Impact of Background and Experience on Software Inspections

Ph.D. Thesis Proposal

Jeffrey Carver

University of Maryland

Abstract

As the popularity of software grows so does the need for quality. In many contexts software inspections have become or are becoming an important part of the quality assurance effort for software products. Software inspections continue to spread across the industry. Inspections are a process whereby software artifacts are examined by a group of inspectors to ensure that they meet some set of quality constraints. Because the performance of different inspectors varies during an inspection, we decided to explore the causes of this variation. Because research has shown that the variations among inspectors can have a greater effect on the outcome of the inspection than the variations in the actual inspection process, we have chosen to focus our investigation on those variations among inspectors. We want to explore what types of defects inspectors having different backgrounds and experience find. There are three steps to this work:

- Develop a useful defect classification scheme
- Develop a list of Background and Experience variables
- Determine how the variables relate to the defect classes

We will be interested in investigating the overall performance of the different inspectors as well as their performance with respect to the defect classes in our defect classification schemes. We will perform our evaluation on data collected from experiments and other empirical studies. The details of these studies will be described as well as how the data will be used. The overall goal of this work is to provide inspection process mangers with a set of heuristics to guide their selection of inspectors for their inspection teams.

1. Introduction

1.1 Problem Statement

As the popularity of software grows so does the need for quality. This situation has driven software engineering researchers to investigate technologies that can aid in improving quality. In many contexts software inspections have become or are becoming an important part of the quality assurance effort for software products [Glass99], [Johnson94], [Laitenberger99]. Inspections are a process whereby software artifacts are examined by a group of inspectors to ensure that they meet some set of quality constraints. Another common goal for inspections is to uncover defects in the artifact. The increasing popularity and perceived usefulness of this technology can be seen by its spread into various industrial settings [Bush90], [Doolan92], [Fowler86], [Grady94], [Russell91], [Votta93], [Weller93]. While early inspection work focused on how inspections could be used to improve the quality of code, since then researchers have expanded inspections to cover not only code but also requirements [Basili96], [Kelly92], [Schneider92], design [Parnas85], [Laitenberger00], [Kelly92], [Travassos99], test plans [Kelly92], and even user interfaces [Zhang99].

Prior to inspections was the idea of a *walkthrough*. The walkthrough could range anywhere from a simple peer review all the way up to a formal inspection of the type we are discussing here. One of the problems with using walkthroughs in a process that is going to be improved is that normally very little data is collected because walkthroughs are less formal and applied differently in each setting. Because of this the efficiency of defect detection is quite variable [Fagan86]. Because we wish to study a more controlled process, we have chosen to investigate inspections instead of walkthroughs.

It has been pointed out that all project environments and products are different in some way [Basili88], even described by one author as “its own unique world” [Ackerman89]. Because of these differences the application of techniques and methods on different project should expect to vary. There are many dimensions upon which software development organizations can differ. For example, the application domain can vary from business software to satellite support software. Another dimension of potential variation can be the level of risk inherent in the project. For some applications, failure may mean only mild inconveniences, while with other applications, it could mean loss of life. While there are some standard methods and practices for performing inspections, in many cases the application of those methods may need to be tailored in some way because of this variation.

A brief description of inspections will be given here with a more complete discussion presented in Section 1.2.1. The basic idea behind an inspection is that members of a software development organization review a software artifact to ensure that it possesses some level or characteristics of quality. An inspection consists of a series of steps. First, the author of the artifact gives the inspectors an introduction and overview of the artifact. Next, the individual inspectors review the software artifact to prepare for the team meeting. After the individuals have inspected the document, they meet together as a team to record the defects that are found. Finally, the document author is given this list of defect so that he or she may repair them in the software artifact [Fagan76].

In most cases, the research surrounding inspections has dealt with improving the inspection meeting. This line of research makes the assumption that individual inspectors already know how to inspect software artifacts on their own. As most software developers are taught only how to write documents but not how to read them, we question this assumption. Research has shown that providing inspectors with detailed techniques can improve their performance over simple ad hoc reading [Shull98]. Other researchers have even questioned the need for an inspection meeting at all [Porter97, Votta93]. In either case, work on the individual inspection either prior to the meeting, or in place of the meeting, has not been studied as much as the inspection meeting has. Because the research has questioned the necessity of the meeting, and because research has shown improvement for individuals when given a specified technique to use, there is a need for aiding the inspector, either before the inspection meeting or in the absence of the inspection meeting, to become better and more efficient at the job of detecting defects in the software artifacts.

In addition to variability of the techniques and methods used within a software development process, the inspectors also come with different backgrounds and experiences. An inspector who is very experienced in a highly technical domain such as mission planning for a spacecraft should be expected to perform differently in a software inspection for that domain than someone who is not familiar with that domain.

Studies of inspections that have been performed confirm the fact that variations among inspectors can have an impact on the outcome of the inspection. Some argue that lack of product knowledge can be beneficial by allowing for a different perspective on the artifact [Dow94]. On the other hand, other organizations employ experts in various aspects of the product to perform the inspection [Doolan92]. Other researchers have found that certain individual inspectors have a effect on the outcome of the inspection [Porter98], [Siy96].

So, not only does the development organization have to worry about how the specific techniques and methods will positively or negatively affect their software development process, they also must have a way to evaluate the background and experience of each of their team members. The way in which these evaluations are done and the results reported by them must give the process manager some guidance on how to staff an inspection.

