Worker teams using *time to completion* as a measure of effectiveness, statistical significance was found at the $\alpha = 0.05$ level of significance (**p** < .05; $F(16,12) = 2.599$). Significance could not be found when comparing zero-Company-Worker teams against one-Company-Worker teams ($F_{p=10}(10,12) = 2.188$) or one-Company-Worker teams against two-Company-Worker teams ($F_{p=10}(16,10) = 2.233$), possibly owing to the small sample size. (Only teams that actually completed the problems could be counted in this analysis.) However, the trend of increasing variance is visually detectable (see Figure 4.10 – 4.11).

Using *number of successes* as a measure of effectiveness produced significance at the $\alpha = 0.10$ level of significance (**p** < .10; $F(5,4) = 4.051$) when comparing the variance between teams with zero Company Workers and one Company Worker. Significance could not be found comparing teams with one and two Company Workers, although the trend can still be seen visually. Confidence at the 95% level was found when comparing teams with zero and two Company Workers (**p** < .05; $F(6,4) = 6.163$). When data was normalized, the only statistically significant difference in variance could be measured between zero and two Company Worker teams (**p** < .10; $F(6,4) = 4.010$). Figures 4.12 – 4.13 show bubble plots of this data. Figure 4.12 shows discrete number of

successes plotted against number of Company Workers on each team. The area of a bubble reflects the number of sample items at that discrete point. Figure 4.13 shows the same data, but normalizes the number of successes for teams. A sample graphed vertically at 100% would have an average number of successful completions.

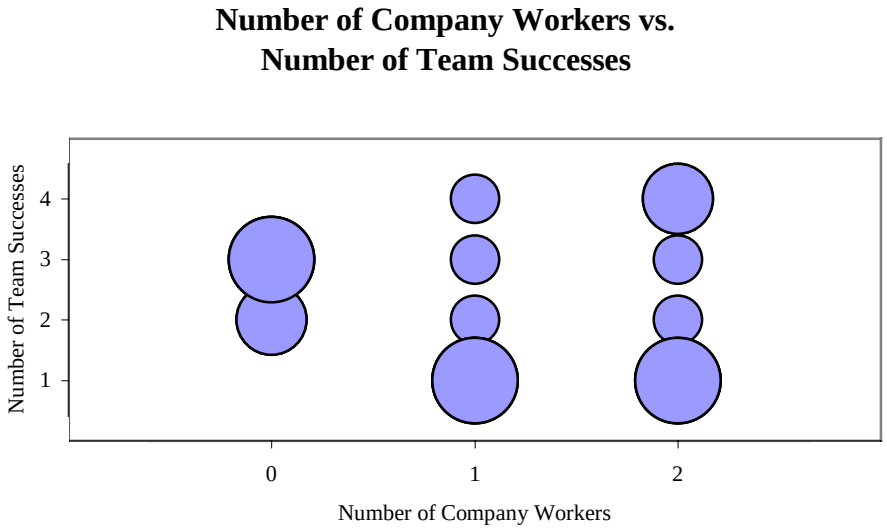


Figure 4.12: Number of Company Workers vs. number of team successes

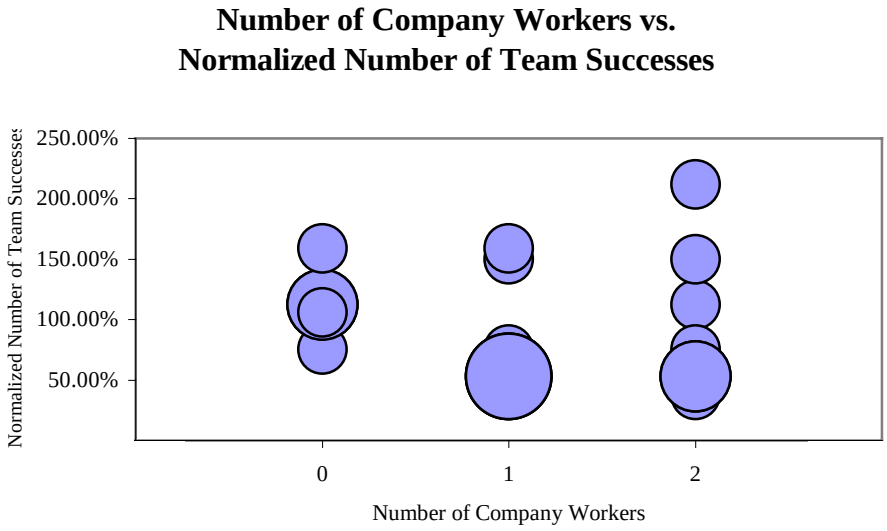


Figure 4.13: Number of Company Workers vs. normalized number of team successes

Two other measures of success were considered. Since teams were allowed to keep submitting solutions while time remained in the experiment, it was conjectured that (for teams which successfully solved a problem) the number of team submissions might bear a similar relation to the number of Company Workers on a team, but no trend could be detected in any way. Teams also used a prescribed self-rating scale to determine methodically how close they came to success, however the survey itself was flawed. First, no realistic claim could be made to the effect that the scale intervals were evenly spaced. Secondly, the survey itself contained a fundamental typographical error, which rendered it invalid.

```
PROC MEANS VAR;
BY NUM_CWS;
VAR TIME_TO_COMPLETION;
RUN;
```

Figure 4.14: SAS code for determining variances of discrete groups in the sample.

The results provoked speculation to explain the observed phenomenon of Company Workers contributing the variance in team success. Initial thoughts might lead one to suspect that Company Workers simply aren't well suited to Computer Science. This possibility can be discounted, however, by the simple fact that none of the participants were particularly poor academically in the class being studied. The senior level software engineering course in which they were enrolled requires substantial prerequisite programming experience, and tends only to attract bright students seeking a challenge.

A second conjecture draws on Belbin's role theory. Company Workers, as stated earlier, tend to seek out the tasks often perceived as unpleasant or undesirable, but which are crucial to the success of the company or team. It is conceivable that the Company Worker may simply require sufficient time to observe and divine the needs of the team before committing a direction to his efforts. As the study experiments lasted only 90 minutes, the Company Worker

may have simply not had the time to find his niche within the team. This reasoning inspired another experiment, which is described in experiment 2 of this chapter.

One might plausibly reason that the Company Worker role may be a redundant concept within the structure of a software engineering team. Belbin describes the Company Worker as the "Implementer" of a team. Most of the team models outlined and described in chapter 1 describe key implementation as being done by a pair of programmers. The descriptions of people likely to suit these roles effectively tend to reflect the Plant and Monitor-Evaluator. If this is the case, the concept of a team "Implementer" may be superfluous.

Other experiments (Experiment 4 – Experiment 6) were conducted, which sought to determine if the various roles were predisposed to work in specific software engineering activities. Analysis of this experiment's results led to a fourth possibility for explaining the surprising effect of the Company Worker on team success, which is detailed later in this chapter, in Experiment 7.

D) Experiment 2: Company Worker vs. Long-Term Success

To test the premise that the Company Worker may require more time to find his niche within the team, a comparison was made between subjects' team roles and their assigned final grades for the software engineering class by which they participated in this study. This measure was used because the final grade assigned for the software engineering class was very largely based on a subject's interaction with no fewer than five separate teams, each of which worked together for the majority of the semester. These teams did *not* include the ones studied in Experiment 1, above.

Final course grade was based on interaction with (and success of) the subject's design team, at least three *other* design teams for which the subject worked as a programmer, and at least one technical review team for which the subject served as an independent observer. In each case (except the technical review group), team members' input helped to decide a subject's final grade.

Using **PROC FREQ** and Fisher's Exact Test in ***SAS*** [SAS90], no significant difference could be determined between roles with respect to final course grades. It is therefore concluded that course success is not dependent on team role (**p** = .89).

```
PROC FREQ DATA = ROLE_VS_GRADE;
TABLE PRIM * GRADE/EXACT EXPECTED;
RUN;
```

Figure 4.15: SAS code for determining interdependence between role and grade.

The negative results of this experiment may reflect reality (role and long-term success may really *be* independent). However, very little variance existed in overall in grades for this class. This fact alone may contribute to the lack of significant results.

E) Experiment 3: Number of Roles Present vs. Team Success

Analysis was done to determine if the number of different role types on a team had a statistically significant impact on a team's success. In this case, if a team had two primary Company Workers and one primary Shaper, for example, two distinct roles would be counted for that team. Using **PROC CORR** in ***SAS***, several measures were used to determine if a relation existed between number of team roles present and team success.

Number of roles present was measured using three tools. These consisted of number of distinct primary roles present on a team, number of distinct roles present (whether primary or secondary), and number of distinct roles existing on a team as determined by threshold method. After each phase of experiments, participants were again administered the Belbin SPI and asked to answer its questions in the context of the team in which they had worked. The three measures of present roles were employed using both the original Belbin surveys and the surveys collected after each phase, for a total of six measures. Team success was measured using the three conventional tools: number of success, time to completion, and normalized time to completion. In all, eighteen analyses were made of present roles against team success. Every possible combination was applied.

Results of this analysis were not strong enough to support any real claim about the effect of number of roles present on team success. Statistically significant results ($\mathbf{p} < .05$) could be found for only two of the eighteen correlations. Number of team successes had a weak negative correlation with the count of roles present on a team, as measured by threshold method using the initial Belbin survey ($\mathbf{r} = -.49$, $\mathbf{p} = .04$). Similarly, time to completion successes had a weak negative correlation with the roles present on a team, as measured by total number of unique discrete primary and secondary roles using the initial Belbin survey ($\mathbf{r} = -.32$, $\mathbf{p} = .04$). Measurement of the same concept using the other tools produced inconsistent and insignificant results. It should be thus concluded that the number of team roles present on a team generally does not hold a significant impact over the team's success.

```
PROC CORR DATA = NUM_ROLES_VS_SUCCESS;
VAR ORIG_NUM_PRIM_ROLES -- RETEST_NUM_THRESH;
WITH NUM_SUCCESS -- Z_TIME;
RUN;
```

Figure 4.16: SAS code for correlating number of unique roles on a team with team success.

F) Experiment 4: Team Role vs. Propensity Toward Activity In Testing Phase

The Company Worker seemed likely to be drawn to the testing and debugging phase of software engineering. This theory stemmed from the observation that programmers generally dislike testing their own code. This is because a successful test (one in which a bug is found) implies an unsuccessful program. The programmer testing his own code is often doomed to have his program code fail, unless his test cases are (whether intentionally or subconsciously) weak. A Company Worker's inclination to accept needed but undesirable responsibilities would make the connection between testing and the Company Worker seem feasible [MEYG79].

A viability questionnaire collected from each participant after each phase of experiments asked teammates to identify which members of their teams were involved in which of three phases of the software engineering process: design, implementation/coding, and testing/debugging. No limitations were imposed on this section of the questionnaire. Subjects were allowed to attribute multiple activities to each person, and multiple people to each activity. This system made the resulting breakdown completely orthogonal. Participants were encouraged to identify all teammates who had worked on each phase, including themselves. With this system, it was possible to identify and differentiate between the participants who *considered themselves* tester/debuggers (for example) and the participants who were *identified by their teammates* as tester/debuggers.

An analysis was done using Fisher's Exact test in ***SAS*** [SAS90]. This analysis sought to determine if team role and predisposition toward the testing phase shared any degree of interdependence. Although interesting trends were discovered in the data, the test did not produce statistical significance. When participants rated their own testing/debugging-phase contribution, no statistical evidence could be shown to disprove the independence of team role and likelihood of activity in testing and debugging (**p** = .36). When participants rated

their teammates' design-phase contribution, again, no statistical evidence could be shown to disprove the independence of the two concepts (**p** = .13).

```
PROC FREQ DATA = ROLE_VS_DESIGN_CODE_DEBUG;
WHERE RATED_IS_SELF = 1;
TABLE PRIM_ROLE * IS_DESIGNER/EXACT EXPECTED;
RUN;
```

Figure 4.17: SAS code to explore dependence of team role on (self-determined) design activity

```
PROC FREQ DATA = ROLE_VS_DESIGN_CODE_DEBUG;
WHERE RATED_IS_SELF = 0;
TABLE PRIM_ROLE * IS_DESIGNER/EXACT EXPECTED;
RUN;
```

Figure 4.18: SAS code to explore dependence of team role on (teammate-determined) design activity

The difference between activities of the Chairman and Plant during the testing/debugging phase is of particular interest. Belbin suggests that these two roles work well together, and that pairing them together is an effective strategy. When considering the percentage of participants involved with the testing/debugging phase, observation of the data shows very little deviation in each role from the overall average of the sample, and very little variation whether roles are self-rated or teammate-rated. Chairmen and Plants in the sample, when ascertaining for themselves whether or not they were involved in debugging, were the most common and least common debuggers, respectively (100% Chairmen, 33.33% Plants). When activity in testing/debugging was judged by teammates, however, these trends reversed. Chairmen were least often perceived by teammates as debuggers, and Plants among the most frequently observed as debuggers (25% Chairmen, 66.67% Plants). Perhaps collaborating Plants and Chairmen work together well enough that the contributions of one are mistaken for the contributions of another. Chairmen and Plants make up a small enough portion of the sample, however (four Chairmen and six Plants out of twenty-seven participants), that any suggestion is likely to be little more than conjecture (see Figures 4.19 – 4.20). Note that when rating themselves, a disproportionately high

percentage of Chairmen and the disproportionately low percentage of Plants are reported in testing/debugging activity, but when rated by teammates, this sample trend is reversed. This observation must be tempered with the understanding that neither of these roles were *strongly* represented in the sample.

Figures 4.19 – 4.24 bear a short explanation. For each role, these bar graphs show the percent of subjects who were involved in a particular software engineering activity, such as debugging. A pure count-based representation would be inappropriate here, because there were unequal numbers of roles represented in the sample. As a result, overall sample averages of this data cannot be determined by simple visual parsing. For this reason, a dotted line appears in each graph to show the percent of subjects overall involved in the pertinent software engineering activity. In these graphs, "CH" denotes Chairmen, "SH" denotes Shapers, "PL" denotes Plants, "ME" denotes Monitor-Evaluators, "CW" denotes Company Workers, and "CF" denotes Completer-Finishers. Resource Investigators and Team Workers do not appear in these graphs because of their poor showing in the sample. Roles here are determined by the primary role method. These figures are grouped in pairs, with the first of each pair showing results from self-reported data (each subject declared whether or not he was involved in the relevant software engineering activity), and the second graph showing results from teammate-reported data (the subject's teammates determined whether or not he was involved in the relevant software engineering activity).

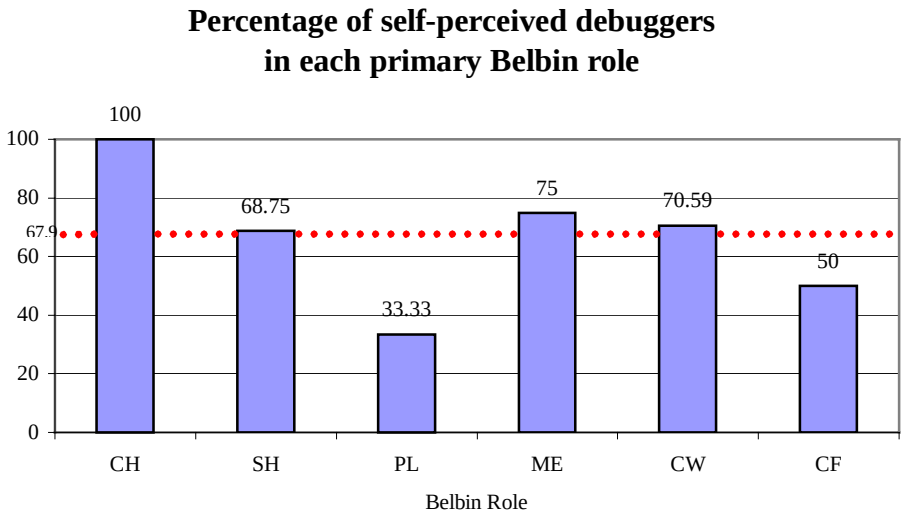


Figure 4.19: Percentage of self-perceived debuggers in each primary Belbin role

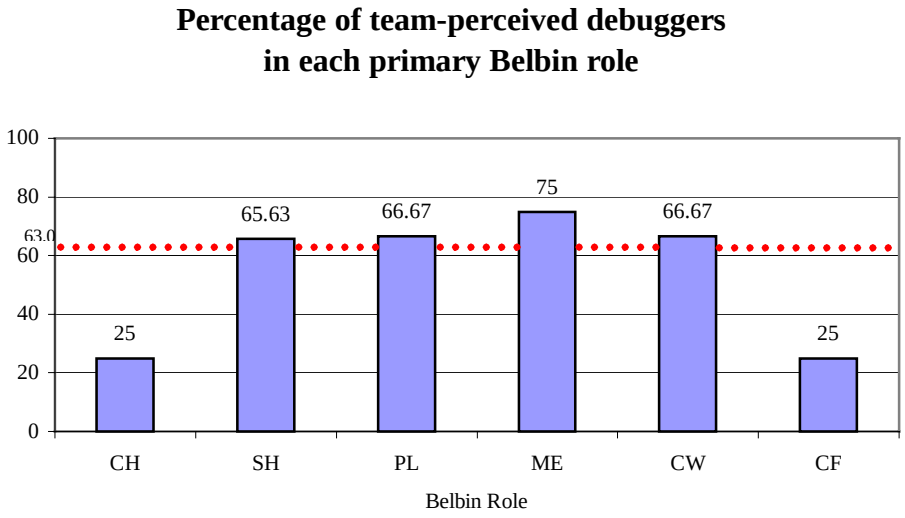


Figure 4.20: Percentage of team-perceived debuggers in each primary Belbin role

In survey responses, very few participants' comments were directed specifically at describing testing and debugging activities, and those comments that did mention testing and debugging tended to be very brief and general. However, in the field notes taken by an outside observer, the greatest number of comments dealing with testing and debugging are attributed to the Company Worker, Monitor-Evaluator, Plant, and Shaper roles. The percentage of participants' comments regarding Plants debugging was more than twice the percentage of Plants in the study sample. This could not be said about any other role. Chairmen received no mention in the observers' notes with respect to testing or debugging. The observer was unaware of roles when taking notes, and made mention only of team number and participants' initials.

Interestingly, the Completer-Finisher, known for tying up loose ends and attention to detail, made a poor showing as a debugger in this sample. Only 50% of Completer-Finishers saw themselves as debuggers, and teammates identified them as such 25% of the time. Completer-Finishers account for only two participants out of twenty-seven, however. Only one comment in the observers' notes involved a Completer-Finisher doing testing and debugging.

G) Experiment 5: Team Role vs. Propensity Toward Activity In Coding Phase

When evaluating their own degree of activity in the coding phase of the software engineering projects, participants did not show a significant degree of interdependence between role and activity in coding (**p** = .21). Similarly, when degree of involvement in coding phase was evaluated by their teammates, no significant interdependence appears between participants' roles and likelihood for activity in coding (**p** = .15) (see Figures 4.20 – 4.21).

In the sample, it is interesting to notice the disparity between the percent of Shapers and the percent of Company Workers involved in the coding phase, both with regard to self-determination of coding activity and teammate-determination of coding activity. In the study sample, Shapers were most frequently involved in coding, while Company Workers seemed to shy away from coding. Shapers and Company Workers account for the majority of the study sample, making up 30.2% and 32.1% of the sample, respectively.