In order to do this, the relationship between different background characteristics and the performance during an inspection needs to be studied. If a causal effect can be located, then the process manager can be given some guidance for planning their inspections. For example, if we can determine that having high domain experience improves performance during a requirements inspection, then when building the attitude control system of a new satellite, the process manager will want to choose inspectors who are very familiar with the domain to be members of his or her Software Requirements inspection team.

Because a structured review technique is not always used in performing an inspection, we want to investigate not only those inspections that were performed using a technique but also inspections that were done in an ad hoc way, meaning inspectors are allowed to review the document in the way that they think is most effective. Investigating both types of inspections will allow us to say something about inspections in general instead of being limited to only those in which a structured review technique is used.

1.2 Definitions

In this section we will provide more extensive definitions of some terms and concepts that will be used throughout the proposal. We will begin with a more complete discussion of *inspections*. The goal here is to provide a brief overview of the state of the practice. Following this, two specific

1.2.1 Inspections

The concept of inspections is not unique to software engineering. Many fields have some type of inspection of their artifacts. The goals of these inspections can vary from setting to setting, but a general goal is ensuring that the artifact is of sufficient quality to be used by the customer(s) of that document. For example architects have their drawings inspected for feasibility before passing them along to the construction team. In the same way, software development teams have their software artifacts inspected before passing them along (to the next phase of the lifecycle). Since the first work on inspections in 1976 [Fagan76], many variations have emerged.

The main idea of the inspection as defined by Fagan is as follows. Once the author completes a software artifact, which could be a requirements document, design, or code, they submit it for inspection. The inspection consists of multiple steps. The inspection leader first chooses a team of people who will perform the inspection. Then, the document to be inspected is distributed to these team members. The author of the document gives a brief overview and background to provide some context. The team members spend time individually reading over the document and becoming familiar with it. Once all inspectors are prepared, the inspection leader calls for a meeting. At this meeting, the document is read through and defects are uncovered by the inspectors. The defects are collected in a list, which is returned to the document author. The author then fixes these defects, or explains why they are not defects.

Since the initial definition of the inspection process, many variations have been made on it. Changes to the structure of the meeting, variation of the number of meeting participants, elimination of the meeting and many other minor changes have all been suggested as variations to the basic inspection process. But, the main goal still remains to critically examine a software software artifact to uncover defects that may be present so those defects can be removed before the document is used by its customers. For more information see [Fagan76], [Fagan86], [Gilb93], [Porter97], [Votta93]. While much of this research has focused on improving the inspection meeting, some work has been done on improving the individual inspection. Various techniques have been proposed to aid the inspectors when performing the individual stage of the inspection. Two such techniques are explored in the experiments that make up this proposal. Perspective Based Reading (PBR) for the inspection of requirements document will be discussed in Section 1.2.2. Object Oriented Reading Techniques (OORTs) for inspection of an object oriented design will be discussed in Section 1.2.3.

1.2.2 Perspective Based Reading (PBR)

The generic inspection procedure does not provide any assistance to the individual reader to help in the preparation for the inspection meeting. In order to help readers perform the individual inspections for requirements, a technique called Perspective Based Reading (PBR) [Basili96] was developed. This technique was based on the fact that there are various “customers” of the requirements document throughout the software lifecycle. The goal of this technique was to provide each individual inspector with a specific view or perspective from which to review the requirements document, such that all of these customers were represented in the review team. The assumption is that if multiple reviewers can all inspect the document from different perspectives, then when those perspectives are put together we will achieve much better coverage of the document.

The initial perspectives that were defined are tester, user, and designer. For each perspective, the inspector is instructed to create a high-level abstraction of the software artifact that customer would normally create. In the process of creating that software artifact, the inspector will uncover defects. For example, when using the tester perspective, the inspector creates a test plan for the requirements. If the inspector comes across a requirement that is not defined in such a way that a test plan can be created, then they have uncovered a defect.

1.2.3 Object Oriented Reading Techniques (OORTs)

Similar to PBR described in the previous section, inspectors need to be provided with guidance in performing the individual inspection for Object Oriented designs. Because the requirements and the design are supposed to be different representations of the same system, these techniques aim to ensure that all of the information was captured from the requirements in a consistent way. In order to do this, two types of techniques were defined, horizontal and vertical. Horizontal techniques are ones that are used to compare two different design documents. The goal of these techniques is to ensure that the design is internally consistent. For example are all of the methods on the sequence diagrams actually methods in the class diagram. The vertical techniques are used to compare the design documents back to the requirements and the use cases. These techniques are used to ensure that not only is the design consistent with itself, but does it correctly capture the requirements. For example, the inspector might find some functionality described in the requirements that does not appear anywhere in the design. More details on these techniques and their evolution can be found in [Travassos99b], [Shull99].

1.3 Proposal Approach

As we began to examine the results of inspections that had been conducted as part of experiments, we noticed a large variation among the performance of different inspectors. These inspectors were inspecting the same artifact using the same technique, but were performing quite differently. We decided to explore whether or not the inspectors that were performing the best had some characteristic(s) in common. If we could determine what those characteristics were, then we could provide an inspection planner with some valuable information during the planning stages.

This result is not unique to our own research. Other researchers have also found that differences among inspectors can have and effect on the outcome of the software inspection [Sauer00], [Porter97]. In fact they go so far as to argue that the greatest improvements in the performance in an inspection can be seen by improving the effectiveness of the individual inspectors and not by improving the team meeting. This implies that choosing the best members of the inspection team could potentially have a great and positive impact on the outcome of the inspection process.