It is also interesting to note the division between roles *more* often involved in coding than the average subject and those *less* often involved in coding than the average subject. In both cases, Shapers and Monitor-Evaluators were more often involved in coding than the average participant. No other role shares this feature in the study sample. This separation is identical in the sample, whether one considers coding rated by self-perception or team-perception.

In free-answer surveys, Shapers in the study described their coding activity in the study with quotes like, “I coded everything. [My teammates] watched. Sometimes they contributed ideas which made it faster,” and, “I did it. Had some help from both team members.” One participant reported, “[The Shaper] did coding. [One Company Worker] did design. [The other Company Worker] criticism.” Not only does this comment acknowledge the Shaper’s propensity to coding and the Company Worker’s tendency to shy from it, but also notes the varying tendencies of the Company Worker in general.

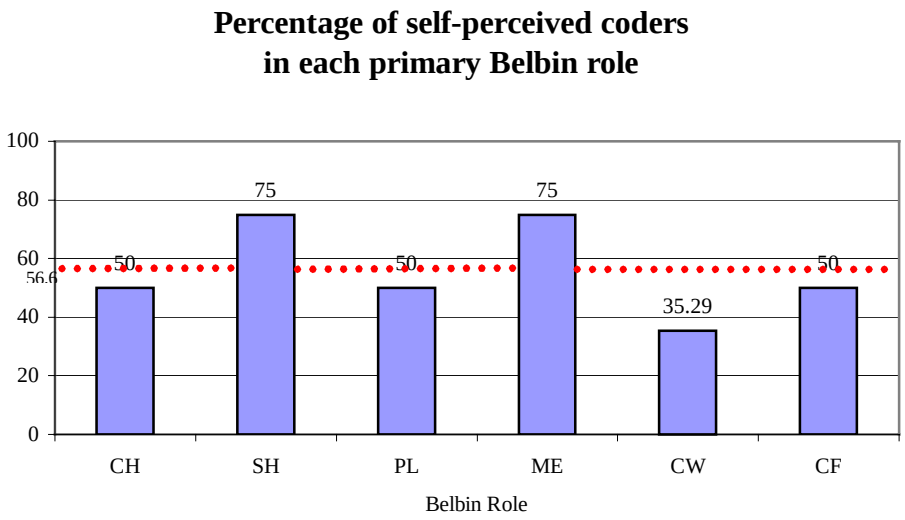


Figure 4.21: Percentage of self-perceived coders in each primary Belbin role

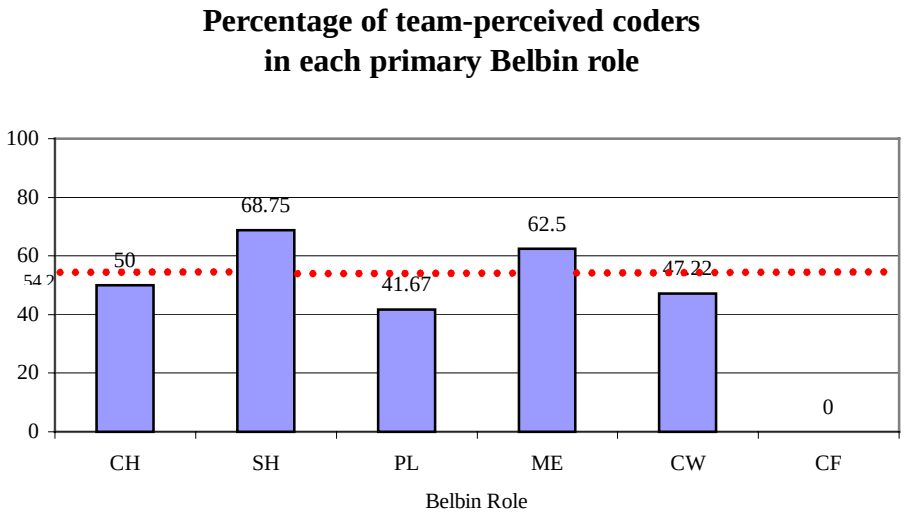


Figure 4.22: Percentage of team-perceived coders in each primary Belbin role

H) Experiment 6: Team Role vs. Propensity Toward Activity In Design Phase

Analysis of the design phase yielded statistically significant results. In this case, most subjects (72%) perceived themselves as designers, while only 56% were perceived by their teammates in that position. The same process in ***SAS*** (Fisher's Exact test) confirmed that Belbin team role and tendency toward participation in design are not independent (**p** = .04 for self-perception, **p** = .02 for teammate perception) (see Figure 4.23 – 4.24).

In the sample, Shapers least frequently identified themselves as designers (44%) For teammate perception, they were still well below the average (38%), but Completer-Finishers were seen by their teammates as even less involved in design (25%). In both analyses, Company Workers were most frequently involved in design [SCHP01]. Interestingly, the most extreme results (extreme meaning a noticeable deviation from overall sample average) were those roles most richly represented in the sample. Together, Shapers and Company Workers account for only 25% of the possible roles (as they represent two of the eight Belbin roles), but in the study sample, they accounted for 62% of the population.

Participants' free-form survey comments about their experiences on the study teams reinforced the statistic observations about roles and design. One participant wrote, "[One teammate] and I [both Company Workers] designed. [The other teammate, a Shaper] coded…" Another Company Worker wrote about his Shaper teammate, "I got [him] going on coding while I worked on the algorithm." Another participant wrote, "[The Shaper] did most of the actual coding. [The other teammate] and I [both Company Workers] did most of the algorithm stuff, except on problem one." These types of comments were fairly common, and generally seemed to represent the most detailed types of free-form comments on the surveys.

This phenomenon sparked a new hypothesis: given the observations that Company Workers seem to add variance to a team's success, and that Company Workers seem drawn to design, perhaps these two trends have a common root. Perhaps adding more designers to a team is what makes the team unpredictable. The old adage that "too many cooks spoil the broth" is an appropriate analogy. This notion warranted an additional analysis, in which the design and success concepts with regard to the Company Worker are combined. This new analysis is detailed as Experiment 7. Plants and Monitor-Evaluators also demonstrated an attraction to the design phase in this analysis, but they accounted for a much smaller portion of the sample than did the Company Worker.

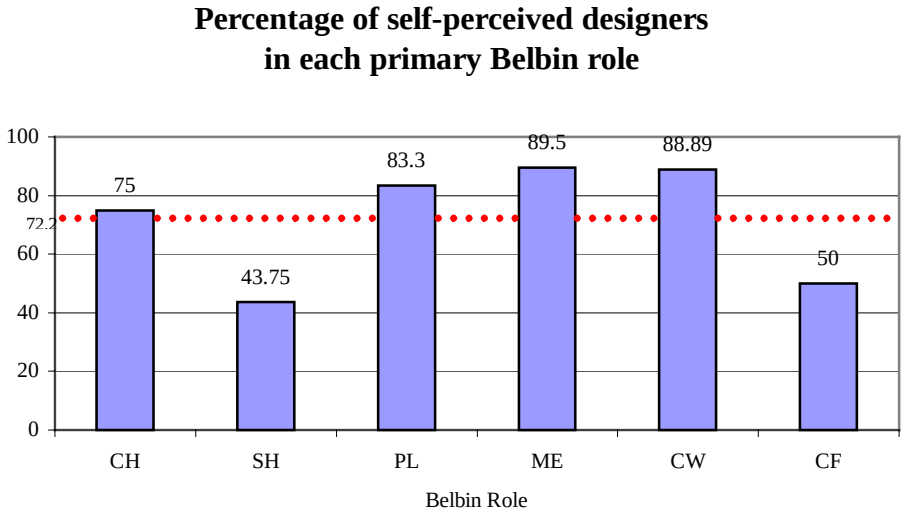


Figure 4.23: Percentage of self-perceived designers in each primary Belbin role

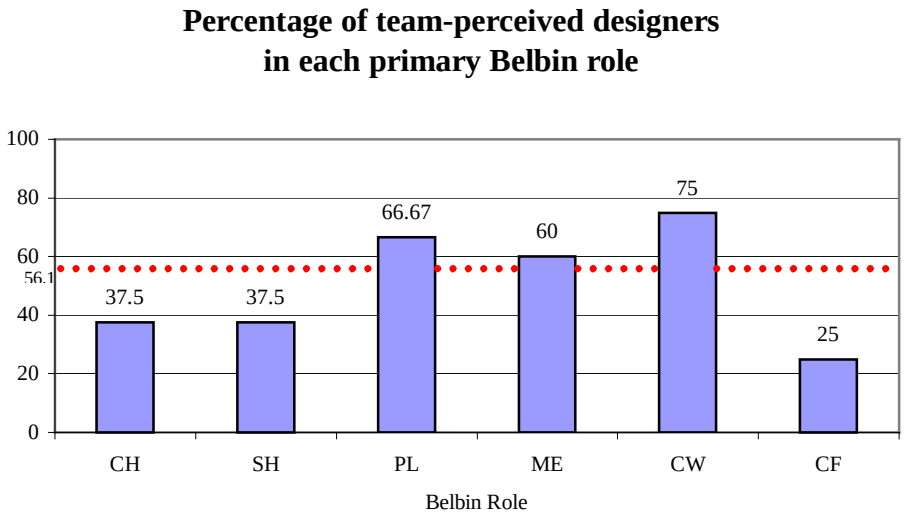


Figure 4.24 Percentage of team-perceived designers in each primary Belbin role

I) Experiment 7: Relation Between The Company Worker, The Designer, and Team Success

To ascertain whether the Company Worker's predisposition toward the design phase was related to their effect on team success, the success measures were restudied, this time using number of designers on a team as a means of grouping. Measures of team success included number of total successes and time to completion. Number of designers on a team was determined in two ways: first, a count was taken of the number of team members who claimed to have been designers. Then, a count of each team member's estimation of their teammates' involvement in design was considered, and the sum of this count for all three team members was used. These two methods offered two perspectives: one in which involvement in design was self-determined, and another in which design involvement was determined by peers. Possible values for the first method naturally range between zero and three, while values for the second method range between zero and six.

Self-rated determination showed no variation trend at all statistically, but visual inspection suggests that this may be because of outliers and small sample size. (Visual representation seems to show a hint of the increasing variance trend, although this method of analysis is admittedly speculative.) The lack of statistical significance here may be because people in general were more likely to dub themselves designers than they were to so dub their teammates. Normalized results are shown in bubble plots in Figures 4.25 and 4.26. Figure 4.25 shows normalized time to completion for teams with one, two, or three self-proclaimed team designers. Figure 4.26 offers similar data for number of successful problem completions. The relative size of bubbles indicates the number of samples at that discrete point.

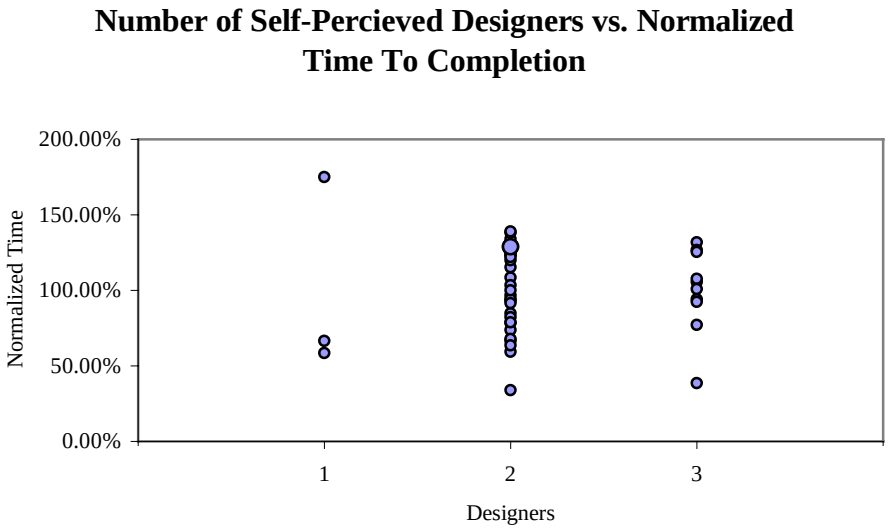


Figure 4.25: Number of self-perceived designers vs. normalized time to completion

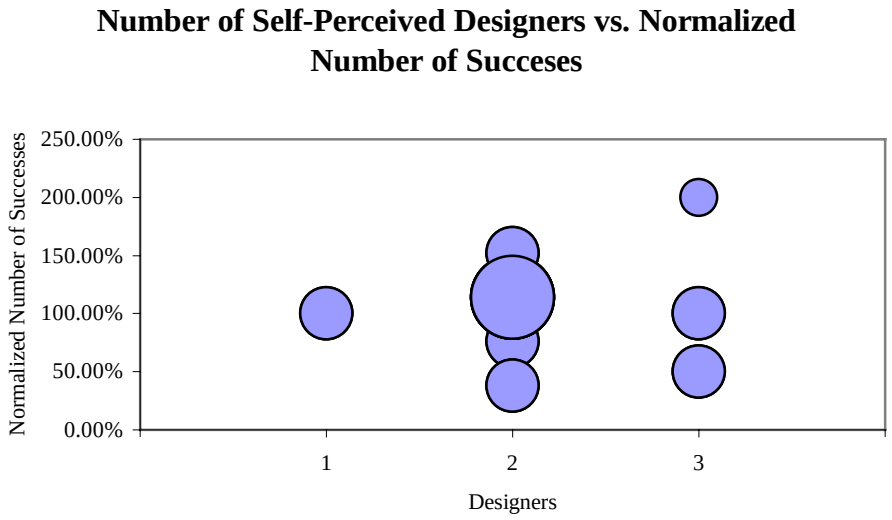


Figure 4.26: Number of self-perceived designers vs. normalized number of successes

Visual representation of the teammate-appraised data bore a resemblance to the Company Worker vs. Success study (from Experiment 1, above). Increasing variance can be seen easily in graphical representations, but in most cases, statistical significance could not be found (Figures 4.27 – 4.30). This may be due to the fact that there were a relatively large number of groups, each with a relatively small part of the sample. To reach significance, variance ratios are required to be higher for small samples than for large sample. (A "group" in this context is the set of all teams with the same measure of designers on the team.)

Figures 4.27 – 4.30 show bubble plots of success rates (problem completion time and number of successful problem completions) against the number of teammate-counted designers on each team. Note that even though there are three people on each team, this designer count can range from zero to six, because for each person on the team, an estimation of design activity is given by both of his teammates. For each success rate, both raw and normalized plots are graphed. In these graphs the trend of increasing variance for this sample is still visually detectable, but in most cases, significance was not found to support

this notion. In these figures, the relative size of bubbles indicates the number of samples at that discrete point.

This data was examined and analyzed using an F test for equality of variances [SNEG80]. With respect to self-rated designers, no statistically significant difference in variance was determined between any two groups. When the teammate-determined method was used and normalized time to completion was studied, significantly different variances between some groups could be determined, but this trend was not found to be consistent – again, possibly because of the weak sample size. The only group that did not show any significant degree of difference in variance with any other group was also the smallest group (the two-designer group, with only two instances). The greatest degree of significance was found between the two largest groups (the four-designer group, with nineteen instances and the three-designer group, with twelve instances). It seems reasonable to believe that a larger sample would have shown a more significant degree of variance between groups.

The most significant differences in completion time variance between teams grouped by number of teammate-assessed designers are summarized in Table 4.2. The table shows which two groups are being compared, the size of each group, and the ratio of variances in completion time for those groups. Also shown are the F-values required for significance. These values must be ties to a specific confidence level (provided in the table), and the degrees of freedom for the test ($Group1_{SIZE}$-1, $Group2_{SIZE}$-1). If the variance ratio exceeds the required F-value, the test is positive, and the variances are determined to be different statistically [SNEG80] [MILJ95].

Table 4.2 Statistically significant variance rations for groups with different numbers of designers.

Time To Completion							
Group 1	**size**	**Group 2**	**size**	**Ratio$_{VAR}$**	**Degrees of Freedom**	**Confidence Level**	**F**
5 designers	4	2 designers	7	3.313	(3, 6)	90%	3.289
4 designers	18	3 designers	11	2.822	(17, 10)	95%	2.812
5 designers	4	3 designers	11	3.623	(3,10)	90%	2.728

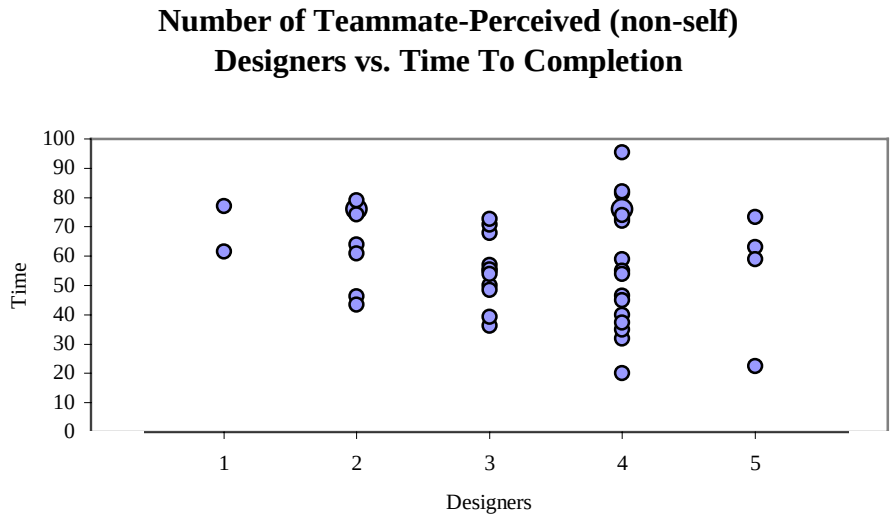


Figure 4.27: Number of teammate-perceived designers vs. time to completion

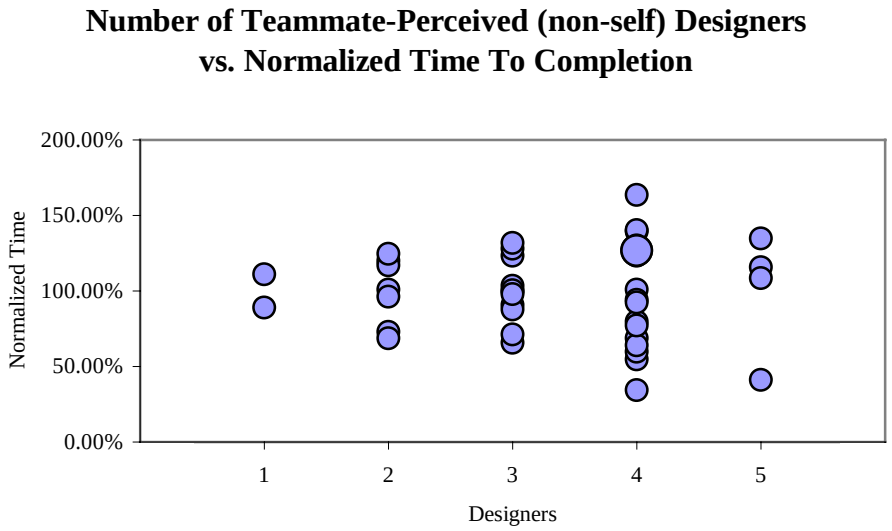


Figure 4.28 Number of teammate-perceived designers vs. normalized time to completion