Realizing that there are differences among inspectors and that those differences are potentially very important, we have decided to investigate those differences and their effect on the software inspection process. In order to do this, we first need to understand what those differences are. This process is discussed in more detail in Section 1.3.2. We also realize that in order to measure how these variables affect the process we also saw the need for defect classifications. The defect classifications are described in more detail in Section 1.3.1. Finally, once we have the important variables and the defect classifications, we must look for relationships between the two. That is, we will be looking for situations where the value of one of the variables has a direct effect on finding defects of a particular class. This will be discussed in Section 1.3.3.

1.3.1 Defect Classification Schemes

In order to understand the types of defects that occur in software artifacts better, researches have developed defect classification schemes. Defect classification schemes, if properly specified, can be useful for repeatability and comparisons across studies and environments [Basili84]. These schemes attempt to group the defects that occur in a particular environment into classes. Below we will discuss work done by others in the area of classifying defects. In each case, the reason the organization classifies their defects is the main driver as to what the defect classifications will look like. The scope and specificity of the classifications described below varies greatly depending on the goals of the organization. After the related work is discussed, our approach to defect classification will be described. Then we will discuss how our approach differs from those presented in the literature. Finally we will present the methodology which we will follow.

1.3.1.1 Related Work

The idea of grouping defects into classes is not a new one. Any organization that wishes to measure their process with the intent of improving must classify the defects that are found using it. This can range from a relatively simple classification of only 2 classes, major and minor, to something much more elaborate as our approach will be. The types of classification that an organization uses will be based on its reason for classifying the defects. The classification will also depend on the lifecycle phase in which the inspection is going to occur.

Taking into account their local environments, researchers have proposed many different defect classifications schemes. Of these, some claim to be useful for documents produced throughout the software lifecycle. Others are directed at documents from a specific lifecycle phase. Most of them have dealt with either defects in requirements documents or defects in source code. Very few of the published classifications deal specifically with defects in design documents. The examples below provided both input to our initial classifications as well as a first check on those newly developed classifications.

Defect Classification Schemes for Requirements

In [Basili81] the authors describe a study of the evolution of a requirements document for the on-board operational flight program for the A-7 aircraft. This is a complex, real-time program. The defects are actual defects that were made in the evolution of the document. The authors here defined a one level

classification scheme with 7 categories. These classes were used to make hypothesis about what kinds of defects were the most common to make.

Clerical – relatively simple problems with the document like typos.
Ambiguity – something in the document has more than one meaning.
Omission – something has been left out of the document.
Inconsistency – two parts of the document are inconsistent with one another.
Incorrect Fact – something in the document is incorrect with respect to the domain.
Information Put in Wrong Section – information included in the document was placed in the wrong section.
Implementation Fact not Included – information necessary for a proper implementation was not given.
Other – defects that do not fall into other classes.

In a paper discussing the benefits that have been gained by applying inspections throughout the software development lifecycle, [Ackerman89] gives a classification scheme for defects found in requirements documents. The authors point out that for an inspection to be effective, the types of defects to be looked for must be specified. Therefore, their goal for classifying the defects is to provide the inspections with a guide to keep them on task during the inspection. While they state that inspections can be used on requirements, design, code, test plans, and test specifications, they only provide the classification for requirements. Their classification scheme has, like ours, two levels, but they do not give a reason for having these two levels. It appears that the top level may be more for grouping and organization while the second level provides the more specific defect detection questions. Here is their defect classification scheme for requirements:

Completeness

1. Are all sources of input identified?
2. What is the total input space?
3. Are there any "don't care" sets in the input space?
4. Is there a need for robustness across the entire input space?
5. Are there any timing constraints on the inputs?
6. Are all types of outputs identified?
7. What is the total output space?
8. Are there any timing constraints on the outputs?
9. Are there sets that are in both the input and output spaces? Is there any state information?
10. What are all the types of runs?
11. What is the input space and the output space for each type of run?
12. For each type of run, is an output value specified for each input value?
13. Is the invocation mechanism for each run defined? Are initial states defined?
14. Are all environmental constraints described?
15. Are sample input/output examples provided where appropriate?
16. Are all necessary performance requirements described?
17. Should reliability requirements be specified? Is an operational profile specified?

Ambiguity

1. Are all special terms clearly defined?
2. Does each sentence have a single interpretation in the problem domain?
3. Is the input-to-output mapping clearly defined for each type of run?

Consistency

1. Do any of the designated requirements conflict with the descriptive material?
2. Are there any input states that are mapped to more than one output state?
3. Is there a consistent set of quantitative units used? Are all numeric quantities consistent?

In [Schneider92], the authors were investigating a variation on the traditional inspection. In this variation, instead of only one inspection being performed on a requirements document, the authors investigate the benefits gained by having multiple, independent inspections of the same requirements document. The requirements document in this study was for a "real-time track control for railway companies". The defects

that were present in the document consisted of a combination of those that the document author, an expert in the domain of railroad engineering, made during the actual creation of the document, as well as some defects added by the researchers to balance the numbers among the different defect classes. They had two high level classes with subclasses in each. Their reason for classifying the defects was to see if their new inspection process affected how inspectors performed in finding any of these classes of defects.