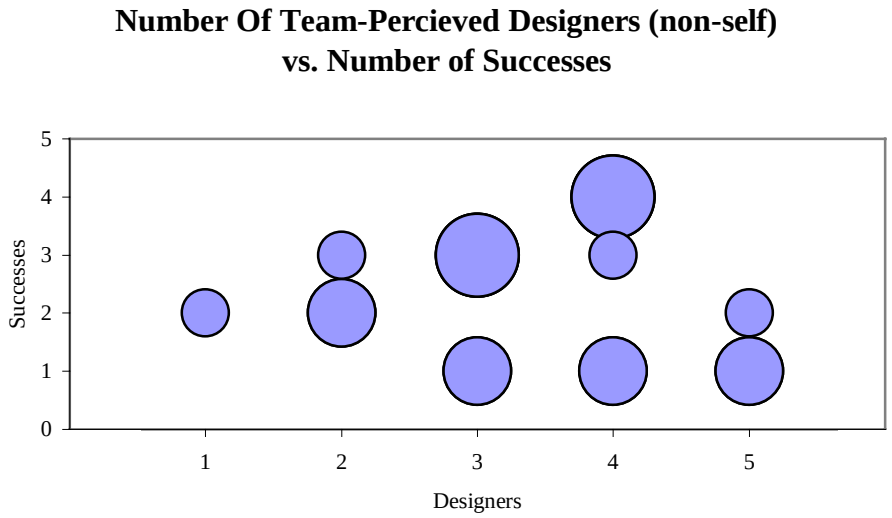


Figure 4.29 Number of Teammate-Perceived Designers vs. Number of Team Successes

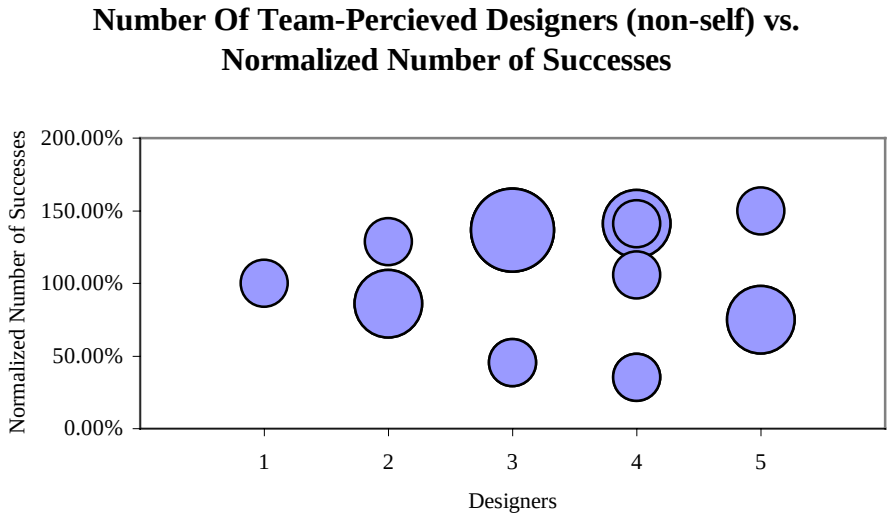


Figure 4.30 Number of Teammate-Perceived Designers vs. Normalized Number of Team Successes

The observed visual increase in variance from one group to another in the sample, paired with the lack of statistical significance (in most cases) may suggest that the Company Workers' inclination to the design phase is *partially* responsible for the increasing variance effect observed in the first experiment, but other undiscovered factors also are at work.

J) Experiment 8: Number of Company Workers vs. Overall Team Viability

An investigation was done to determine whether the number of Company Workers on a team had an impact on overall team viability. More specifically, did the teams work together well, or did conflicts within the group prevent easy interaction? Within the viability survey given to participants after each phase of experiments was the question, "Did your team function well together?" A six-point scale allowed participants to evaluate the degree to which the team cooperated, with "**1**" being "not at all" and "**6**" being "very much." The number

of Company Workers on each team was compared with the each of the teammate responses to this viability question. Visual parsing of a bubble plot showing number of Company Workers vs. degree to which team members worked together (Figure 4.32) might suggest to some that teams with more Company Workers work better together. This notion, however, is not supported statistically. Using **PROC CORR** in ***SAS***, no significant correlation could be made with this method between the number of Company Workers on a team and the degree to which a team worked together (**r** = .13, **p** = .34). The relative size of bubbles in this figure indicates the number of samples at that discrete point.

From a team role theory perspective, there is no reason to think that the Company Worker (any more than other roles) would cause a team to have a greater degree of cohesion, or a greater ability to work together. The Team Worker role conceptually would promote smooth operation of a team, but too few Team Workers existed in the sample to make such a study feasible. Furthermore, teams were balanced as much as possible with respect to role (with the exception of the Company Worker, for control purposes).

```
PROC CORR DATA = VIABILITY_Q7;
VAR NUM_CWS_ON_TEAM;
WITH WORK_TOGETHER;
RUN;
```

Figure 4.31 SAS code for correlating number of Company Workers on a team with scale-based data

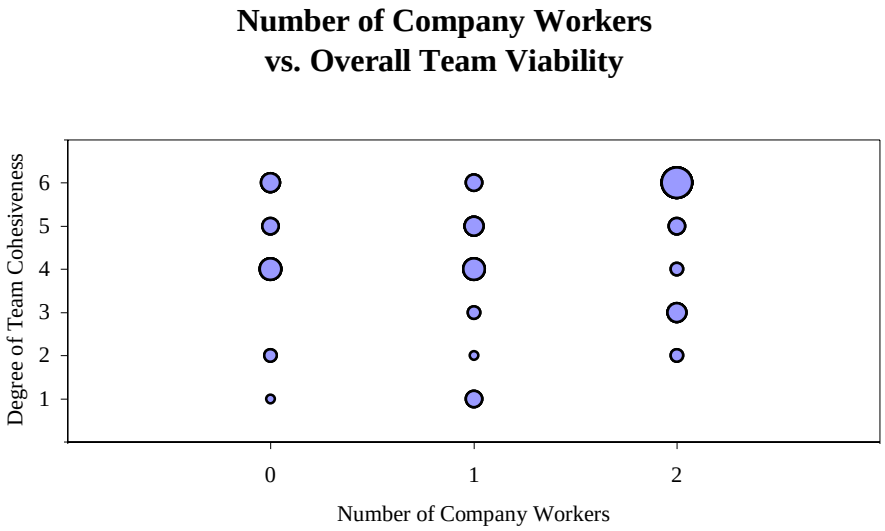


Figure 4.32: Number of Company Workers vs. Overall Team Viability

K) Experiment 9: Other Miscellaneous Tests and Results

The viability questionnaire given to participants after each phase required students to rate themselves and their teammates on a sliding scale for each of five questions. Table 4.3 lists these specific questions. Figure 4.33 shows an example of the sliding scale used. (The complete questionnaire can be seen in Appendix C.)

Research has shown that, generally speaking, people cannot estimate on an absolute scale how they feel about a given topic or question, but they are capable of making accurate side-by-side comparisons [ISA96]. For this reason, before analysis was attempted, a transformation on the sliding scale responses was developed. Six points were divided among the three teammates for each response. In simple cases, where each teammate was ranked differently on the scale, the highest ranked teammate was given a three, the lowest ranked teammate was given a one, and the remaining teammate was given a two. In the case of a three-way tie, all three teammates were given a two. In two-way tie cases, the

non-tied teammate received the score he or she would normally receive (either a one or a three.) The remaining points from the original six were split equally between the tied teammates. For the sake of a more concrete example, teammates ranked {6, 1, and 4} respectively would receive transformed scores of {3, 1, and 2 (which sums to six)}. Teammates ranked {3, 5, and 3} respectively would receive transformed scores of {1.5, 3, and 1.5 (which again sums to six)}.

Table 4.3 Questions included in the viability questionnaire

Q1	To what extent did each person exhibit *leadership* characteristics in your group?
Q2	Where did the *ideas* for your solution come from (which person/people)?
Q3	Who kept the team "*on-track*", (managed time or other resources)?
Q4	How much was each person *involved*?
Q5	For your team, did each person work more as an *individual* or more together with the team?

Once this transformation was complete, four distinct analyses were performed. A study was conducted investigating how a participant ranked himself, as compared with his original Belbin results. A separate analysis investigated how a person was perceived by his teammates on the five viability questions, compared with his original Belbin results. The next division relates to how the Belbin results are measured. Absolute raw score on each of the Belbin role scales could be used (which roughly matches the methods used by Stevens), or primary role for each person could be used as an absolute score by itself. Each of the two Belbin measures were used to conduct each of the two types of investigations (self-rating or teammate-rating).

1) To what extent did each person exhibit ***leadership*** characteristics in your group?

	NOT AT ALL……VERY MUCH					
You	**1**	**2**	**3**	**4**	**5**	**6**
Teammate A	**1**	**2**	**3**	**4**	**5**	**6**
Teammate B	**1**	**2**	**3**	**4**	**5**	**6**

Figure 4.33 A sample sliding scale question from the viability questionnaire

For the majority of cases, no significant correlations were found. Sliding scales were compared against each Belbin role score using Pearson's Product-

Moment Correlation [SAS90], via **PROC CORR** in ***SAS.*** When participants' self-rating was studied, the only significant correlation was somewhat weak. The Monitor-Evaluator role showed a positive correlation with question 2, which asks from whom the ideas on the team came (**r** = .44, **p** = .02). In other words, stronger Monitor-Evaluators had a (weak) tendency to consider themselves significant sources of ideas for their teams.

The same question (question 2) was, ironically, the only one whose results approached significance when self-ratings were compared against primary role. In this analysis, with respect to the adjusted sliding scales, scores of two were thrown out of the sample. Scores greater than two (suggesting either a "strongest" on the team for that question or a tie for the strongest) were grouped together. Similarly, scores less than two (suggesting either a "weakest" on the team or a tie for the weakest) were grouped together. A table was generated, and Fisher's Exact test [SAS90] was used to establish whether primary role was related to the sliding scale responses. For question 2, source of ideas for a team was found to be dependent on team role, albeit with weak confidence (**p** = .10). Five out of six Monitor-Evaluators rated themselves as the primary idea-generators for their teams, as compared with exactly 50% of the study sample as a whole. Using this method, results for none of the other questions approached significance.

When teammates estimated each others' contributions (through the five questions), little could be concluded with respect to statistical significance. A few interesting results were seen, however. For question 2, using the Fisher's Exact method [SAS90], a relation between role and source of ideas was found with significance within 10% (**p** = .06). Completer-Finishers were least likely to supply ideas (our out of four primary Completer-Finishers were ranked poorest by their teammates). Chairmen were most likely to supply ideas (four out of five primary Chairmen were ranked best by their teammates). This is, admittedly, a very small segment of the overall sample, but these results are matched by correlation by Pearson's method: a weak positive correlation between Chairman

score and source of ideas (as determined by teammates) was found (**r** = .42, **p** = .02). Completer-Finishers did not show a significant correlation, but they *did* represent the only negative correlation found (**r** = -.20, **p** = .23). On an interesting note, the role normally associated with ideas (the Plant) did not show significant results in any study relating to ideas.

The third question dealt with which team member(s) kept the team on-track. When individual absolute role scores were correlated against adjusted sliding-scale questionnaire values for this question, the only significant correlation was with respect to the Monitor-Evaluator scale, in which a very weak positive correlation was found (**r** = .28, **p** = .03). This correlation is actually weak enough to be virtually ignored, but was the only one to reach or even approximate significance. When Fisher's Exact method was used in this case, a significant relation was revealed between a person's team role and his likelihood to be perceived by his teammates as keeping the team on-track. Approximately 39% of the study sample was noted as the primary force for keeping the team on-track. This contrasts with 67% of Monitor-Evaluators (6 out of 9), 18% of Company Workers (3 out of 17), and 64% of Shapers (9 out of 14) (**p** = .02). This difference between these roles reinforces the notion of the Shaper as the leader or driving force, and the Monitor Evaluator as the decision maker (noticing and correcting problems before they become severe). These findings may also reflect the earlier observation that the Company Worker makes the team less predictable (see Experiment 1 in this chapter).

The fourth question in the survey addressed the extent to which each person was involved on the team. This question produced perhaps the most interesting and significant results of all. When Pearson's Product-Moment Correlation [SAS90] was used to relate individual, raw role scores with adjusted sliding-scale values for this question, a moderate, positive correlation between the Chairman role score and "involvement" on the team was found (**r** = .58, **p** < .01). A very weak negative correlation (significance within 10%) was found between

the Completer-Finisher role score and "involvement" on the team (**r** = -.28, **p** = .08). These results were confirmed using the Fisher's Exact method. In the sample as a whole, 55% of participants were identified as being primarily "involved" on the team. (It is important to remember that in the case of a two-way first-place tie, both first place members are noted here.) This is contrasted with 100% of Chairmen (3 out of 3), 0% of Completer-Finishers (0 out of 4), and 100% of Plants (6 out of 6) (**p** < .01). The Chairman is generally not typified as a technical type, and seems to bow out to the Shaper in terms of leadership on a software engineering team, but these results suggest that the Chairman is far from being uninvolved. The Chairman obviously makes a real contribution, although it remains to be determined what this contribution is. Experiments 4-6 in this chapter suggest that this role may be drawn to debugging, but peer evaluation does not support this notion. The Completer-Finisher seems to be of little value to a team, as results suggest that this role is relatively uninvolved in team activity. Every analysis in Experiments 4-6 of this Chapter puts his involvement in each of the software engineering phases well below the sample average. However, this cannot be taken as a definitive conclusion, because 1) this study was not aimed at Completer-Finishers and 2) Completer Finishers are not strongly represented in this sample.

Questions 1 and 5 did not produce any interesting or significant results. These questions dealt with leadership on the team and teamwork (effort as an individual as opposed to a part of a team), respectively. Most participants mentioned in free-form statements on the questionnaire that their teams functioned smoothly, explaining the lack of significant results for question 5. Perhaps this suggests that other factors (professional attitude, skill, etc.) are more important in determining team cohesiveness than are team roles.

Chapter V: Experiments with the Belbin SPI and the Keirsey Temperament Sorter

Previous work in this area involved overlapping study of the Belbin Self-Perception Inventory and the Keirsey Temperament Sorter. The Keirsey Temperament Sorter, like the more popular Myers-Briggs Type Indicator (MBTI), measures four facets of personality, with each measure independent of the others. In fact, the Keirsey test was based on the MBTI, and the format of the results of the two tools is identical [KEID84] [STET98]. Furthermore, Stevens demonstrated fairly strong statistical correlations between these two instruments [STET98].

The four personality facets of the Keirsey Temperament Sorter and the MBTI are Introversion-Extroversion (I-E), Intuition-Sensing (N-S), Thinking-Feeling (T-F), and Perceiving-Judging (P-J). A person's personality, or temperament, is then described using four letters, each of which represents polarity on one of the four scales. An *Introverted Sensing Feeling Judging* person, for example, would be labeled as an *ISFJ*. It is possible for a person to measure neutrally on one or more scales. When this happens, an "X" is substituted for the scale for which no preference is determined. In the experiments conducted for this study, the four scales were used with each of 102 participants. Neutral scale measurements only occurred 28 out of 408 times (less than 7% of all scale measurements). No subjects were ranked neutrally in more than two of the Keirsey scales.

The Keirsey and MBTI tests have some roots in Jungian psychology and Hippocratic temperament theory [BRAK64] [JUNC23] [MYEI80] [KEID84], and also have common points with widely accepted modern tools like the 16PF [CATR70] and the Big Five [COSP92]. All of these tools study the areas of Introversion / Extroversion, for example. The work of Stevens and Henry

attempted to draw a mapping between the Belbin roles and Keirsey temperaments, but this study suffered from a small sample [STET98].

The importance of such a mapping to this and similar work becomes apparent when one considers that complex, involved studies of software engineering team projects, using Belbin roles as a basis, take a very long time. It is a relatively quick and simple process, however, to merely administer both the Keirsey Temperament Sorter and the Belbin SPI to a large sample size, and to determine the relationship between the two. It may be helpful to one day generalize and restructure the Belbin results in terms of the Keirsey scales. This could offer the potential to:

1) Approximate or loosely predict a person's Belbin role, given their Keirsey type
2) Discover which attributes of a team role are most firmly entrenched in the type (if high correlations between some Belbin roles and Keirsey scales exist)
3) Determine the degree of structural commonality between the two tests
4) Speculate on the value of Belbin's role theory, using Keirsey as a measure.

In previous work at Virginia Tech, the Keirsey Temperament Sorter was chosen instead of the MBTI for five main reasons, as detailed by Stevens regarding his research:

1) Administration - the MBTI can be administered only by a trained professional, which due to availability of administrators, and cost to a minor degree, adds great complexity to data gathering.
2) Distribution - Some of the participants are located around the country; collecting the data would not have been possible for many of the participants. This also added to the amount of data collected because of the ease to the participants of providing the data.
3) Recommendation - the Keirsey test was recommended by the Director of the Counseling Center at Virginia Tech, Dr. Warren, as a simple, equivalent replacement for the MBTI.
4) Consistency - although the participants for the controlled study described in Chapter 3 are all in one location, some of the participants for the other studies described in this section are distributed around the country, therefore some of the data needed to be collected using the Keirsey test and consistency among the

data sets for this investigation necessitated that all data use the same test.

5) Cost - the MBTI costs money, whereas the Keirsey test is free. [STET98]

This current study adds the benefit of consistency with previous work to the list of reasons for using the Keirsey Temperament Sorter.

A) The Keirsey Temperaments in the Computer Science Community

Stevens' work sought, as a peripheral study, to determine and describe the computer science population using the Keirsey temperaments. This population breakdown was then compared and contrasted with the makeup of the general populace. This study has merit if only in the notion that computer scientists are, as a group, different than other people. Stevens' study and related work have demonstrated that Belbin's role theory may apply differently to business management teams (for which it was designed) and to software development teams. Is this because of the different nature of the *work* or the different nature of the *people* – or both? (Perhaps the work attracts a different type of people.)

Prior research on the general population had determined that roughly three-fourths of the population as a whole tends toward Extroversion as opposed to Introversion, and three-fourths tends toward Sensing as opposed to Intuition. On the other two scales (Feeling/Thinking and Preceiving/Judging), the general population is roughly equally split [KEID84] [BRA64].

Table 5.1 shows the demographic breakdown of Keirsey scales, as reported by various studies, of the computer science community and the general population. Numbers shown are percents of the overall sample population. Ties for scales are not considered in these percents. Percents from the general population are as reported by [KEID84] and [BRA64]. Lyons' study of 1229

computer professionals demonstrates a significant deviation from the norm in of the scales [LYOM85]. (Lyons used the Myers-Briggs Type Indicator instead of the Keirsey Temperament Sorter.) Stevens' study of software engineers [STET98] shows a profile similar to the current study, but the numbers shown are estimated from graphical displays in his dissertation. (No actual numbers were reported.) The current study involves 102 junior-level and senior-level computer science students at Virginia Tech.

Table 5.1: Keirsey scale profiles of the general population and computer science samples.