Missing Information

Missing Functionality or Missing Feature – Information describing internal behavior of the system has been omitted

Missing Interface – Information about how the system communicates with objects outside the system has been omitted.

Missing Performance – Performance specifications have either been omitted or described in an untestable way.

Missing Environment – Information about the external hardware/software/databases/personnel has been omitted.

Wrong Information

Ambiguous Information – An important term has not been defined or has been defined in a way that has multiple interpretations.

Inconsistent Information – Two parts of the document contradict each other.

In their paper, [Porter94, Porter95] discuss a study on scenario-based reading, where each inspector executes a series of steps to discover a particular class of defects. The domains of the requirements inspected were a cruise control system and a water level monitoring system. The defect classification scheme used here relied heavily on the work of [Basili81] and [Schnedier92], discussed above. In the experiment described in the paper, the authors are comparing ad hoc reading, to checklist reading, to their new technique of scenario-based reading. Scenarios were developed so that each one targeted a specific class of defects. The reason for using a defect classification here was two-fold. First, because they were comparing three detection methods, they needed a constant basis for comparison. Secondly, because scenarios were developed to target specific classes of defects, they needed a classification scheme that was useful for that purpose. Again, the defect classification has two levels, with the first one being very abstract, and the second one being more useful in actual defect detection. Both the top level and the second level draw greatly from the two studies mentioned above. These authors combined those classification schemes into one that useful for their environment.

Omission

1. **Missing Functionality** – Information describing the desired internal operational behavior of the system has been omitted.
2. **Missing Performance** – Information described the desired performance specification has either been omitted or described in a way that is unacceptable for acceptance testing.
3. **Missing Environment** – Information describing the required hardware, software, database, or personnel environment in which the system will run has been omitted.
4. **Missing Interface** – Information describing how the proposed system will interface and communicate with objects outside the scope of the system has been omitted.

Commission

1. **Ambiguous Information** – An important term, phrase or sentence essential to the understanding of the system has either been left undefined or defined in a way that can cause confusion and misunderstanding.
2. **Inconsistent Information** – Two sentences directly contradict each other or express actions that cannot both be correct or cannot both be carried out.
3. **Incorrect or Extra Functionality** – Some sentence asserts a fact that cannot be true under the specified conditions.
4. **Wrong Section** – Essential information is misplaced within the document.

In their paper [Basili96] describe an empirical study of Perspective Based Reading (PBR). The goal of the study was to determine if PBR was more or less effective at defect detection than simple ad hoc reading. PBR is described in Section 1.2.2. In their study, they used a different defect classification scheme from the ones mentioned above. There are many similarities, but there are some differences. They created a one level defect classification:

Omission
Incorrect Fact
Inconsistency
Ambiguity
Extraneous Information

In [Doolan92], the author describes experiences with using the *Fagan Inspection* for requirements in the Seismic Software Support Group (SSSG) at Shell Research. The SSSG is responsible for a software package of over 2 million lines of Fortran code. The package has over 250 separate high-level geophysical functions for a variety of platforms. A new release for the package is provided approximately every 6 months. The SSSG places high value and emphasis on the quality of its product. This is the reason that they chose to investigate the usefulness of the "Fagan Inspection". In order to gauge this effectiveness, they judged the cost savings for each hour invested in the inspection. Therefore their classification scheme is only based on the severity of the defects. The three classes of defects are **Minor**, **Major**, and **Super Major**. Based on their history, they equate 25 minor defects to equal one major defect, and one super major defect is equivalent to 10 major defects. They used this information to determine that for each hour invested in software inspections, they saved nearly 30 hours later in the software lifecycle.

Defect Classification Schemes for Design

Less work has been done on classifying design defects than has been done for classifying defects from the other lifecycle phases. We have done some previous work on defect detection in Object Oriented designs [Travassos99b], [Shull99]. This work has so far been demonstrating the feasibility of reading techniques for Object Oriented Design inspections, and improving those techniques based on the results of empirical studies. The defect classification scheme that has been used thus far in this work grew out of the defect classification scheme for PBR from [Basili96], mentioned in the previous section. The names of the defect classes are the same, but the descriptions are tailored to make sense for design. Again, this design classification is only one level, and it is as follows:

Omission – One or more design diagrams that should contain some concept from the general requirements or from the requirements document do not contain a representation for that concept.
Incorrect Fact – A design diagram contains a misrepresentation of a concept described in the general requirements or the requirements document.
Inconsistency – A representation of a concept in one design diagram disagrees with a representation of the same concept in either the same or another diagram.
Ambiguity – A representation of a concept in the design is unclear, and could cause a user of the document (developer, low-level designer, etc.) to misinterpret or misunderstand the meaning of the concept.
Extraneous Information – The design contains information that, while perhaps true, does not apply to this domain and should not be included in the design.

Another example of a design defect classification scheme comes from [Parnas85]. In this paper they describe a process called Active Design Reviews. The goal of this type of review is to use multiple inspectors and give each one a slightly different focus. In order to help focus the inspectors' attention and to take advantage of different skills and knowledge, they developed a classification scheme. The purpose of the classification was as a guide for design the reviews to find as many defects as possible. It is as follows:

Inconsistencies – Places where the design won't work.
Inefficiencies – Places where the design imposes a barrier to efficient programming or use.
Ambiguities – Places where the design specification may be interpreted in several ways.
Inflexibilities – Places where the design does not accommodate change well.