	I	E	N	S	F	T	P	J
General population	25	75	25	75	50	50	50	50
Lyons (MBTI)	67	33	54	46	19	81	--	--
Stevens' study	~60	~40	~44	~56	~38	~62	~26	~74
Current study	64	36	35	65	39	61	28	72

Stevens concluded that a noteworthy difference in Belbin role distribution existed between the general population and the computer science population. This argument certainly seems reasonable in view of the data. Stevens' numbers also seem valid, given the strength of agreement between his study and the current one.

A few noticeable differences between the Lyons study and the Stevens and current studies can be seen. Lyons notes a somewhat larger percentage of Intuitives and Thinkers than do Stevens and this study. This may be attributed to two factors. First, Lyons used the Myers-Briggs Type Indicator (MBTI) [MYEI80] as opposed to the Keirsey Temperament Sorter, and while a substantial correlation has been established between the two, they are still different tests. Secondly, Lyons studied computer scientists (more specifically, data processing professionals), whereas the current study focuses on students. It must be noted that there is virtue in studying students, as they bear much similarity to real-world professionals [HOLR87], but it must be conceded that differences also exist. To wit: Stevens' study of the profile of Belbin roles in computer science showed significantly more Team Workers in professional settings than in academic

settings. [STET98] The Team Worker characteristically is the oil that makes a team work together [BELR81]. The Team Worker resolves disputes, acts as a peacemaker, and sees to the general smooth operation of the team. Perhaps this role is less often filled in student studies because the perceived need for the role does not become apparent until the professional setting. This may explain some of the difference in the Lyons study and the current one.

B) Mapping Belbin and Keirsey

In order to study the prospective trends between the Belbin SPI and the Keirsey, the primary Belbin team role for each participant was compared against the participant's polarized score for each of the four Keirsey scales to allow for the detection of the presence of a relation between team role and each individual Keirsey scale. For example: do team roles differ in their preferences with respect to the Introversion/Extroversion scale, or are all roles essentially the same in this respect? If a Monitor-Evaluator is significantly more likely to be Introverted than is a Shaper (for example), this difference in roles should be confirmed.

All four Keirsey scales (Introversion/Extroversion, Intuition/Sensing – the mode used for information gathering, Thinking/Feeling – the mode used for information processing, and Judging/Perceiving – the preference for either *making decisions* or *continuing to gather information*) were analyzed with regard to Belbin team roles. This analysis used Fisher's EXACT test in ***SAS*** with **PROC FREQ** to establish whether any significant relationship existed between Belbin roles and Keirsey scales. Statistical significance in this case suggests that the categories being compared (for example, *team roles* and *Introversion-Extroversion*) cannot be assumed to be independent.

One important reason for this study is that earlier mapping attempts in this domain suffered from a very poor sample size (**N** = 23), and two from that sample

were tie cases: analysis of primary role showed two participants each to be scored equally for their two top-ranked roles. This current study combined the results of Stevens' earlier study and the test results of two senior-level software engineering classes and two junior-level operating systems classes at Virginia Tech. Tie cases for primary team role were then thrown out of the sample, bringing the final study sample size to (**N** = 102). This greatly improves on the reliability of the original test.

Three of the four scales reached or approached statistical significance at the 10% level. None reached significance at the 5% level, although the Judging-Perceiving scale came quite close. Data-level results and percentage breakdowns can be seen in Tables 5.2 – 5.4. Each of the next four sections address a single Keirsey scale, and refer to Tables 5.2 – 5.4.

Table 5.2 shows the number of subjects in each role, with Keirsey scale divisions for each role (including tie-score/neutral cases, which are noted in the "X" column for each scale). The computed "expected value" is shown in parenthesis, and indicates the count that would be appear for that cell if all roles were identical with respect to the Keirsey scales. Large discrepancies between the actual count in a cell and the expected value might suggest areas where a role may be significantly different from the sample as a whole. Table 5.3 shows the percent of each role drawn to each Keirsey scale pole. Tie cases are noted in the "X" columns. Numbers in parenthesis reflect the number of samples counted in that category. These are identical to those found in Table 5.2, and are included in Table 5.3 for the sake of context: larger sample sizes are likely to yield more trustworthy percentages. Table 5.4 repeats this process, but all percents are computed after tie-cases for each scale are removed from the sample (**N** = 102). In these next four sections, results from this sample may suggest relationships between Belbin roles and Keirsey scales that reflect outside generalizations, but they do not attempt to propose definitive relationships outside the sample.

```
PROC FREQ DATA = BEL_KEIR;
WHERE BELBIN_TIE_EXISTS = 0;
TABLE PRIM_ROLE * ABSOLUTE_IE/EXACT EXPECTED;
RUN;
```

Figure 5.1: SAS code to determine dependence of Keirsey scales on Belbin roles.

Table 5.2: Breakdown of sample, showing how roles polarized in Keirsey scales, with expected values

Role	E	I	X	N	S	X	F	T	X	J	P	X
CH	2 (1.3)	2 (2.3)	0 (0.5)	2 (1.3)	2 (2.4)	0 (0.2)	1 (1.5)	1 (2.3)	2 (0.2)	0 (2.7)	4 (1.1)	0 (0.2)
SH	10 (7.5)	9 (13.6)	5 (2.8)	6 (8.0)	15 (14.6)	3 (1.4)	8 (8.9)	16 (13.9)	0 (1.2)	19 (16.5)	3 (6.4)	2 (1.2)
PL	5 (5.3)	10 (9.7)	2 (2.0)	11 (5.7)	5 (10.3)	1 (1.0)	5 (6.3)	11 (9.8)	1 (0.8)	9 (11.7)	8 (4.5)	0 (0.8)
RI	2 (0.9)	1 (1.7)	0 (0.4)	2 (1.0)	1 (1.8)	0 (0.2)	3 (1.1)	0 (1.7)	0 (0.1)	2 (2.1)	1 (0.8)	0 (0.1)
ME	1 (2.8)	7 (5.1)	1 (1.1)	1 (3.0)	8 (5.5)	0 (0.5)	4 (3.4)	5 (5.2)	0 (0.4)	7 (6.2)	1 (2.4)	1 (0.4)
CW	10 (9.4)	18 (17.1)	2 (3.5)	9 (10.0)	20 (18.2)	1 (1.8)	14 (11.2)	15 (17.4)	1 (1.5)	21 (20.6)	7 (7.9)	2 (1.5)
TW	1 (2.2)	4 (4.0)	2 (0.8)	2 (2.3)	4 (4.3)	1 (0.4)	1 (2.6)	6 (4.0)	0 (0.3)	5 (4.8)	2 (1.9)	0 (0.3)
CF	1 (2.5)	7 (4.5)	0 (0.9)	1 (2.7)	7 (4.9)	0 (0.5)	2 (3.0)	5 (4.6)	1 (0.4)	7 (5.5)	1 (2.1)	0 (0.4)
total	**32**	**58**	**12**	**34**	**62**	**6**	**38**	**59**	**5**	**70**	**27**	**5**

1) The Introversion – Extroversion Scale

With respect to the Introversion-Extroversion scale, almost twice as many in this sample were determined to be Introverts than Extroverts, although a noticeably large number of subjects rated neutrally on this scale. No role deviated substantially from the overall trend in this regard (**p** = .40). This in itself is somewhat surprising, as Belbin specifically describes some roles as quite Introverted or Extroverted. The Monitor-Evaluator, for example, should rank disproportionately on the Introverted side [BELR81]. The sample certainly shows a tendency for the software engineer Monitor-Evaluator to lean toward Introversion, but this tendency is not significantly different from the tendencies of this software engineer sample as a whole. Perhaps for computer scientists, this distinction between the team roles is made impotent.

2) The Intuitive – Sensing Scale

The distinction between roles as checked against the Intuitive-Sensing scale almost reached significance at the 10% level (**p** = .11). Two sets of roles are of particular interest here. First, the Plant and Resource Investigator are noteworthy because the trend they seem to display regarding this scale is perfectly contrary to the trend exhibited by all the other roles, and by the overall sample average. According to this sample, software engineers seem to be drawn to the Sensing pole about 61% of the time. Plants, however, were drawn 65% of the time to the Intuitive end of the scale, as were Resource Investigators 67% of the time (see Table 5.3). The Resource Investigator results here are unreliable because of their extremely poor representation in the sample (three Resource Investigators), but are still interesting because of the theoretical tie with the Plant. Another way of illustrating the lack of confidence in results for the Resource Investigator is to examine expected results as contrasted with observed results. While analysis predicted 5.7 intuitive Plants (as opposed to eleven observed, for a difference of over five), the same analysis predicted one intuitive Resource Investigator (as opposed to two observed, for a difference of only one). The Plant and Resource Investigator are the two innovation roles, according to Belbin [BELR81]. It is quite interesting that they have a tendency to let intuition guide their information gathering processes, unlike other computer scientists, who seem to have a tendency to prefer straightforward facts as gathered by the senses.

The other set of interest involves the Shaper, Monitor-Evaluator, and Completer-Finisher. This set of roles is of particular interest for two reasons. First, these roles seem to group together both in this scale (*N-S*) and again in the *J-P* scale. In this case, whereas the overall sample tended ~61% of the time toward the Sensing pole, Completer-Finishers tended 88% of the time in that direction, Monitor-Evaluators 89% of the time, and Shapers 63% of the time. Notice here that Company Workers are more often polarized in this sample toward Sensing than are Shapers, but the Shaper group also had several ties in this scale (neutral measurements); the result being that these Shapers are also less

frequently noted as being Intuitive than are the Company Workers. With that taken into account, this grouping of roles can be picked out readily again in the J-P scale.

3) The Feeling – Thinking Scale

The Feeling-Thinking scale offers two interesting observations of the sample. This scale is shown, within a 10% level of significance (**p** = .08), not to be independent of team role. The sample in general is polarized (although somewhat weakly) toward the Thinking side, however, two team role extremes stand out from the others. Team Workers in the study were drawn approximately 87% of the time toward the Thinking pole, as compared with 57% of the time for the sample in general. In role theory, the Team Worker is a sort of "peacemaker" role: the "grease" that helps the rest of the team to operate smoothly. The Team Worker's link here to an unusually high predisposition to Thinking (rather than Feeling) presents a puzzling concept. Given the lack of theoretical underpinnings for this observation and the relatively small representation in the sample, this may simply be a random occurrence. Team Workers account for less than 7% of the sample.

The other extreme in the Feeling-Thinking scale is more interesting. The Resource Investigator role, which generally seems not to occur in the field of computer science, is represented by less than 3% of this sample. Not only is the Resource Investigator role the only one in this sample to polarize toward the Feeling side of the *F-T* scale, but 100% of the Resource Investigators were drawn to that side of the scale. (In the entire sample, this only occurred in one other case, in the *J-P* scale.) Even though the role is poorly represented, this is of particular interest. The trend certainly matches theory: the Resource Investigator, like the Plant, is a team innovator. Unlike the Plant, however, the Resource Investigator brings in ideas, talent, and new directions and concepts from outside the team, rather than by any actual personal synthesis of ideas. The Resource

Investigator is drawn toward working with others. This nature, paired with an inclination toward Feeling, might make him quite useful in the parts of computer science not normally associated with software development *per se*, such as requirements engineering and usability engineering. Both require work with others and could benefit from the sixth sense which the Feeling aspect may provide. (Certainly these areas need the benefits offered by a Thinking modality, but data-heavy methodical techniques, such as formative evaluations, may provide the needed grounding in such cases.)

4) The Judging – Perceiving Scale

The Judging-Perceiving scale offered the most statistically significant results, with confidence very near the 95% level (**p** = .06). Again, two interesting trends could be observed in the sample. First, the trio of roles noted in analysis of the *N-S* scale appears grouped again here. This trio includes the Shaper, Monitor-Evaluator, and Completer-Finisher. All three are especially drawn here to the Judging pole, with average tendencies of 79%, 78%, and 88% respectively, compared to an overall average of 69%. Also note that a few Shapers and Monitor-Evaluators scored on this scale, the result being that while 26% (on average) of the sample was drawn to the Perceiving side, only 13% and 11% of Shapers and Monitor-Evaluators were so predisposed. These three roles are characterized by a high degree of focus, and are grouped both here and in the N-S scale. Could it be the Sensing/Judging combination that instills this degree of focus in these roles? While the answer is beyond the scope of this work, the question still affords mention.

Also of interest here is the Chairman role. In this study, the Chairman distinguished himself in all three of the other scales by demonstrating no predisposition at all! With respect to the Introversion-Extroversion scale, the Intuitive-Sensing scale, and the Feeling-Thinking scale, the Chairman was evenly split in every case. (In the Feeling-Thinking scale, half of the Chairmen even

scored neutrally.) In this scale (Judging-Perceiving), however, 100% of Chairmen were drawn to the Perception pole, as compared with 26% of the sample as a whole. Perhaps this is the only scale defining a Chairman. This analysis must be tempered with an understanding that there were only four primary Chairmen out of a sample of 102. These results, therefore, are interesting, but suspect. A more important assertion, though, would be that this degree of neutrality is the reason that Chairmen are generally able to lead without conflict in management teams. Also, the predisposition to perception instead of judgment may be the reason that Chairmen are generally not predisposed to leadership in technical domains. Very interesting is the fact that, regarding the two leadership roles, Shapers make up six times more subjects in the study sample than do Chairmen.

Table 5.3: Percentage of each role drawn to each Keirsey scale pole, including neutral cases/ties (X)

Role	E	I	X	N	S	X	F	T	X	J	P	X	All
CH	50.00% (2)	50.00% (2)	0.00% (0)	50.00% (2)	50.00% (2)	0.00% (0)	25.00% (1)	25.00% (1)	50.00% (2)	0.00% (0)	100.00% (4)	0.00% (0)	3.92% (4)
SH	41.67% (10)	37.50% (9)	20.83% (5)	25.00% (6)	62.50% (15)	12.50% (3)	33.33% (8)	66.67% (16)	0.00% (0)	79.17% (19)	12.50% (3)	8.33% (2)	23.53% (24)
PL	29.41% (5)	58.82% (10)	11.76% (2)	64.71% (11)	29.41% (5)	5.88% (1)	29.41% (5)	64.71% (11)	5.88% (1)	52.94% (9)	47.06% (8)	0.00% (0)	16.67% (17)
RI	66.67% (2)	33.33% (1)	0.00% (0)	66.67% (2)	33.33% (1)	0.00% (0)	100.00% (3)	0.00% (0)	0.00% (0)	66.67% (2)	33.33% (1)	0.00% (0)	2.94% (3)
ME	11.11% (1)	77.78% (7)	11.11% (1)	11.11% (1)	88.89% (8)	0.00% (0)	44.44% (4)	55.56% (5)	0.00% (0)	77.78% (7)	11.11% (1)	11.11% (1)	8.82% (9)
CW	33.33% (10)	60.00% (18)	6.67% (2)	30.00% (9)	66.67% (20)	3.33% (1)	46.67% (14)	50.00% (15)	3.33% (1)	70.00% (21)	23.33% (7)	6.67% (2)	29.41% (30)
TW	14.29% (1)	57.14% (4)	28.57% (2)	28.57% (2)	57.14% (4)	14.29% (1)	14.29% (1)	85.71% (6)	0.00% (0)	71.42% (5)	28.57% (2)	0.00% (0)	6.86% (7)
CF	12.50% (1)	87.50% (7)	0.00% (0)	12.50% (1)	87.50% (7)	0.00% (0)	25.00% (2)	62.50% (5)	12.50% (1)	87.50% (7)	12.50% (1)	0.00% (0)	7.84% (8)
	31.37%	**56.86%**	**11.76%**	**33.33%**	**60.78%**	**5.88%**	**37.25%**	**57.84%**	**4.90%**	**68.63%**	**26.47%**	**4.90%**	

Table 5.4: Percentage of each role drawn to each Keirsey scale pole, ignoring neutral cases/ties

Role	E	I	N	S	F	T	J	P
CH	50.0	50.0	50.0	50.0	50.0	50.0	0.0	100.0
SH	52.6	47.4	28.6	71.4	33.3	66.7	86.3	13.6
PL	33.3	66.7	68.8	31.3	31.3	68.8	52.9	47.1
RI	66.7	33.3	66.7	33.3	100.0	0.0	66.7	33.3
ME	12.5	87.5	11.1	88.9	44.4	55.6	87.5	12.5
CW	35.7	64.3	31.0	69.0	48.3	51.7	75.0	25.0
TW	20.0	80.0	33.3	66.7	14.3	85.7	71.4	28.6
CF	12.5	87.5	12.5	87.5	28.6	71.4	87.5	12.5
Ave.	**35.6**	**64.4**	**35.4**	**64.6**	**39.2**	**60.8**	**72.2**	**27.8**

C) Relative Keirsey Trends for Each Team Role

Perhaps it is even plausible at this point to suggest a loose mapping of roles. Two such mappings should be done: one using the 50% point as a basis for deciding tendencies (so that individual tendencies can be measured with the Keirsey test), and another which compares individual tendencies against the average tendencies for the sample. This may allow one to make more reasonable guesses, along the lines of, "John seems to be even more of an introvert than most software engineers I've met, so perhaps he is a Monitor Evaluator or Completer-Finisher." In each case, an "x" is noted for a role if the role's deviation from the established midpoint is at most 10%, viz., if the midpoint is 50%, then role averages between 40% and 60% would score neutrally. Between 10% and 25%, a lower-case letter representing the appropriate scale pole denotes role tendencies. Deviations greater than 25% are noted with a capital letter. For purposes of these mappings, neutral rankings were ignored.

Tables 5.5 and 5.6 show Belbin role trends for the four Keirsey scales. In Table 5.5, the Keirsey test itself is the basis for determining relative tendency for each scale. (This is the way the Keirsey Temperament Sorter is normally scored.) An "X" represents no tendency. A small capital letter represents a weak polar

tendency. Large, bold, italic letters represent strong polar tendencies. In Table 5.6, the computer science study sample for this current research is the basis for determining relative tendency for each scale. For each scale, the computer science sample average was determined, and used as a standard for judging individual role tendencies. An "X" here represents no tendency. A small capital letter represents a weak polar tendency. Large, bold, italic letters represent strong polar tendencies.

Table 5.5 Role trends for the Keirsey scales, using the Keirsey test as a standard measure

CH	XXX***P***
SH	XST***J***
PL	INTJ
RI	EN***F***J

ME	***IS***X***J***
CW	ISXJ
TW	IS***T***J
CF	***IS***T***J***

Table 5.6 Role trends for the Keirsey scales, using sample average as a standard measure

CH	ENF***P***
SH	***E***XXJ
PL	X***N***XP
RI	***ENF***X

ME	ISXJ
CW	XXXX
TW	IXTX
CF	ISTJ

The second of these charts presents a final insight: the "average" Company Worker, (if such exists) is not terribly different from the "average" computer scientist (if such exists). As compared to the general study sample, very little deviation in Keirsey scale pole preference trends could be found in the Company Worker team role. This in itself makes the Company Worker unique, and perhaps this lack of specific polarity of Keirsey scales is responsible for some of the variance that the Company Worker was observed to add to team success. This seems, at very least, a plausible conjecture: if the Company Worker's (Keirsey) temperament, as compared to the general population, is unpredictable,

then it seems reasonable that the effect he has on a team's success is also unpredictable.

Table 5.7: Stevens' Mapping between Belbin team role and Keirsey scales.