Defect Classification Schemes for Code

Defect classification schemes for code are the more common compared to those for the other lifecycle phases. There have been many studies published in which a classification of code defects has been provided.

In a paper investigating testing vs. code reading [Basili87] provides two defect classification schemes for defects that can be found in code. Four programs were studied, each from a different domain. Two different languages were used for those programs, Fortran and Simpl-T. The four programs were 1) a text processor, 2) a mathematical plotting routine, 3) a numeric abstract data type, and 4) a database maintainer. The two schemes are:

1) **Omission** – Result from the programmer forgetting to include something in a segment of code.
2) **Commission** – Result from an incorrect segment of existing code.

And

1) **Initialization** – Improperly initializing a data structure.
2) **Computation** – Cause a calculation to evaluate the value for a variable incorrectly.
3) **Control** – Causes the wrong control path to be taken for some input.
4) **Interface** – Passing an incorrect or argument or assuming that an array passed as a parameter was padded with blanks.
5) **Data** – Result from incorrect use of a data structure.
6) **Cosmetic** - Clerical mistakes when entering a program.

Each defect can fall into one of the classes from the first set and one from the second set. The goal of these schemes was to distinguish among the reasons that programmers make faults in software.

In [Russell91], the author discusses the results of a study performed at Bell-Northern Research. In this study, he investigated inspections of 2.5 million lines of code. The code was for programs in the telecommunications domain, specifically digital and packet switches. The code, which comes from many different products, was developed and inspected over a period of about 10 years. The goal of the study was to determine if code inspections were more or less effective than testing. To determine the types of defects that were more or less likely to be found by inspection, they created a small classification system that has 3 classes:

Wrong Statements
Extra Statements
Missing Statements

While not as complete or extensive as some of the other classification schemes, this one does provide us with a high level distinction between defects.

Another paper dealing with a defect classification scheme for code defects is presented in [Chillarege91]. Their work is based on the Orthogonal Defect Classification for Design and Code defects proposed by IBM [IBM99], which will be discussed in a later section. Their overall goal is to understand if there is a measurable cause-effect relationship between defect type and development effort. As that is a difficult goal to accomplish, this paper deals with a sub-goal of that, namely, the relationship between defect type and reliability of the final product. As their measurements are based on grouping defects into classes to determine if any class has a greater impact on the reliability of the product, they had to create a defect classification scheme. Because their scheme was meant to measure the relationships between large defect classes and the output of the development process and not meant to improve the individual steps in an inspection, they did not need a two level classification scheme as we will. Their classification is as follows:

Function – Occurs when a functionality was either omitted or was implemented incorrectly.

Checking – Occurs when a check such as a boundary or condition check is either incorrect or omitted.

Assignment – Occurs when an incorrect data value is assigned to a variable after initialization, or a value is not assigned when it should be.

Initialization – Occurs when a default value or an initialization value is either set incorrectly or not set at all.

Documentation – Occurs when an incorrect description of the program is given, or the description is omitted.

Similar to the classification scheme described by [Basili87], each one of the above classes can be divided into defects that are caused by **Missing Information** and those caused by **Incorrect Information**.

In [Knight93], the authors discuss another variation on the basic inspection technique. They propose something called *Phased Inspections*. In this model, there are many small inspections, each of which checks the work product for some desirable property. One of their arguments is that when inspectors look for all defects at once, they are overwhelmed. In order to help with this, they have developed a classification scheme for defects in code, and only have inspectors focused on a few during each "phase" of the inspection. Their classification is as follows:

Errors in Logic
Errors in Computation
Errors in Tasking Structure
Unacceptable Inefficiencies
Omitted Functionality

Defect Classification Schemes for multiple lifecycle levels

In addition to the defect classification schemes described above that are relevant or tailored for the inspection of artifacts from one phase of the lifecycle, there have been some schemes developed that apply to inspections of artifacts from multiple phases of the lifecycle. Below we will discuss some of these.

In [Fagan76], the initial description of the inspection process for software work products, the author describes the development of inspections. The usefulness and benefits of inspections are also demonstrated through analysis of the defect data collected at IBM. This data analysis was used to measure and control the inspection process. The domain of the software that was inspected is operating systems. The programming languages investigated include PL/S, Assembler, COBOL and FORTRAN. Fagan developed three different defect classification schemes. Each defect falls into one class in each of the three schemes. So, each defect has three classes. Two of the classification schemes are consistent for all inspections. The first classification scheme breaks defects into classes based on whether they are **Major** or **Minor**. The second classification scheme puts defects into classes based on whether they are caused by **Missing Information**, **Wrong Information**, or **Extraneous Information**. The defect classification schemes given for code defects and for design defect are similar but not identical, so each will be presented below. First, for design defects the classification scheme is:

Logic
Test and Branch
Data Area Usage
Return Codes/Messages
Register Usage
Module Attributes
External Linkages
More Detail
Standards
Prologue or Prose
Higher Level Design Doc.
User Spec.
Maintainability
Performance
Other

And, for code defects, the classification scheme is:

Logic
Test and Branch
External Linkages

Register Usage
Storage Usage
Data Area Usage
Program Language
Performance
Maintainability
Design Error
Prologue
Code Comments
Other

In [Fowler86], the author discusses AT&T's success with using inspections. These inspections occur at multiple points throughout the development lifecycle. While the exact domain of the projects under study is not given, it can be assumed that they are in the larger domain of telecommunications. The reason given in this paper for the necessity of a defect classification scheme is to provide a framework for the analysis of the inspection data for process control and improvement. AT&T's approach is a bit different from the others mentioned in this section. While they do have one defect classification scheme that is used across all lifecycles, **Missing**, **Wrong**, or **Extra**, they do recognize and state the need for a defect classification scheme to be developed for each type of inspection within a development organization. They provide examples of the defect classes from some of their inspections. The examples include: **Logic defect in Code**, **Interface Defect in Design**, and **Missing Menu Screen in Requirements Document (human-interface defect)**.