	E/I	S/N	T/F	J/P
CH	-	+0.2102 .0443	-	-
SH	+0.2474 .0174	-	-	-
PL	-	-0.4205 .0001	+0.1731 .0990	-0.4323 .0001
RI	+0.2915 .0048	-	-	-0.2462 .0180
ME	-	-	-	-
CW	-	+0.2280 .0288	-	+0.2763 .0077
TW	-	-	-	-
CF	-	+0.2122 .0423	-	+0.3757 .0002

Stevens' mapping scheme between Belbin roles and Keirsey scales is pictured in Table 5.7. Non-empty cells represent points at which "significant" correlations were found. The top number in a cell represents the correlation factor (**r**), and the bottom number represents the confidence (**p**). Positive poles for the four scales are E, S, T, and J, respectively. In this table, "CH" denotes Chairmen, "SH" denotes Shapers, "PL" denotes Plants, "RI" denotes Resource Investigators, "ME" denotes Monitor- Evaluators, "CW" denotes Company Workers, "TW" denotes Team Workers, and "CF" denotes Completer-Finishers [STET98].

Stevens' results were compared with the results of the current study. A few non-obvious points should be considered when comparing the data. First, Stevens' analysis was done using Pearson's Product-Moment Correlation [SAS90]. Each run of this technique results in a correlation between a single Belbin scale and a single Keirsey scale. Generalities along the lines of, "As the Plant team role dimension score increases, the Perceiving scale tends to increase." Since only computer scientists were included in the study sample, the tendencies

can be generalized only within the computer science domain. For this reason, it makes more sense to compare Stevens' results against the current results, in as much as they show role tendencies within the context of the software engineering sample as a whole. In other words, rather than simply stating that the Completer-Finisher is drawn to the Judging side of the *P-J* scale, it is more relevant to note that he is more strongly drawn to Judging than is the average software engineer.

With this in mind, a side-by-side comparison of Stevens' results and the current results is shown (Table 5.8). In this table, boldface type represents tendencies found within the context of a computer science domain sample. Italics represent tendencies found simply within the context of the Keirsey test itself. Under the "Agreement Points" column, italics and boldface type distinguish whether Stevens' results agree with the test-context current results, the computer science-context current results, or both.

The current study in computer science context seems to agree with Stevens' study almost half the time, and in many of the non-agreement points, no trend at all was detected in one or the other of the studies. The comparison of these two experiments can be seen in Table 5.8. Italics here denote mappings that suggest a role's tendency relative to the Keirsey test as a metric. Boldface type denotes Kiersey scale tendencies relative to the software engineering population sample. The far right column notes areas of agreement between the studies.

Table 5.8: Stevens' Belbin-Keirsey mapping results compared with the results of two current study methods

	Current Study (test context)	Current Study (CS context)	Stevens' Study (CS context)	Agreement points
CH	*XXXP*	**ENFP**	XSXX	*X-X-*
SH	*XSTJ*	**EXXJ**	EXXX	**EXX-**
PL	*INTJ*	**XNXP**	XNTP	**X*N*TP**
RI	*ENFJ*	**ENFX**	EXXP	***E*---**
ME	*ISXJ*	**ISXJ**	XXXX	**--X-**
CW	*ISXJ*	**XXXX**	XSXJ	**X*S*X*J***
TW	*ISTJ*	**IXTX**	XXXX	**-X-X**
CF	*ISTJ*	**ISTJ**	XSXJ	***-S-J***

D) The Company Worker and the Keirsey Scales

The last point worthy of exploration here is analysis of the trends of the Company Worker. Stevens work suggests that the Company Worker tends slightly toward the Sensing pole of the Intuitive – Sensing scale, and slightly toward the Judging pole of the Perceiving – Judging scale. This was also noted in the current analysis of the Company Worker in the context of the Keirsey Test itself. However, the software engineering sample as a whole also tended to be Sensing and Judging. If the Company Worker's tendencies are compared against the average tendencies of the sample as a whole, the Company Worker suddenly has no noticeable trend in *any direction* on *any of the four scales*. In other words, the Company Worker is equally likely to be Introverted or Extroverted relative to the software engineering sample as a whole. He is equally likely to be Intuitive or Sensing relative to the software engineering sample. In the same sense, he is equally likely to be Thinking or Feeling, or Perceiving or Judging relative to other software engineers. In short, knowing that a software engineer is a Company Worker gives you no reliable information about their Keirsey tendencies that you could not deduce simply by knowing that he was a software engineer. The

significance of this insight is that again, Company Workers are seen to be unpredictable: a Company Worker on a software engineering team is equally likely to add the influence of any of the eight Keirsey poles to the team's interactions.

Chapter VI: Belbin Revisited

A) Test-Retest Validity of the Belbin SPI

The software engineering experiment teams used to explore team success as affected by the presence of Company Workers offered an additional benefit. Because previous samples of computer science students and professionals showed extraordinarily low representation for certain Belbin roles (most notably the Resource Investigator), this study sought to determine if these roles might emerge from need within the context of a specific team. If this were the case, it could be reasoned that such an emergence of roles demonstrates an innate need for these roles for software engineering teams. After each phase of four experiments (during which teams were unchanged), each subject was asked to take the Belbin SPI again. Although no emergent roles could be observed from this study, a more significant and important study was made possible by this retest method.

In general, correlations between the results of original administrations of the SPI and team-context retests were poorer than expected, leading to the realization that this could be evidence of problems internal to the SPI itself. Psychology calls this concept the "test-retest validity" of a survey. A test is generally deemed valid if significant correlations of at least .7 are observed in retesting.

With this in mind, comparisons were made between each person's original Belbin SPI results, the results of the retest after phase one, and the results of the retest after phase two. In all, each person took the test three times within a single four-month semester. If the test is to be trusted, correlations should be fairly high, especially considering the relatively small amount of elapsed time between the tests. Again, cases in which ties existed for primary roles were eliminated.

Table 6.1 Retest correlations for each role scale, using each retest pairing

Role	Original vs. Retest 1	Original vs. Retest 2	Retest 1 vs. Retest 2
CH	.81 *	.48	.28 (NS)
SH	.45	.36 (NS)	.67 *
PL	.31 (NS)	.42	.51
ME	.53	.76 *	.56
CW	.34 (NS)	.23 (NS)	.49
TW	.38	.46	.42
CF	.00 (NS)	.36 (NS)	.00 (NS)

Table 6.1 shows consistency correlations between tests for each team role scale. In this data, the Resource Investigator results are not given, because of the role's low representation in the sample. In this Table, (NS) stands for "Not Significant", and means that the given correlation does not reach the 5% significance level. Correlations which are significant and which reach or approximate correlation levels indicating possible validity (.7 or greater) are noted with an asterisk (*). For roles in which a correlation meets or nearly meets one test-retest validity requirement, the role's test-retest validity is not consistent among all three combinations of tests.

This does not in itself speak well of Belbin role theory or the Self-Perception Inventory as published in Belbin's 1981 book. A personality-based test in which scores a person differently three times in approximately as many months should be extremely suspect. For the sake of completeness and visualization, correlations were calculated between original and retest values for each individual Belbin scale. In this case, minute differences in test-retest results (enough to alter a person's measured primary role) would still show reasonably high correlations. Saville and Wilson's counter-assertion that ipsative and normative tests are equally valid may indeed stand true [SAVP91].

Figure 6.1 shows the results of this study. Several noteworthy, significant correlations can be observed. The X-axis for each role shows the score for that role on the participant's original Belbin test. The Y-axis represents the corresponding score on that participant's retest (both phase 1 and phase 2 retests are considered in this analysis). In cases where participants did not score strongly enough on a scale to even register an interpretive value, those participants were assigned a score of -1 for that scale. (The -1 value was chosen because the test registers existence of a role with a score of zero or more.) At the bottom left corner of each graph in the figure is a very large circle. This circle represents those who received the -1 score for both the original test and the retest. While inclusion of this information may seem odd, it can be justified by the notion that a valid test should not only be able to correlate well for the degree to which someone *is* a Plant, for example. It should also be able to consistently identify those who are *not* Plants at all. Along a similar vein, sample points along the axes represent cases where either a person was measured as a noticeable Plant (for example) in the original test but not in the retest, or vice-versa (was not measured as a noticeable Plant in the original test but *was* in the retest).

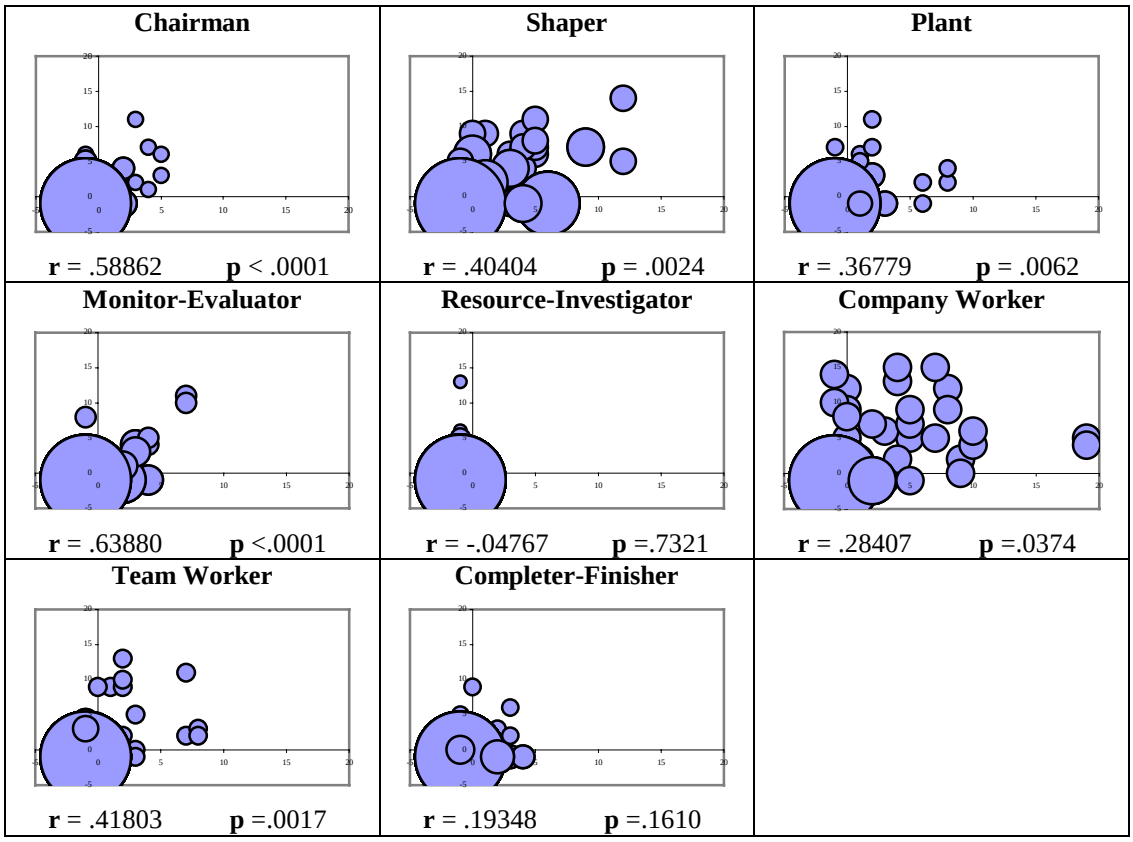


Figure 6.1: Test-retest correlations for each individual Belbin scale.

The results of the individual scale test-retest correlations, while generally not strong, fare somewhat better than the Primary-Role-only method from Tables 6.1 and 6.2. The results for the Shaper and Plant roles are particularly interesting, because of the high representation and relatively high correlation, respectively. Also of interest is the seemingly random scatter in the Company Worker plot. This may reinforce the notion of the Company Worker's lack of predictability.

In the interest of completeness, similar correlations were checked between the results of the participants' first retest and their second retest. The value in such a study is that if significantly higher correlations are found when checking a person's two team-context self-retests, this may suggest that the test should be taken only after the experience of working with *some* team, after which a person's role is less likely to change to a significant degree. It may also be that college-level team projects are too small and temporary to motivate a person to role

stability. Perhaps a test-retest validation study for the SPI would produce more positive results with subjects in an industry setting.

The analysis of retest-to-retest consistency (between a person's retest after experiment phase 1 and his retest after phase 2), when both retests are in *some* team context, proved to be no more or less significant than the original test-retest correlations. Results of the analysis are shown in Figure 6.2. When comparing this figure to 6.1, one can see that roughly half of the correlations increase and roughly half decrease. One should not conclude, therefore, that taking the test in team context adds any real benefit to the test results. Scales and formats in this Figure 6.2 are identical to those in Figure 6.1.

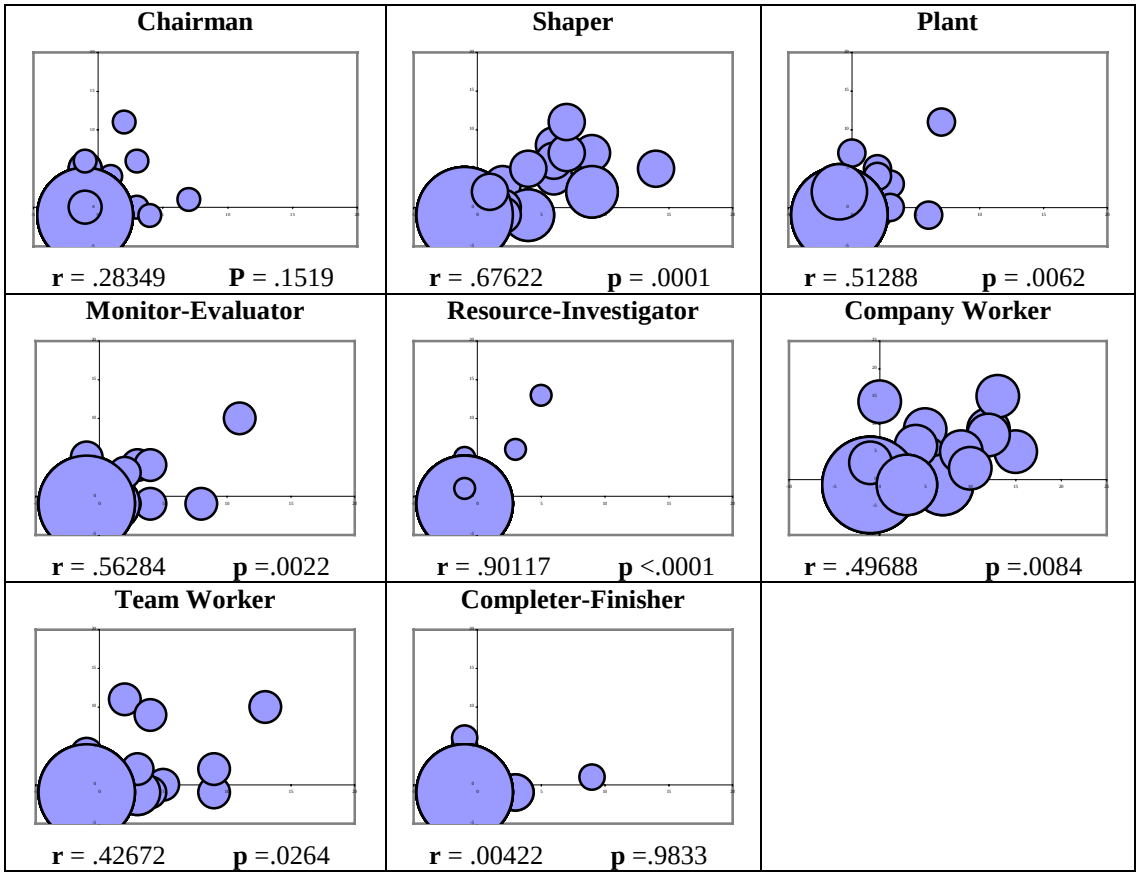


Figure 6.2: First retest – second retest correlations for each individual Belbin scale

B) Emergence of Roles

Participants in the Chapter 4 experiments were also asked to retake the Belbin SPI after each phase of experiments. It was speculated that this approach might provide information to address a hole in the logic of the Stevens study by comparing each subject's original SPI results with retests after the subject had worked in a team context.

Stevens' work seems to suggest that because some team roles generally do not exist within a software engineering team, those roles are not useful to such a team. While there is a certain degree of reasonableness to this logic, it is nonetheless flawed. In fact, the near non-existence of a certain role within the software engineering community may identify a fundamental weakness in the industry, a weakness that could then be remedied as the role is introduced into the field. It was suggested that if new roles were observed to emerge when participants were asked to retake the Belbin SPI (answering questions this time with respect to the teams with which they had just worked), this may suggest that the emerging roles were in fact needed, and that their emergence could be taken as evidence of a reaction to the fundamental need for the role.

In addition, participants were asked, after each experiment phase, to predict the answers that they thought their teammates would give to the Belbin SPI questions. Originally, this data was to be used to draw correlations between how a person viewed himself and how he was viewed by his teammates. This direction, however, was abandoned for several reasons, not the least of which was the raw complexity of the problem – correlating eight roles against eight roles, for three team members, each with three teammates. This does not even address the issue of whether teammate ascertainments should be compared with a person's original Belbin score or his team-context retake of the SPI. Also, multiple methods for correlation are possible (threshold method, primary role, etc.). The number of studies grows very quickly out of hand.

Another problem introduced by asking a participant to take the Belbin SPI for his teammate, is that the SPI was not designed for that purpose. Belbin's battery includes another test, the Observer Assessment (OA), for this purpose. Coworkers of the person whose role is being assessed complete the OA. The OA is used in parallel with the SPI, but it does not appear in Belbin's book *Management Teams*, and is available only for a fee from his website [BELA00]. The SPI cannot be assumed reliable (or held accountable) for anything except reasonably accurate self-perception of a person's team role. Lastly, in spite of these problems, some basic analyses and correlations were conducted to test the types of relations described above. In the vast majority of cases, nothing of statistical significance (or even mild interest) could be concluded about these analyses. In the rare cases where interesting trends appear, however, those trends are presented in this chapter.

The emergence of a role in retests was the most potentially interesting phenomenon that could have been observed, however, this phenomenon did not occur. The Primary Resource Investigator role, which was absent in the original run of the Belbin test occurred only once in the Belbin retake (see Table 6.1). (The reader is reminded here that this is once out of eighteen experiments.) This is likely to suggest that in a software development team model like the ones used in these experiments, Resource Investigators simply are not necessary: not only do they not exist naturally, but nobody attempts to take on the role. Note that in these experiments, cases in which there was a tie for a primary role were eliminated, whether the tie was for an original test or a retest.

Table 6.2 shows the count of each role in the original administration of the SPI, as well as the count of each role after retests were administered. The disparity between original and retest count totals is due to the fact that when a subject scored a tie between two primary roles, his test was excluded from the sample. Percentages shown in Table 6.2 are scaled as appropriate to reflect this

correction. The column labeled "% Change in representation in role context" shows the degree to which the role's representation in the sample changed from test to retest using the original role representation as a perspective. For example, Chairmen represented 15% of the original sample. When retests were completed, this diminished to 9%. Therefore, the Chariman role experienced a loss of 40% representation in the sample. { (15-9) / 15 = 40 }. The column labeled "Change in representation in sample context" shows the difference in context of the sample as a whole, and is computed as the simple difference between the third and fifth columns. This column is less sensitive to changes in roles in which the sample is small.

This work certainly does not attempt in any way to suggest that the Resource Investigator is not useful in *any* software engineering situation. In fact, it seems likely, as mentioned in chapter 5, that usability engineering and requirements engineering may be apt domains for people with this disposition. These domains however, did not play any significant part of the problems that were given to the experiment teams.

Table 6.2: Changes in sample profile from original SPI testing to SPI retesting

Role	Original Count	Portion of role in original sample	Retest Count	Portion of role in retest sample	% Change in representation in role context	Change in representation in sample context
CH	7	15%	4	9%	- 40%	- 6%
SH	12	25%	11	24%	- 4%	- 1%
PL	2	4%	4	9%	+109%	+ 5%
RI	0	0%	1	2%	*****	+ 2%
ME	6	13%	2	4%	- 65%	- 9%
CW	6	13%	14	30%	+143%	+17%
TW	9	19%	8	17%	- 7%	- 2%
CF	6	13%	2	4%	- 65%	- 9%
total	48		46			

The greatest increase in a role's representation between the original Belbin survey and the retake was for the Company Worker role. This seems ironic, given

the Company Worker's fairly neutral or "generic" trend, as shown in Chapter 5, but this may also be a reflection of the notion that because the Company Worker shows no particular trends, anyone can effectively be a software engineering Company Worker; all benefits derived from specific trends in other roles are irrelevant for the Company Worker. In this retest analysis, the count of primary Company Workers rose from 6 to 14 when subjects were retested.

The count of primary Plants also doubled when retests were considered, but only two primary Plants existed originally, so the addition of two more is somewhat unexciting. Actually, this may simply be the adoption of a person's secondary role. Monitor-Evaluators, Chairmen, and Completer-Finishers in the sample tended somewhat to shy away from their original Primary roles when re-tested, but it should not be argued that any role emerged because of any genuine need on the teams.

If some team need is being addressed, it seems to be with regard, again, to the Company Worker. The Company Workers of the sample, shown in Chapter 5 to have no particular tendency that deviates substantially from the general software engineering sample studied, may also have another explanation. Belbin describes the Company Worker as one who seeks out what needs to be done. Perhaps this flexibility, or ability to adjust to need, is reflected in the lack of the Company Worker's slant toward the pole of either scale.

C) The SPI Criticisms Revisited

Mention was made in Chapter 2 about the criticisms Belbin's Self-Perception Inventory has suffered. Problems range from ipsativity and lack of theoretical foundation [FURA93a] to construct validity and lack of agreement with other, more widely proven and accepted tests [BROW96]. In short, the test

while widely used, has not been unanimously validated. Perhaps the test-retest validity problems should be added to the list of complaints about the test.

The criticisms of the SPI, however (even in the words of one critic) have been, "far from damning," [FURA93b]. It generally is agreed that the concept deserves deeper exploration. Furthermore, the common results of attempts at a Belbin-Keirsey mapping system suggest that, indeed, *something* is being measured. Research, therefore, cannot be completely disregard the SPI. The significant results of experiments in this and earlier studies suggest that exploring team role concepts has potential to reap benefit in software engineering teams. Finally, the notion that each person can assume multiple roles cannot be overlooked.

D) Noteworthy Teammate Assessments

One dimension of the current study sought to determine if participants could identify their teammates' roles. In addition to previous tests, surveys, and techniques mentioned in this study, students were asked, after each phase of experiments, to retake the Belbin SPI for both of their teammates. That is to say, participants were asked to divine, from their experience in the experiments, how their teammates would answer the questions on the SPI. The purpose of such an exploration was to determine if the role/function a person professed to play matched the role their peers perceived them to fill.

The potential benefit in this line of reasoning is that, if one could successfully and accurately ascertain a teammate's role using the SPI, it would be unnecessary for management or project leaders to have each developer complete the test individually in order to build more effective teams using Belbin role theory. Instead, a sharp project leader who knew his team well could simply use

the test himself to discover each employee's role, and apply the benefit of his own familiarity with his team to the software process.

With this in mind, correlations were checked between each role scale for the following combinations:

1) A person's original Belbin test against his teammates' perception of him.
2) A person's Belbin retest (in team context) his teammates' perception of him.
3) One teammates' perception of an individual against the other teammates' perception of the individual.

Correlations for this experiment, generally speaking, were quite poor. In a few isolated instances, intriguing results were found when examining statistical p-values, but on visual inspection, these generally could be explained away because of extremely small representation in the sample (correlations involving the Resource-Investigator scale, for example) or very weak correlations. The strongest significant correlation only had a correlation coefficient of .23. The Team Worker scale statistically showed a weak positive correlation (**r** = .23, **p** = .02) between a person's original self-perception and his teammates' appraisal of his strength on this scale. Visual parsing of this data (Figure 6.3) shows that no real trend exists at all. In fact, high or low estimations by teammates of a person's Team Worker strength are just as likely for high-scoring, low-scoring, or no-scoring self-perceptions on this scale. If this represents the strongest significant correlation, certainly this avenue of exploration appears to be a fruitless.

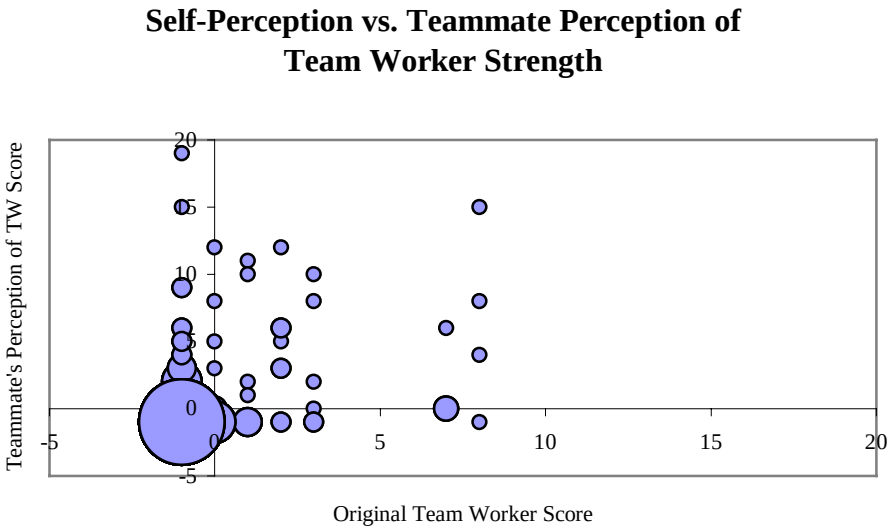


Figure 6.3: Team Worker strength – self-perception vs. teammate perception

E) Ascertainment of Previous Study

While interesting and statistically significant results were obtained in Stevens' earlier work in this domain, the Stevens study itself was somewhat plagued by weak correlations. It is possible that the problems identified here with the Belbin SPI are at the root of those weak correlations. In some cases, however, other factors should be considered.

The results of Stevens' studies of team leadership and the Plant role show lucid trends and phenomena, which also align neatly with role theory and intuitive reasoning. The Monitor-Evaluator study, however, produced results inconsistent with hypothesis, role theory, or the inherent needs of a software engineering team.

Stevens concluded that the Monitor-Evaluator might not be necessary for software engineering, based on a lack of any strong correlations in his empirical studies. (However, these conclusions were contradicted by some of his own anecdotal evidences!) What is more likely, however, is that the Monitor-

Evaluator has strong potential to affect the success of a real-world development team.

Stevens described the Monitor-Evaluator as the "decision-maker" of the team [STET98], and argued, while setting up the experiment, that decision-making is crucial to team success. This is an overly simplistic view of both decision-making and the role theory behind the Monitor-Evaluator. The Monitor-Evaluator may better be thought of as a "catcher." If his team proposes to follow a plan that, at its heart, is not possible or feasible, the Monitor-Evaluator is the one on the team who will recognize that fact and argue (perhaps incessantly) to prevent the mistake. If the team is presented with multiple disparate options, each of which is possible and reasonable, the Monitor-Evaluator has no great cause to argue for or against any of them; his effort is generally not wasted on decisions such as these, which are (in his perspective) of little or no consequence to the team.

Unfortunately, Stevens' academic study involved students who were involved in short, ninety-minute projects. Also (as mentioned in an earlier chapter), the senior-level software engineering course used for the study has a reputation for attracting the very brightest students in the Computer Science department. These facts, paired together, may suggest that: 1) the projects were not sufficiently complex to instigate the generation of inherently bad design solutions, and therefore 2) The Monitor-Evaluators had little or no opportunity to make any real use of their gift – catching bad plans before they become manifest in bad implementations.

Stevens' industry study reinforces this notion. Industry projects, by virtue of the fact that they are larger, real-world projects that often span a significant amount of time, should offer more opportunity for complex designs, erroneous plans, and bad decisions. In addition, real-world projects often involve previously unexplored or unfamiliar knowledge, whereas short programming projects (while

often presenting tricky twists) generally have somewhat underlying concepts that are largely pre-understood. The industry survey conducted by Stevens suggested that Monitor-Evaluators are beneficial, contrary to his own empirical research.

The effect of this role on software engineering teams should be re-evaluated, but from an industry perspective. Stevens' results in this case should be considered inconclusive.

Chapter VII: Conclusions And Future Work

A) Discussion of Results

Experimental results seem to show that a software engineering team's productivity, effectiveness, and success are affected by the presence or absence of Belbin team roles. Previous work demonstrates that a team benefits from the presence of a single Shaper and at least one Plant. This study demonstrates that the presence of Company Workers may make the team less predictable. This unpredictable nature is supported on several fronts, in addition to the original experiment which demonstrated that a team's success rate, as measured by completion time and number of successful completions, becomes less predictable as the number of Company Workers on the team is increased.

The Company Worker's draw to the design stage of software engineering, suggests that this inclination may contribute to the variance that the Company Worker adds to a team's effectiveness. The adage that "too many cooks spoil the broth" is relevant in this sense; if Company Workers tend to design, and there are a high number of Company Workers on the team, then it seems reasonable to assume that there may be too many designers for the design's integrity to be trusted. A weak increase in variance of success was observed as the number of reported designers on the team increased, suggesting that the connection here is at least somewhat justified. This reinforces and supports the work of proponents of small development teams such as the Chief Programmer Team, where design is generally limited to one or two individuals.

Additionally, when compared with other software engineers, the Company Worker exhibited no trend, disposition, or propensity at all toward *any* of the Keirsey scales. This statement could not be made for any other team role. In other words, the software engineering Company Worker is equally likely to be more Introverted or Extroverted than his average colleagues. He is equally likely

to be more Intuitive or Sensing, Thinking or Feeling, and Judging or Perceiving. In short, the knowledge that a person is a Company Worker software developer provides no more information about his likely Keirsey disposition than one could gain simply by knowing that he is a software developer.

Next, an analysis of claims regarding which team members functioned to keep their teams "on track" during experiments suggests that the Company Worker is least often helpful in that respect. Whereas 39% of the study sample reputed to be primary forces in maintaining the team's source of focus, only 18% of Company Workers were so reported. Consider this in parallel with the fact that more primary Company Workers were present in the sample than any other role, and each participant was independently rated by four different teammates (two teammates per phase). Monitor-Evaluators and Shapers each provided this focus between 60% and 70% of the time. Logically, if Company Workers are unlikely to keep a team on track, then filling a team's slots with Company Workers reduces the likelihood than anyone on the team will ensure that the team is constantly proceeding as needed. While it does make sense to imagine that a team could be successful without someone deliberately steering, such a team would succeed only by luck or talent, not by method. Again, this notion reinforces the idea that the Company Worker is a wildcard on the team.

Finally, the relatively weak test-retest correlation for the Company Worker scale of the Belbin SPI – even in comparison to the other role scale correlations – suggests that the Company Workers scale is unpredictable at best.

B) Application of Results

A re-examination of the original experiment data from Chapter 4 (see Figures 4.7 – 4.8) reveals that individual teams with two Company Workers generally perform somewhat consistently. In other words, high-performing teams

with two Company Workers seem to continue to perform well, and low-performing teams with two Company Workers seem to continue to perform poorly. This suggests that the Company Worker does not cause a stable team to perform erratically. Instead, the effect of an *individual* Company Worker on a *particular* team cannot be predicted.

For project leaders and managers, on a practical level, this means that if a team with several Company Workers is already known to perform well, *don't change the team!* Altering an effective team simply to reduce the number of Company Workers (to make the team more predictable) is likely to cause the performance of the team to drop sharply. Conversely, if a team rich in Company Workers is performing poorly, swapping one or more of the Company Workers – either for other Company Workers or for other roles entirely – is likely to result in positive productivity changes.

C) Emerging Roles

No evidence from these studies suggests strongly that roles are needed which are not already present. This conclusion, however must be tempered with several conditions.

1) Only small teams were studied;
2) Only projects of extremely short duration and scope were studied;
3) Only traditional software development was studied, excluding other sub-domains, such as usability engineering or requirements analysis.

It should not be categorically concluded that other roles are counterproductive or unneeded. Further research should examine where those other roles are needed before concluding that other roles are counterproductive or unnecessary.

D) Implications for Software Engineering

It has been shown that team role has an effect on team dynamic, success, and individual propensity for various software engineering activities. It is reasonable to assume that these facts could have implications in a broader sense. Not only are there different phases of software engineering, there are also several types of programming projects. Some software engineering teams are devoted to specific domain applications, while other teams focus on different divisions of the same application or project. Belbin roles may affect these team subsets as much as they do generic three-person team, but the effect may be very different that what has been observed in this study.

Buie pointed out that among computer scientists, a division in Myers-Briggs type in the Sensing-Intuitive scale reflected a tendency in the computer science field toward commercial / industrial applications and scientific / systems work [BUIE88]. The software industry has expanded, divided, and specialized many times since Buie's work, and her initial speculation certainly must have further reaching implications than those she enumerated.

It is particularly interesting that the usability engineering community, largely consisting of software developers with a background in psychology, has not yet picked up on the concept of team role effects. Usability seems to generally attract a different "breed" of computer scientist than do databases or networking. To offer a more specific example, this study includes very little mention of Belbin's Resource Investigator role. This role is characterized by the ability to draw innovative concepts from sources beyond the team itself. In this study, there were simply no Resource Investigators present, and previous work suggests that the Resource Investigator is uncommon in general software engineering. The Resource Investigator is an extremely extroverted role, and Belbin describes him by saying that "he is never in his office, and if he is, he's on the phone" [BELR81]. These characteristics, which are uncommon among most software developers, may be a boon to a usability team.

Certain personality traits would likely be beneficial to usability engineering, and certain combinations of team roles may be advantageous for a usability team, yet not for a more generic software development team. Usability serves merely as an example here, and similar concepts could surely be applied to other specialization areas.

E) Kiersey-Belbin

A comparison between results of the Belbin Self-Perception Inventory and the Keirsey Temperament Sorter suggest that a person's team role is related on some level to his temperament, as measured by the Keirsey test. The fact that Keirsey (and temperament theory in general) are generally accepted personality metrics and concepts suggests that these results – while certainly not a validation of the Belbin SPI – do confirm that the SPI is doubtlessly measuring *something*. This is confirmed simply in the sense that personality differences identified by Belbin are reflected to some degree by Keirsey.

F) The Belbin SPI

Although significance can obviously be obtained by using Belbin's roles to study the effectiveness of software engineering teams, the criticisms of the Belbin SPI, combined with analysis of the test-retest validity for this study suggests that the SPI may have enough problems to make it suspect as a tool. This is not to suggest that Belbin's work, theories, or even the SPI itself are valueless, but rather that potential for application may be limited within the software engineering domain.

To illustrate with a hypothetical example, suppose that application of Belbin's roles, theory, and tools offers a 10% increase in performance to a development team. While this increase may look attractive to a project manager, the personnel reshuffling needed to realize the benefit from Belbin may likely cause a team to become imbalanced with regard to other more important factors, such as domain knowledge, skill, or even personal attitude and interest. The setbacks suffered by a team which loses needed resources or attributes may more than counteract any benefits or good intentions of a manager who rearranges his teams in accordance with team role theory. Although the 10% increase may seem tempting, the restructuring may cost the team 30% of its overall ability. No manager could challenge the assertion that spending $30.00 to get $10.00 is a foolhardy pursuit.

With this in mind, project managers are cautioned that while evidence suggests some value in role theory and the SPI, neither should be used as a substitute for more reliable and proven software project management concepts. Generally speaking, if one chooses to make use of Belbin theory, it can be useful, but should be taken with a grain of salt.

G) The most important factor to Software Engineering:

A few researchers have independently offered similar speculation on what causes different teams to perform differently. Tannenbaum et. al assert the basic premise that a more motivated, skilled team will always perform better than a less motivated and skilled team, if all other factors are constant [TANS96]. Perhaps a more interesting notion was put forth by Glass, who suggests that improving quality may be simpler than is normally represented, and advocates fostering and nurturing a quality *climate* "in which people who think quality are hired and helped" [GLAR91]. While it is acknowledged that such a theory may be difficult or even impossible to prove experimentally, the value of the theory itself is

conceded. Observations of participants in these experiments and similar experimental settings, as well as real-life situations with other students and colleagues, demonstrate the existence of the occasional team member who, for varied or unexplained reasons, simply is uninterested or unmotivated and does little to contribute to the team's effort or success.

As described in the Bible, "Daniel was preferred above the presidents and princes, because an excellent spirit was in him; and the king thought to set him over the whole realm" [DAN63]. It is this attitude – this "spirit of excellence" – the genuine desire and drive to produce quality for its own sake, which is desirable and most beneficial for software development.

H) Summary of Conclusions

1. A software engineer's personality has an effect on his team's success;
2. There is a difference in the nature of Belbin's team roles as applied to software engineering teams, as opposed to management teams;
3. Whereas Belbin's Company Worker is useful in a management team setting, the role makes a software engineering team less predictable;
4. In addition to adding unpredictability to a team, the Company Worker role in software engineering is also unpredictable, in that it can not be distinguished from other software engineers with regard to the Keirsey Temperament Sorter;
5. Various roles seem to be drawn to particular stages of the software engineering process;
6. The Belbin Self-Perception Inventory, as published in his 1981 book, is useful, but has not been validated, and should be taken with a grain of salt;
7. The test-retest validity of the Self-Perception Inventory suggests that either the test is problematic or a different testing method is warranted. The test may have to be taken by subjects in pre-existing long-term teams, for example;
8. A person's team role may change over time, or from team to team;
9. Trends between the Belbin Self Perception Inventory and the widely accepted Keirsey Temperament Sorter suggest that the Self Perception Inventory is measuring *something*.

I) Future Work

1) Future Role Studies

1a. The Chairman

It is apparent from the last experiment analyses of chapter 4 that the Chairman may be a source of ideas for his team, and is certainly perceived by his teammates as being very involved. For involvement in the design, implementation, and debugging phases, however, his teammates rate him somewhat below average in each case. (When rating himself, the Chairman was consistently convinced that he was involved in design, however.) The Chairman's exact contribution to his team remains to be determined. He certainly is not able to take the leadership position from the Shaper.

Belbin suggested that the Plant and Chairman work particularly well together in management teams [BELR81]. Perhaps, given the observations mentioned above, the Chairman is able to provide the initial ideas to inspire the Plant's curiosities, and it is then the Plant who develops the ideas into a workable design. This process would be quite interesting to observe in action.