In [Bush90], the author discusses JPL's experience with implementing *Fagan Inspections*. JPL was experiencing some difficulty with some of its systems, and realizing that the impact of software on their work would only increase as time passed, they decided to implement *Fagan Inspections* as a way to help alleviate the problem. At JPL, the lifecycle phases included System Requirements, Subsystem Requirements, Functional Design, Software Requirements, Architectural Design, Detailed Design, Source Code, Test Plan and Test Procedures. Inspections were carried out at each of these lifecycle phases. They developed a one level defect classification scheme that was used for all of the inspections. Their defect classification included the following classes (not complete):

Clarity
Correctness
Completeness
Consistency
Compliance with Standards
Data Functionality
Level of Detail
Maintainability
Performance
Reliability
Traceability

The Orthogonal Defect Classification for Design and Code [IBM99] developed by researchers at IBM Research is a major attempt at developing a classification scheme. They classify the defects based on two different pieces of information. The first is when a defect is opened or first discovered. At this point, the inspector generally knows what caused them to find the defect. The second is when the defect is closed. At this point, the nature of the fix is known. The classification presented here is used in conjunction with the activities that trigger the defect detection to determine when particular types of defects should be found, and to allow for the evaluation and improvement of the inspection process. The classification scheme is as follows:

Assignment/Initialization – Values assigned incorrectly or not assigned at all.
Checking – Errors caused by missing or incorrect validation of parameters or data in conditional statements.

Algorithm/Method – Efficiency or correctness problems that affect the task and can be fixed by (re)implementing an algorithm or local data strcutre without the need for requesting a design change.
Function/Class/Object – The error should require a formal design change, as it affects significant capability, end-user interfaces, product interface with hardware architecture, or global data structure(s).
Timing/Serialization – Necessary serialization of shard resource was missing, the wrong resource was serialized, or the wrong serialization technique was employed.
Interface/OO Message – Communication problems between modules, components, device drives, objects and functions via macros, call statements, control blocks, or parameter lists.
Relationship – Problems related to associations among procedures, data structures and objects.

1.3.1.2 Our Proposed Approach

In each case described above a defect classification scheme has developed which is based upon the goals of the particular organization. While many of the organizations discussed above have as their goal to improve the quality of their software, our goal is improve the quality of the inspection process. Because our goal was different from the ones presented above, we decided to create a new set of defect classification schemes. Our goal really has two parts. One is to evaluate the relationships between the defect classes and the background and experience variables. The other goal is to improve the inspection process.

Because of the two goals, we decided that it would be necessary to develop defect classification schemes that had two levels of specificity. The first level should be general enough to capture the major kinds of defects. The idea here is to have a small number of general classes. This will allow for a large numbers of defects to appear in each class. We can then perform statistical analysis to study how those classes of defects behave in relation to the background and experience variables, which are discussed later. The second level should break each of the general classes up into more specific defect types. These defect types should not be specific to any one application, but should be specific enough and concretely described such that it is easy to see exactly what has gone wrong. The goal here is to allow for process improvement. Each of the specific defects could be a step that needs to be added to a reading technique, or an item that should be added to a checklist to improve the performance of the inspection.

1.3.1.3 Methodology

Figure 1 below represents the methodology that will be used in the development of the defect classification schemes. We will begin with the artifacts used in (Loan Arranger and Parking Garage) and data collected from an experiment which will be described in Section 2.1. We will use the defect lists for both the requirements documents and the Object Oriented design documents. This distinction will be described in the description of the experiment. In the experiment, the requirements and design documents were seeded with some known defects before the experiment (**Seeded Defects**). In addition to the defects that were