1b. Resource Investigator

The Resource-Investigator's existence, input, and contribution with respect to non-traditional software teams (like requirements analysis or usability engineering) would be interesting to study. However, it is unlikely that a study of this type could be conducted with a software engineering class, as was this study. A usability class project or industry setting would likely produce more significant results.

1c. Monitor-Evaluator

Stevens' study of the Monitor-Evaluator begs for a second look. The true nature of this role requires a study conducted of complex, lengthy projects where the Monitor-Evaluator's gifts become truly advantageous.

1d. Team Worker

Although no significant results were observed in this study regarding the Team Worker, it is interesting to note in Stevens' survey of computer science academia and industry that (unlike other roles) the percentage of Team Workers in industry far outweighs the percentage in academia. Why is this? Does the *need* for the Team Worker not become apparent until a real-world setting? Do Team Workers major in other areas in college, and then get recruited into the field? Why do they suddenly appear so conspicuously in industry?

1e. Completer-Finisher

The Completer Finisher remains an enigma. His role description suggests deep attention to detail, which should seem very important to a software engineering team. His contributions, as perceived both by himself and his teammates in this study, however, seem incredibly small in every respect when contrasted against other roles. What is the Completer Finisher doing? Why does a role, so characterized by attention to detail, seem not to help his team? Perhaps attention to detail slows him down, and his teammates simply could not afford to sacrifice the project velocity.

2) Other Areas For Consideration

Although Belbin's work seems to warrant much of the scrutiny it has received, the concepts have been used successfully. Similar work on team roles by Team Management Systems (UK) Ltd. (TMS) also resulted in a defined set of

eight roles, several of which bear striking similarity to Belbin's roles [RUSR96]. The TMS system is also used successfully with regard to management teams, and may suggest that Belbin's basic role theory itself is valid, but bears some refinement. Furthermore, the TMS roles have not suffered the same criticisms as has Belbin's work. If future studies continuing this body of research are conducted, it would be prudent to consider the potential of other, less debated tools, such as this.

Bibliography

[AGIL01] Agile Alliance Website, http://www.agilealliance.org , 2001.

[ALBA79] Albrecht, A.J., Measuring Application Development Productivity, *Proc. IBM Applications Development Symposium*, IBM, 1979.

[ARTL92] Arthur, Lowell J., *Improving Software Quality: An Insider's guide to TQM*, John Wiley and Sons Inc, 1992.

[AUEK01] Auer, K., *Extreme Programming Applied*, Addison-Wesley Publishing Co., 2001.

[BAKF72] Baker, F.T., Chief programming team management of production programming, *IBM Systems Journal*, Vol. 11, No. 1, 1972.

[BECK99] Beck, K. *Extreme Programming Explained: Embrace Change* Addison-Wesley Publishing Co., 1999.

[BELR81] Belbin, R. Meredith, *Management Teams*, John Wiley & Sons, New York, 1981.

[BELR93] Belbin, R. Meredith, A reply to the Belbin Team-Role Self-Perception Inventory by Furnham, Steele, & Pendleton, *Journal of Occupational and Organizational Psychology* (66), 1993.

[BELA00] Belbin Associates, https://team-belbin.com/ 2000.

[BIDB79] Biddle, Bruce J., *Role Theory: Expectations, Identities, and Behaviors*, Academic Press, New York, 1979.

[BOEB78] Boehm, Barry W., *Characteristics of software quality*. North Holland Publishing Co., Amsterdam New York 1978.

[BOEB81] Boehm, Barry W., *Software Engineering Economics*. Prentice-Hall, Inc., 1981.

[BRAK64] Bradway, K., Jung's Psychological Types, *Journal of Analytical Psychology*, Vol. 9, Tavistock Publishers, 1964.

[BROF95] Brooks, Fred P., Jr., *The Mythical Man Month*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1995.

[BROF87] Brooks, Fred P., Jr., No Silver Bullet - Essence and Accidents of Software Engineering, *IEEE Computer*, April, 1987, pp. 10-19.

[BROW96] Broucek, W. G. & Randell, G. An assessment of the construct validity of the Belbin Self-Perception Inventory and Observers Assessment from the perspective of the five-factor model, *Journal of Occupational and Organizational Psychology* (69), 1996.

[BRUT01] Brumley, Troy, (Lead Software Engineer; Cincom Systems Inc.), Personal Interview, October 18, 2001.

[BUIE88] Buie, Elizabeth A., Personality and System Development: What's the connection?, *System Development*, January, 1988.

[BUSC85] Bush, Chandler M. and Lawrence L. Schkade., In Search of the Perfect Programmer, *Datamation*, 31(1), 1985.

[CATR70] Cattell, Raymond B., Herbert W. Eber, and Maurice M. Tatsuoka, *Handbook for the Sixteen Personality Factor Questionnaire (16 pf)*, Institute for Personality and Ability Testing, Champaign, Illinois, 1970.

[CHIDS94] Chidamber, S. R. and C.F. Kemerer, A Metrics Suite for Object Oriented Design, *IEEE Trans. Software Engineering*, vol. 20, 1994.

[COSP92] Costa, P.T. Jr. & McCrae R.R., Four ways five factors are basic, *Personality and Individual Differences*, 13, 1992.

[CURB86] Curtis, Bill, Elliot M. Soloway, Ruven E. Brooks, John B. Black, Kate Ehrlich, and H. Rudy Ramsey, Software Psychology: The Need for an Interdisciplinary Program, *Proceedings Of the IEEE*, August, 1986.

[CURB90] Curtis, Bill, Empirical Studies of the Software Design Process, *Human-Computer Interaction - INTERACT '90*, 1990.

[CURB91] Curtis, Bill, Techies as Non-Technical Factors in Software Engineering?, *IEEE Transactions on Software Engineering*, 1991.

[CURB97] Curtis, B., Software Process Improvement: Methods and Lessons Learned, *Proceedings of the 1997 international conference on Software engineering*, 1997.

[DAN63] Daniel 6:3, *The Bible, King James Version*.

[DEMT82] DeMarco, T., *Controlling Software Projects*, Yourdon Press : Englewood Cliffs, N.J., 1982.

[DULV95] Dulewicz, V. A validation of Belbin's team roles from 16PF and OPQ using bosses ratings of competence, *Journal of Occupational and Organizational Psychology* (68), 1995.

[EYSH64] Eysenck, Hans, *The Eysenck Personality Inventory*, 1964.

[FURA93a] Furnham, A., Steele, H. & Pendleton, D., A psychometric assessment of the Belbin Team-Role Self-Perception Inventory, *Journal of Occupational and Organizational Psychology* (66), 1993.

[FURA93b] Furnham, A., Steele, H. & Pendleton, D., A response to Dr. Belbin's reply, *Journal of Occupational and Organizational Psychology* (66), 1993.

[GLAR91] Glass, Robert, The link between software quality and software maintenance, *Software Conflict: Essays on the Art and Science of Software Engineering*, Prentice-Hall, 1991.

[HACJ90] Hackman, J. Richard, *Groups That Work (and Those That Don't)*, Jossey-Bass Publishers, San Francisco, 1990.

[HALM77] Halstead, Maurice H. *Elements of Software Science, Operating, and Programming Systems Series* Volume 7. New York, NY: Elsevier, 1977.

[MILH83] Mills, Harlan D., Programmer Productivity Through Individual Responsibility, in *Software Productivity*, Little, Brown and Company, 1983.

[HENS81] Henry, S.M. and Kafura, D.G. Software Structure Metrics Based on Information Flow, *IEEE Transactions on Software Engineering*, (7,5), September, 1981.

[HENS99] Henry, S. and Stevens, K.T., Using Leadership Roles to improve Team Effectiveness: An Empirical Investigation, *Journal of Systems and Software*, 44, 1999.

[HOGC90] Hogg, C., Team Building, *Personnel Management Factsheet*, 1990.

[HOLR87] Holt, Robert W., Deborah A. Boehm-Davis, and Alan C. Schultz, Mental Representations of Programs for Student and Professional Programmers, *Empirical Studies of Programmers: Second Workshop*, (Gary M. Olson, Sylvia Sheppard, and Elliot Soloway eds.), Ablex Publishing Corporation, Norwood, New Jersey, 1987.

[HUMW89] Humphrey, W., *Managing the Software Process*, Addison-Wesley Publishing Company, 1989.

[HUMW95] Humphrey, W., *A Discipline for Software Engineering*, Addison-Wesley Publishing Company, 1995.

[IND] Industrial Training Research Unit, Cambridge University, England, *Personal Preference Questionnaire*, Year Unknown.

[ISA96] Isaacs, E., Interviewing Customers: Discovering What They Can't Tell You, *Proceedings of the Conference on Computer-Human Interaction*, Boston, MA: ACM Press, 1996.

[JOHC88] Johnson, C.E., Wood, R. & Blinkhorn, S.F., Spuriouser and spuriouser: The use of ipsative personality tests. *Journal of Occupational Psychology*, 61, 1988.

[JONC86] Jones, C., *Programming Productivity*, McGraw-Hill, 1986.

[JONC98] Jones, C., *Estimating Software Costs*, McGraw-Hill, 1998.

[JUNC23] Jung, C., *Psychological Types*, New York: Harcourt Brace, 1923.

[KEID84] Keirsey, David and Marilyn Bates, *Please Understand Me*, Prometheus Nemesis Book Company, Del Mar, California, 1984.

[KELM91] Kellner, Marc I., Non-Technical Issues in Software Engineering, *IEEE*, 1991.

[LIKR32] Likert, Rensis, A technique for the measurement of attitudes. *Archives of Psychology*, 140, June, 1932.

[LIW93] Li, W. and Henry, S. M. Object Oriented Metrics which Predict Maintainability, special issue on the *Object Oriented Paradigm of the Journal of Systems and Software*, Vol. 23, No. 2, November, 1993.

[LYOM85] Lyons, Michael L., The DP Psyche, *Datamation*, 1985.

[MCCT89] McCabe, Thomas J. & Butler, Charles W., Design Complexity Measurement and Testing, *Communications of the ACM* 32, 12, 1989.

[MCCT 94] McCabe, Thomas J. & Watson, Arthur H., Software Complexity, *Crosstalk, Journal of Defense Software Engineering* 7, 12, 1994.

[MEYG79] Meyers, Glenford. J., *The Art of Software Testing*. John Wiley and Sons, Inc., New York, 1979.

[MILH71] Mills, H., Chief programmer teams, principles, and procedures, *IBM Federal Systems Division Report FSC 71-5108*, Gaithersburg, Maryland, 1971.

[MILJ95] Milton, J. S., and J. C. Arnold, *Introduction to Probability and Statistics: Principles and Applications for Engineering and the Computing Sciences, 3rd ed*., McGraw-Hill, Inc., 1995.

[MYEI80] Myers, Isabel Briggs, *Gifts Differing*, Consulting Psychologists Press, Inc., Palo Alto, California, 1980.

[PAUM93] Paulk, Mark C., Bill Curtis, Mary Beth Chrissis, and Charles V. Weber, Capability Maturity Model, Version 1.1, *IEEE Software*, Vol. 10, No. 4, July 1993.

[PAU94] Paulish, Daniel J. and Anita D. Carleton, Case Studies of Software Process Improvement Measurement, *IEEE Computer*, 1994.

[PELJ01] Pelrine, Joseph, (CO, MetaProg), Personal Interview, October 18, 2001.

[PRER01] Pressman, R. S., *Software Engineering, A Practitioner's Approach, 5th ed.*, McGraw-Hill, 2001.

[RUSR96] Rushmer, Rosemary K., Is Belbin's Shaper Really TMS's Thruster-Organizer? An empirical investigation into the

correspondence between the Belbin and TMS team role models, *Leadership and Organization Development Journal*, V. 17-1, 1996.

[SACH68] Sackmann, H., Erikson, W.J., & Grant, E.E. Exploratory Experimental Studies Comparing Online and Offline Programming Performance, *Communications of the ACM*, Vol. 11 No. 1, Jan. 1968.

[SAS90] *SAS Procedures Guide Version 6 Third Edition*, SAS Inc., 1990.

[SAVP91] Saville, P. & Wilson, E., The reliability and validity of normative and ipsative approaches in the measurement of personality, *Journal of Occupational Psychology*, 64, 1991.

[SCHB80] Schneiderman, Ben, *Software Psychology*, Winthrop Publishers, Inc., Cambridge, Massachusetts, 1980.

[SCHP01] Schoenhoff, Peter Klaus, Henry, S. and Lloyd, W., Software Engineering Teams Evaluation, *Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, 2001.

[SHES79] Sheppard S.B., Curtis, B., Milliman, P., & Love, T., Modern coding practices and programmer performance. *IEEE Computer*, Vol. 12 No. 12, 1979.

[SITS84] Sitton, S., and Chmelir, G., The intuitive computer programmer, *Datamation*, 15, October, 1984

[SNEG80] Snedecor, George W. and William G. Cochran, *Statistical Methods*, The Iowa State Univ. Press, 1980.

[STET98] Stevens, T., *The Effects of Roles and Personality Characteristics on Software Development Team Effectiveness*. (Dissertation) 1998.

[TANS96] Tannenbaum, S. I., Salas, E. & Cannon-Bowers, J.A. Promoting team effectiveness. In West, M.A. (Editor) *Handbook of workgroup psychology*. Wiley & Sons: Chichester, 1996.

[THOR90] Thomsett, Rob, Effective Project Teams A Dilemma, a Model, a Solution, *American Programmer*, July/August 1990.

[VONA84] von Mayrhauser, Anneliese, Task Analysis and the Selection of Software Development Team Structures, *Proceedings of COMPSAC84*, IEEE Computer Society Press, 1984.

[WATG80] Watson, G., & Glaser, E. M., *Watson-Glaser Critical Thinking Appraisal*, New York: Psychological Corporation, 1980.

[WEIG71] Weinberg, G.M. *The Psychology of Computer Programming*, Van Nostrand Reinhold, 1971.

[WILL00] Williams, Laurie A., *The Collaborative Software Process*, (Dissertation), 2000.

[YOUE92] Yourdon. E., *Decline and Fall of the American Programmer*, Englewood Cliffs, N.J. Yourdon Press, 1992.

Appendix A: The Belbin Self-Perception Inventory

For each of the following sections, **distribute ten (10) points** among the 8 sentences that you think best describe your behavior. These points may be distributed among several sentences: in extreme cases they might be spread among all 8 sentences or ten points may be given to a single sentence. Enter the points in the spaces in front of each sentence. For example, for section 1, you might give five points to statement 2, two points to each statement 4 & 5, and one point to statement 7. (Suggestion: Read all of the sentences, crossing out the ones that are not true or hardly true, then distribute points among those sentences left.)

I. What I believe I can contribute to the team:

1. ___ I think I can quickly see and take advantage of new opportunities.
2. ___ I can work well with a very wide range of people.
3. ___ Producing ideas is one of my natural assets.
4. ___ My ability rests in being able to draw people out whenever I detect they have something of value to contribute to group objectives.
5. ___ My capacity to follow through has much to do with my personal effectiveness.
6. ___ I am ready to face temporary unpopularity if it leads to worthwhile results in the end.
7. ___ I can usually sense what is realistic and likely to work.
8. ___ I can offer a reasoned case for alternate courses of action without introducing bias or prejudice.

II. If I have a possible shortcoming in teamwork, it could be that:

1. ___ I am not at ease unless meetings are well structured and controlled and generally well conducted.
2. ___ I am inclined to be too generous towards others who have a valid viewpoint that has not been given proper airing.
3. ___ I have a tendency to talk too much once the group gets on to new ideas.
4. ___ My objective outlook makes it difficult for me to join in readily and enthusiastically with colleagues.
5. ___ I am sometimes seen as forceful and authoritarian if there is a need to get something done.
6. ___ I find it difficult to lead from the front, perhaps because I am over-responsive to group atmosphere.
7. ___ I am apt to get caught up in ideas that occur to me and so lose track of what is happening.
8. ___ My colleagues tend to see me as worrying unnecessarily over detail and the possibility that things may go wrong.

III. When involved in a project with other people:

1. ___ I have an aptitude for influencing people without pressurizing them.
2. ___ My general vigilance prevents careless mistakes and omissions being made.
3. ___ I am ready to press for action to make sure that the meeting does not waste time or lose site of the main objective.
4. ___ I can be counted on to contribute something original.
5. ___ I am always ready to back a good suggestion in the common interest.
6. ___ I am keen to look for the latest in new ideas and developments.
7. ___ I believe my capacity for judgment can help to bring about the right decisions.
8. ___ I can be relied upon to see that all essential work is organized.

IV. My characteristic approach to group work is that:

1. ___ I have a quiet interest in getting to know colleagues better.
2. ___ I am not reluctant to challenge the views of others or to hold a minority view myself.
3. ___ I can usually find a line of argument to refute unsound propositions.
4. ___ I think I have a talent for making things work once a plan has to be put into operation.
5. ___ I have a tendency to avoid the obvious and to come out with the unexpected.
6. ___ I bring a touch of perfectionism to any job I undertake.
7. ___ I am ready to make use of contacts outside the group itself.
8. ___ While I am interested in all views I have not hesitation in making up my mind once a decision has to be made.

V. I gain satisfaction in a job because:

1. ___ I enjoy analyzing situations and weighing up all of the possible choices.
2. ___ I am interested in finding practical solutions to problems.
3. ___ I like to feel I am fostering good working relationships.
4. ___ I can have a strong influence on decisions.
5. ___ I can meet people who may have something new to offer.
6. ___ I can get people to agree on a necessary course of action.
7. ___ I feel in my element where I can give a task my full attention.
8. ___ I like to find a field that stretches my imagination.

VI. If I am suddenly given a difficult task with limited time and unfamiliar people:

1. ___ I would feel like retiring to a corner to devise a way out of the impasse before developing a line.
2. ___ I would be ready to work with the person who showed the most positive approach.
3. ___ I would find some way of reducing the size of the task by establishing what different individuals might best contribute.
4. ___ My natural sense of urgency would help to ensure that we did not fall behind schedule.
5. ___ I believe I would keep cool and maintain my capacity to think straight.
6. ___ I would retain a steadiness of purpose in spite of the pressures.
7. ___ I would be prepared to take a positive lead if I felt the group was making no progress.
8. ___ I would open up discussions with a view to stimulating new thoughts and getting something moving.

VII. With reference to the problems to which I am subject to working in groups:

1. ___ I am apt to show my impatience with those who are obstructing progress.
2. ___ Others may criticize me for being too analytical and insufficiently intuitive.
3. ___ My desire to ensure that work is properly done can hold up proceedings.
4. ___ I tend to get bored rather easily and rely on one or two stimulating members to spark me off.
5. ___ I find it difficult to get started unless the goals are clear.
6. ___ I am sometimes poor at explaining and clarifying complex points that occur to me.
7. ___ I am conscious of demanding from others the things I cannot do myself.
8. ___ I hesitate to get my points across when I run up against real opposition.

Appendix B: The Keirsey Temperament Sorter

For each question, circle either answer **(a)** or answer **(b)**. There are no right or wrong answers, as about half the population will agree with either answer you choose. Please answer all questions. Mark exactly one answer for each question. This survey should take approximately 20-30 minutes.

This test is taken from ***Please Understand Me***, by David Keirsey and Marilyn Bates, © 1978.

1. At a party do you
- (a) interact with many, including strangers
- (b) interact with a few, known to you

2. Are you more
- (a) realistic than speculative
- (b) speculative than realistic

3. Is it worse to
- (a) have your "head in the clouds"
- (b) be "in a rut"

4. Are you more impressed by
- (a) principles
- (b) emotions

5. Are you more drawn toward the
- (a) convincing
- (b) touching

6. Do you prefer to work
- (a) to deadlines
- (b) just "whenever"

7. Do you tend to choose
- (a) rather carefully
- (b) somewhat impulsively

8. At parties, do you
- (a) stay late, with increasing energy
- (b) leave early, with decreased energy

9. Are you more attracted to
- (a) sensible people
- (b) imaginative people

10. Are you more interested in
- (a) what is actual
- (b) what is possible

11. In judging others are you more swayed by
- (a) laws than circumstances
- (b) circumstances than laws

12. In approaching others is your inclination to be somewhat
- (a) objective
- (b) personal

13. Are you more
- (a) punctual
- (b) leisurely

14. Does it bother you more having things
- (a) incomplete
- (b) completed

15. In your social groups do you
- (a) keep abreast of others' happenings
- (b) get behind in the news

16. In doing ordinary things are you more likely to
- (a) do it the usual way
- (b) do it your own way

17. Writers should
- (a) "say what they mean and mean what they say"
- (b) express things more by use of analogy

18. Which appeals to you more
- (a) consistency of thought
- (b) harmonious human relationships

19. Are you more comfortable in making
- (a) logical judgments
- (b) value judgments

20. Do you want things
- (a) settled and decided
- (b) unsettled and undecided

21. Would you say you are more
- (a) serious and determined
- (b) easy going

22. In phoning do you
- (a) rarely question that it will all be said
- (b) rehearse what you'll say

23. Facts
- (a) "speak for themselves"
- (b) illustrate principles

24. Are visionaries
- (a) somewhat annoying
- (b) rather fascinating

25. Are you more often
- (a) a cool-headed person
- (b) a warm-hearted person

26. Is it worse to be
- (a) unjust
- (b) merciless

27. Should one usually let events occur
- (a) by careful selection and choice
- (b) randomly and by chance

28. Do you feel better about
- (a) having purchased
- (b) having the option to buy

29. In company do you
- (a) initiate conversation
- (b) wait to be approached

30. Common sense is
- (a) rarely questionable
- (b) frequently questionable

31. Children often do not
- (a) make themselves useful enough
- (b) exercise their fantasy enough

32. In making decisions do you feel more comfortable with
- (a) standards
- (b) feelings

33. Are you more
- (a) firm than gentle
- (b) gentle than firm

34. Which is more admirable:
- (a) the ability to organize and be methodical
- (b) the ability to adapt and make do

35. Do you put more value on the
- (a) definite
- (b) open-ended

36. Does new and non-routine interaction with others
- (a) stimulate and energize you
- (b) tax your reserves

37. Are you frequently
- (a) a practical sort of person
- (b) a fanciful sort of person

38. Are you more likely to
- (a) see how others are useful
- (b) see how others see

39. Which is more satisfying:
- (a) to discuss an issue thoroughly
- (b) to arrive at agreement on an issue

40. Which rules you more:
- (a) your head
- (b) your heart

41. Are you more comfortable with work that is
- (a) contracted
- (b) done on a casual basis

42. Do you tend to look for
- (a) the orderly
- (b) whatever turns up

43. Do you prefer
- (a) many friends with brief contact
- (b) a few friends with more lengthy contact

44. Do you go more by
- (a) facts
- (b) principles

45. Are you more interested in
- (a) production and distribution
- (b) design and research

46. Which is more of a compliment:
- (a) “There is a very logical person”
- (b) “There is a very sentimental person”

47. Do you value in yourself more that you are
- (a) unwavering
- (b) devoted

48. Do you more often prefer the
- (a) final and unalterable statement
- (b) tentative and preliminary statement

49. Are you more comfortable
- (a) after a decision
- (b) before a decision

50. Do you
- (a) speak easily and at length with strangers
- (b) find little to say to strangers

51. Are you more likely to trust your
- (a) experience
- (b) hunch

52. Do you feel
- (a) more practical than ingenious
- (b) more ingenious than practical

53. Which person is more to be complimented: one of
(a) clear reason
(b) strong feeling

54. Are you inclined more to be
(a) fair minded
(b) sympathetic

55. Is it preferable mostly to
(a) make sure things are arranged
(b) just let things happen

56. In relationships should most things be
(a) renegotiable
(b) random and circumstantial

57. When the phone rings do you
(a) hasten to get to it first
(b) hope someone else will answer

58. Do you prize more in yourself
(a) a strong sense of reality
(b) a vivid imagination

59. Are you drawn more to
(a) fundamentals
(b) overtones

60. Which seems the greater error:
(a) to be too passionate
(b) to be too objective

61. Do you see yourself as basically
(a) hard-headed
(b) soft hearted

62. Which situation appeals to you more:
(a) the structured and scheduled
(b) the unstructured and unscheduled

63. Are you a person that is more
(a) routinized than whimsical
(b) whimsical than routinized

64. Are you more inclined to be
(a) easy to approach
(b) somewhat reserved

65. In writings do you prefer
(a) the more literal
(b) the more figurative

66. Is it harder for you to
(a) identify with others
(b) utilize others

67. Which do you wish more for yourself
(a) clarity of reason
(b) strength of compassion

68. Which is the greater fault:
(a) being indiscriminate
(b) being critical

69. Do you prefer the
(a) planned event
(b) unplanned event

70. Do you tend to be more
(a) deliberate than spontaneous
(b) spontaneous than deliberate

Appendix C: Viability Questionnaire

Your Name _____
Teammate A _____
Teammate B _____

1) To what extent did each person exhibit leadership characteristics in your group?

	NOT AT ALL……VERY MUCH					
You	**1**	**2**	**3**	**4**	**5**	**6**
Teammate A	**1**	**2**	**3**	**4**	**5**	**6**
Teammate B	**1**	**2**	**3**	**4**	**5**	**6**

- If there was a team "leader," was this person officially chosen by your team, or did he/she simply assume a leadership role?

- What characteristics or actions made you identify this person (or these people) as a leader?

2) Where did the ***ideas*** for your solution come from (which person/people)?

	NOT AT ALL……VERY MUCH					
You	**1**	**2**	**3**	**4**	**5**	**6**
Teammate A	**1**	**2**	**3**	**4**	**5**	**6**
Teammate B	**1**	**2**	**3**	**4**	**5**	**6**

- Explain

3) Who kept the team "***on-track***", (managed time or other resources)?

	NOT AT ALL……VERY MUCH					
You	**1**	**2**	**3**	**4**	**5**	**6**
Teammate A	**1**	**2**	**3**	**4**	**5**	**6**
Teammate B	**1**	**2**	**3**	**4**	**5**	**6**

- How was that done? (for ***each*** problem)

4) How much was each person ***involved***?

	NOT AT ALL……VERY MUCH					
You	**1**	**2**	**3**	**4**	**5**	**6**
Teammate A	**1**	**2**	**3**	**4**	**5**	**6**
Teammate B	**1**	**2**	**3**	**4**	**5**	**6**

- What did each person contribute? (for ***each*** problem)

- Who worked on what activities? (for ***each*** problem)

5) For your team, did each person work more as an individuals or more together with the team?

WORKED AS INDIVIDUAL……..WORKED WITH THE TEAM

You	**1**	**2**	**3**	**4**	**5**	**6**
Teammate A	**1**	**2**	**3**	**4**	**5**	**6**
Teammate B	**1**	**2**	**3**	**4**	**5**	**6**

- Elaborate.

6) How did you divide team responsibilities?

- Why did you divide them that way?

- Check which team member was ***primarily*** involved in which activity. (If more than one apply, mark all that are apropriate.)

	3) esign	**Code**	**Test & Debug**
You			
Teammate A			
Teammate B			

7) Did your team function well together?

NOT AT ALL……VERY MUCH

1 2 3 4 5 6

- Were there any conflicts within the group that inhibited your progress?

NOT AT ALL……VERY MUCH

Between you & tmmt A	**1**	**2**	**3**	**4**	**5**	**6**
Between you & tmmt B	**1**	**2**	**3**	**4**	**5**	**6**
Between tmmt A & tmmt B	**1**	**2**	**3**	**4**	**5**	**6**

- Explain

8) Describe how your team attacked each problem. For example: were there any turning points in your team's strategy? Were there other plans that you tried or would like to have tried? Why were they rejected? Explain.

- Problem 1:

- Problem 2:

- Problem 3:

- Problem 4:

Appendix D: Expert Team-Rating Survey

Phase: _____

Experiment: _____

Name: _____

Rating: _____

Program Quality Rating Scale

10	Program finished in < 60 minutes with correct output
9	Program finished in < 75 minutes with correct output
8	Program finished in > 75 minutes with correct output
7	Program compiles, with some correct output, nearly finished
6	Program compiles with minimal correct output, with extensive intermediate/testing data output
5	Program compiles (or has minimal errors), no final output, some intermediate/testing data output
4	Program compiles (or has minimal errors), little or no output, reads in input data, performs extensive work on data
3	Program compiles (or has minimal errors), little or no output, reads in input data, performs extensive work on data
2	Does not compile, (many errors), reasonable attempt to read input data, and perform work
1	Does not compile, (many errors), more than a skeleton program, but still far from functional
0	Minimal coding, skeleton like program

Appendix E: Sample Programming Problem 1

ACM South Central Region
1999 Programming Contest

Problem #4: Soundex

Source file: soundex.ext
Input File: soundex.dat
Output File: soundex.out

Introduction
You have been given two lists of names from a small community. One list comprises people who currently live in the town; the other is a list of people whom [sic] have passed away leaving inheritances with no known living relatives. The Town council wishes to apportion this wealth to the closest living relatives in the town. Unfortunately the town records have been lost in a fire. The councilmen have decided to determine possible relatives by comparing names by their sound roots (i.e. soundex). Given the two lists, match possible relatives to the deceased.

Soundex Coding Guide
To compute the soundex value of a person's name, your program will use the first letter (even if it is one of the ignored letters in the soundex value table) of the person's surname, followed by 3 digits taken from the following table for subsequent letters in the person's surname. If the surname contains an insufficient quantity of letters to complete the soundex value, your program will simply pad the soundex value with zeroes to complete the 3 digits. For example, Mr. T's surname would simply be coded as T-000; and Mike Mart's surname would be coded as M-620. Soundex values are limited to a maximum of 3 digits after the first letter regardless of how long the surname may be.

Soundex Value	Mapped Letters
1	B, F, P, V
2	C, G, J, K, Q, S, X, Z
3	D, T
4	L
5	M, N
6	R
Ignored	A, E, I, O, U, H, W, Y

Additional Soundex Coding Rules

1. Names With Double Letters

If the surname has any double letters (even doubles that include the first letter of the surname),
they should be treated as one letter.
Gutierrez is coded G-362 (G, 3 for the T, 6 for the first R, second R ignored, 2 for the Z).

2. Names with Letters Side-by-Side that have the Same Soundex Code Number

If the surname has different letters side-by-side with the same soundex value, they should be
treated as one letter.
Pfister is coded as P-236 (P, F ignored, 2 for the S, 3 for the T, 6 for the R).
Jackson is coded as J-250 (J, 2 for the C, K ignored, S ignored, 5 for the N, 0 added).

3. Case sensitivity

Your program should be completely case insensitive when computing soundex values.

Input
Input to your program consists of a list of deceased townspeople and a list of current townspeople. The first line of input consists of a single left-justified integer (1<=D<=100) with no leading zeroes indicating the number of deceased townspeople. Lines 2 through D+1 each contain a single surname of the deceased. Line D+2 contains a single left-justified integer (1<=T<=100) with no leading zeroes indicating the number of current townspeople. The next T lines each contain a single surname of a current townsperson.

A surname in either list consists of up to 20 alphabetic characters. All surnames are left justified and contain only mixed-case alphabetic characters. There are no leading, trailing, or embedded non-alphabetic characters on any line containing a surname.

No invalid input will be provided.

Output
Output from your program consists of a single line of output for each deceased townsperson. Each output line consists of a name of a deceased townsperson followed immediately by a colon followed by the list of current townspeople whose soundex values are the same as the deceased person. Names on the list of current townspeople on the output line should be preceded by a single space. If there are no current townspeople whose soundex value matches the deceased townsperson, your program should print " (none)" after the colon. Two examples of exact required formatting (soundex values aside) are as follows:

```
Firstdeceased: Firstcurrent Secondcurrent Thirdcurrent
Seconddeceased: (none)
```

Your program should retain the case of the input surnames for use on the output file.

Note: No output to the screen should be provided, and no extraneous output of any kind should be sent to the specified output file.

Sample Input
```
4
McKeger
Rodriguez
Navalsha
Miller
7
Johnson
Alley
Moliere
MacJoy
Nebalsh
Neopolik
Rothworski
```

Sample Output
```
McKeger: (none)
Rodriguez: Rothworski
Navalsha: Nebalsh
Neopolik
Miller: Moliere
```

Appendix F: Sample Programming Problem 2

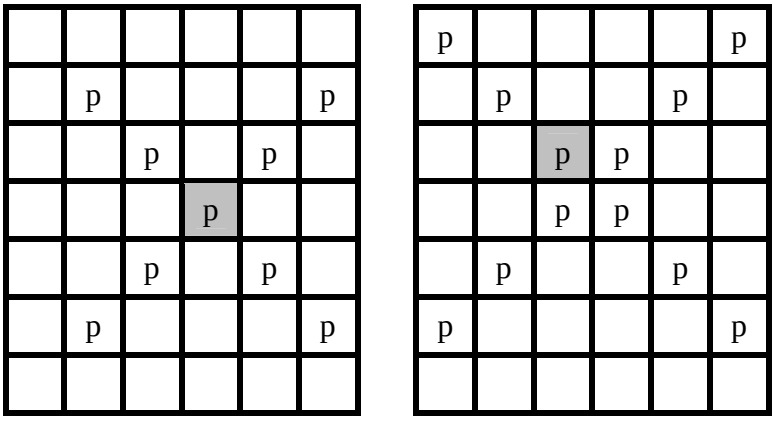
It's the Great Pumpkin Patch Problem – Revised

This problem was copied and adapted from an ACM Programming Contest practice problem used by Southeastern United States division, 2000.

input file: `pumpkin.in` output file: `pumpkin.out`

It's almost Halloween and Linus is going out to the garden to wait for the Great Pumpkin. However, due to diversification, there are lots of other gourds in the garden this year… zucchini (z), yellow squash (y), spaghetti squash (s), and pumpkins (p). Linus has made a map of the garden for convenience. We are entrusting you with this map. Be brave.

Linus hit upon a great idea to attract the Great Pumpkin: he cleared out a large "X" in the garden as a target, where the Great Pumpkin may land. There are no vegetables in the "X" at all. To make the "X" visible, Linus spray-painted it bright, glow-in-the-dark orange. Unfortunately, Linus also marked the target area on the map with the letter "p", for "paint." This "X" will always be the largest **matched** set of diagonal "p" lines in the garden. A **matched** set means that both upper corners (upper right and upper left) of the "X" will be in the same row; both lower corners will be in the same row, both left corners will be in the same column, and both right corners will be in the same column. Both of the following diagrams represent valid "X"'s. The centers are highlighted. If an "X" 's center is a cluster of four squares, the center is considered to be the top left square in the cluster.



The Great Pumpkin is now comparing gardens, and needs you to write a program to tell him how many patches of pumpkins there are - and how big they are - for each of several gardens.

For example, consider the following 10 by 10 garden, with zucchini, yellow squash, spaghetti squash, pumpkins, and paint:

P	z	z	z	z	z	z	z	Z	p
P	y	y	p	z	z	z	z	z	Y
P	p	p	P	s	s	s	s	Y	Y
P	s	s	p	P	s	s	S	Y	Y
s	p	s	p	P	P	s	S	Y	Y
s	s	p	S	p	p	s	S	P	Y
z	P	Z	p	Z	z	s	s	P	Y
p	z	Z	Z	P	Z	s	s	P	Y
Y	Y	Y	Y	P	z	s	s	p	Y
y	Y	y	y	p	p	p	p	P	y

This garden has four patches of pumpkins: one at the top left corner covering 8 squares, one in the top right corner covering 1 square, one in the center covering 4 squares and one near the bottom right covering 10 squares. These patches are shown shaded. Note that in order for a square to be a part of a patch, it must connect with another square in that patch along an edge, not just at a corner (so the squares in the center patch are not part of the patch at the top left corner). The "X" is centered at column 3, row 6. Remember, if an "X" 's center is a cluster of four squares, note the center as the top left square in the cluster.

Also, in case you hadn't noticed, Linus does not distinguish upper and lower case. To him, "P" and "p" are interchangeable.

Input

The input to this program will be a number of different gardens. The first line of the input for each garden will be the dimensions of the garden, r, the number of rows in the garden, and c, the number of columns, where $0 \le r \le 40$ and $0 \le c \le 40$. Following the dimensions will be r lines with c characters on each line. Each of these characters will be an upper or lower case letter representing the type of gourd grown in the square. A 'p' or 'P' will represent pumpkins or paint. A garden with 0 for the number of rows and/or columns indicates the end of input and should not be processed. There will be exactly one identifiable "largest X" of 'p' characters in the garden.

Output

For each garden, output the number of the garden (with the first input set being garden 1), the number of pumpkin patches in the garden, and the size of the pumpkin patches in order from smallest to largest. If there is more than one patch of a given size, print the size as many times as it occurs, then print the coordinates of center of the "X" in parenthesis. the Use the following format:

```
Garden # 1: 4 patches, sizes: 1 4 8 10     X at (3,6)
```

Have a blank line between each line of output.

Sample input

The first input set represents the garden in the garden description.

```
10 10
PzzzzzzzzP
pyypzzzzzy
ppppssssyy
pssppsssyY
spspppssyy
sspsppSspy
zpzpzzsspy
Pzzzpzsspy
yyyypzsspy
yyyyppppppy
44
Pppp
ppPp
Pppp
ppPp
1 7
zZzpzzZ
0 0
```

Sample output (corresponding to sample input)

```
Garden # 1: 4 patches, sizes: 1 4 8 10    X at (3,6)
Garden # 2: 4 patches, sizes: 2 2 2 2     X at (2,2)
Garden # 3: 0 patches, sizes:             X at (4,1)
```

Vita

Peter Klaus Schoenhoff came to the discipline of software engineering after a background and career in music and teaching. The similarities between these fields would probably be the basis for another thesis entirely, but suffice it to say that Pete thinks his background is an advantage in computer science.

Pete holds a Bachelor of Music degree in Music Education from Virginia Commonwealth University where he studied electronic music, and a Master of Science degree in Computer Science / Computer Science Applications from Virginia Tech. He taught music at Bethel, Shawsville, and Belview Elementary Schools in Montgomery County, Virginia between 1994 and 1998, before embarking on computer science studies. As far as Belbin roles go, Pete is a primary Plant and secondary Monitor-Evaluator.

I would like to thank Jesus Christ for His strength, His promises, and His mercy. I would like to thank my wife Holly for her support, love, and boundless patience. (Holly, I love you more than you could know! We finally get to spend time together now!) I would also like to thank Klaus and Linda Schoenhoff, Bill and Carol Bernard, Katherine, Barbara, Robby, and Jennifer for their support and understanding. Perhaps next Thanksgiving, I can bring my guitar instead of my laptop.