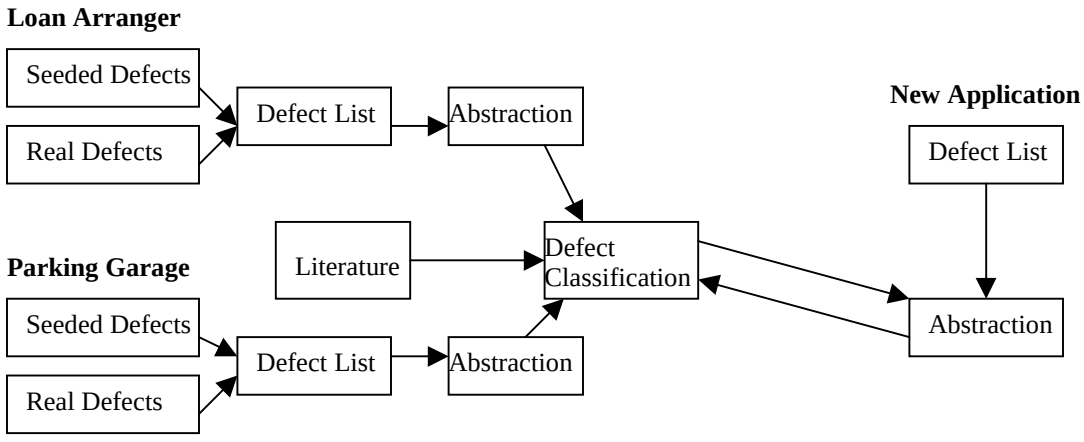

Figure 1 – Methodology for Defect Classifications

seeded, the subjects discovered some other defects that were injected during the normal creation of the documents (**Real Defects**). We call these defects "real" defects, because they were not inserted purposely by the experimenters. Once we have created the defect list, then we must abstract away the domain specific information from each defect, so that the defect classifications will be useful across domains and not for only one specific problem domain. With input from the literature we will then create the initial defect classification schemes. This will take us to the "Defect Classification" box in Figure 1.

The procedure for refining these defect classification schemes, the two boxes on the far right side of the diagram in Figure 1, will be much the same as that which was used to initially create them. For each new experiment or set of defects that we investigate, the defects will be abstracted so that they are no longer domain specific. Next, we will try to group the new defects into the classification schemes that we have already defined. If the new defects fit into the schemes, then we do not have to modify the schemes. If they do not fit into the schemes, then the schemes must be modified in some way to deal with the new type of defect that has been uncovered.

1.3.2 Background and Experience Variables

In addition to classifying the defects, it also important to examine the characteristics of the people who are performing the inspections. We need to determine if there are characteristics of an inspector that can either be helpful or harmful in doing an inspection. If we can determine the most beneficial set of characteristics, we can provide inspection managers with some guidance when choosing the most desirable people for their inspection team.

When we talk about background and experience variables, we are really referring to the characteristics, experience and background that an individual inspector brings to the project. This could be things like knowledge of the application domain, amount of training in a particular method or techniques, or even previous experience with the method or technique to be used or one that is similar. Below we will describe research done by others in the area of background and experience variables. After the related work, we will present our approach to the problem. Following that will be our methodology for studying the problem.

1.3.2.1 Related Work

It makes sense that the background and experiences of an inspector will have an effect on the way he or she performs an inspection. This fact has been recognized by many researches that have studied inspections and most researchers take this fact into account when planning a study or experiment [Parnas85], [Porter97], [Gilb93], [Sauer00]. Most of the time the researchers do not specifically address this fact, but often their observations and conclusions point to this background as a factor.

While others have not studied the particular problem that we are investigating, that of creating a list of relevant background and experience variables, they have understood the impact of an inspectors' background and have either cited it as basis for a conclusion, or used it as an input when designing the experiment. Even though these results themselves are not definite proof of which variables may or may not be important, they do provide a basis for the creation of our initial list to study. Below are brief discussions of the studies, and information about the potential background and experience variable or variables identified in each.

In a paper investigating the variation in software inspections, [Porter98] discuss their findings that the author, the reviewer, and the code unit under inspection all had more of an impact on the output of code inspections than that inspection process did. They observed that different reviewers found different numbers of defects in the same code unit. These reviewers also tended to focus on different kinds of defects. They also noted that because of these differences in reviewers, their combination into teams caused different teams to behave differently in the inspections. In doing their statistical analysis, they found that the presence of two different reviewers had a significant impact on the output of the inspection. They suggest that improving inspection effectiveness could be done by selecting the right reviewers, or by studying the background and characteristics of the best reviewers.

In a paper discussing their experiences with using inspections, [Ackerman89] says that because the creation of software is highly individualistic and each piece of software is in some sense its own little world, it is

difficult for an inspection team to uncover defects without understanding this world. Because of this, he suggests using inspectors who are building "similar worlds". Meaning, choose inspectors who are working on a similar project or who are working on interfacing projects. This is implies that domain knowledge may be an important variable. He also states that when they train inspectors, they always provide a practice inspection. This shows that both training and experience with inspections could be important variables.

In his paper, [Doolan92] discusses the use of inspections in the Shell Software Support Group (SSSG) at Shell Research. Because of the nature of the domain, the SSSG has people who are both experts in software engineering and experts in geophysics. The inspectors were chosen from four different places; 1) geophysical research group, 2) experimental seismic processing group (responsible for evaluating and testing new software), 3) production processing group making full-time use of the package, and 4) SSSG. The inspectors were chosen from those locations, because the SSSG felt that the expertise of people in each of these groups was important. This implies that domain knowledge is an important variable.

In a paper discussing the application of inspections at NASA, [Dow94] investigates whether or not the pool of potential inspectors can be expanded by using people with less domain knowledge. The goal of the experiment is to determine if product knowledge is necessary when performing a code inspection. The authors investigate the assumption that only people with detailed product knowledge can be good inspectors of a segment of its code. The software that was inspected was part of a robot being developed for NASA. The robot was to be responsible for inspecting the tiles on the Space Shuttle after it returns to Earth. The code segment that was inspected converted signals from a joystick into motor control commands. It was 525 non-comment lines of C code. This is a more specific instance of domain knowledge variable; more specifically it is product knowledge. They found the use of inspectors who lacked product knowledge allowed the artifact to be viewed from a different perspective and provided a more complete inspection. It was also found that when there is an "oracle" or document that can be used for comparison, product knowledge was not required for an inspector to be effective.

In a follow up to his initial work on inspections, [Fagan86] explains advances that had been made in the first ten years of inspections. He claims that most inspectors of code are programmers who are involved in the project. This would be people who were both domain experts, as well as experts in the language and somewhat familiar with the problem being inspected. On the other hand, he suggests that the moderator, one role in an inspection meeting, should be someone who is not from the project being inspected, but from a similar product. The reason for this is to lead some objectivity to the inspection. This is another indication that the level of **Domain Knowledge** may be an important variable. It is also an indication that knowledge of the language, or more generally, **Software Development Experience** is an important variable.

[Fowler86] discusses implementation of inspections at AT&T. See Section 1.3.1.1 for a more complete description. While inspections are used in all of the lifecycle phases, they only discuss the background qualifications of the inspectors when performing the requirement inspection. The author suggests that, similar to an argument made by [Dow94], when reviewing a requirements document, the inspector must compare the information against a more abstract picture or idea in their head rather than a concrete specification from a previous lifecycle phase, they need to have high expertise and knowledge. High expertise translates into **Software Development Experience** as well as **Inspection Experience,** while high knowledge translates into **Domain Knowledge**.

1.3.2.2 Our Proposed Approach

In each of the cases described above, the background and experience variables that have been identified were mostly done as a side effect of something else that was being studied. We want to specifically study these variables. Our approach will be to create a list of background and experience variables that accurately characterizes the differences among inspectors. We want each variable to be important and useful in the differentiation of inspectors. Because of this, each variable in this list should have easily differentiable values. Also, in the collection of inspectors under study, we should see the whole range of those values. For example, if we have defined a background or experience variable, and discover that all of the inspectors have the same value for that variable, then it is not a very interesting variable. An initial list

of experience variables has been developed as a staring point. These variables will be described in Section 2.3.3. This list provides a starting point, it is not assumed to be complete or exhaustive.

1.3.2.3 Methodology

Figure 2 below shows the process that we will use to develop the list of Background and Experience Variables. We will begin with an examination of the literature to determine if other researchers have identified potential variables that we could work with. We will also draw from our own personal experiences with participating and observing inspections. By combining the knowledge gained from the literature and our own experience, we will create an initial list of Background and Experience Variables. Each variable in that list will be evaluated to determine if it should remain in the list. Below we will describe how each variable will be evaluated. In addition to the individual evaluation of each variable, the list will be evaluated as a whole to determine if it is complete. If it is found to be lacking, then it will be expanded to include the variable that was missing.

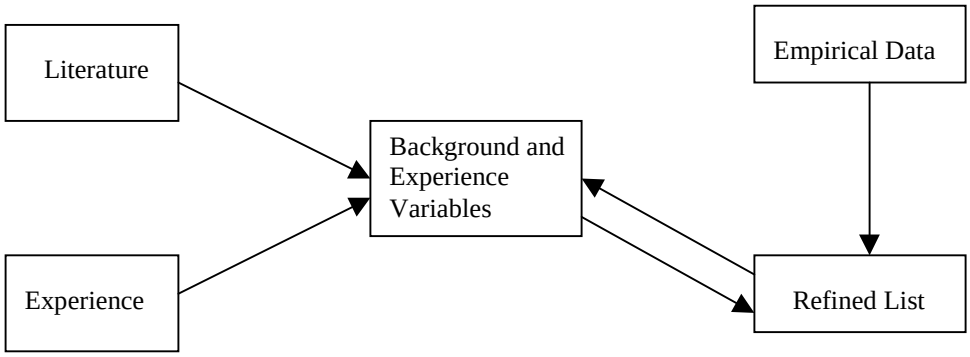

Figure 2 – Methodology for Background and Experience Variables

In order to develop a set of questions with which we can evaluate the individual variables we must first identify what those variables are. This is part of the completed work that will be discussed in Section 2.3. But, in order to present the complete methodology here we will provided the initial variable list along with the questions for each variable.

For each variable we present a rich set of questions to help us understand the variable. We feel that all of these variables are interesting and should be explored. Because of limited data, we will most likely not be able to answer all of the questions that follow. Based on the opportunities that we have to acquire data, we will address as many of the questions as possible.

Environment Variables

EQ6 What is the environment like in which the inspection is being performed?

For each environment the goal is to be able to characterize it in such a way that the distinctive features stand out. We have to be able to determine if there really are differences between environments. This could be a simple thing like the location of the inspection (is it in an office or a dedicated and quiet conference room?). It could also be something more complex such as the general attitude of the management towards inspections, e.g. are inspections viewed by management as a beneficial process or as a waste of time? Or, is the practice of inspections encouraged and rewarded?

EQ7 Are there environmental constraints that affect how the inspection is being performed?

Whereas the previous question deals with the day to day operations and the general working environment of the development organization, this deals more specifically with the requirements or constraints that are put on the inspection process by the management. For instance, this could include things such as the amount of time that is allowed for inspections. It could also include specific procedures that must be followed regardless of the inspection being performed, such as a standard defect report form that is required to be used throughout the organization.

EQ8 Can these constraints (EQ5) be changed?
Here we have to determine how rigidily the constraints from the previous question are enforced. For example, if there is standard defect report form for the entire development organization, would it be allowable for the process team to use their own specialized form as long as all of the data that is on the standard form is collected? If this is allowable, then it will affect what types of inspection techniques can be used and how they can be evaluated within this particular organization.

Training

EQ9 How does the training affect the application of the techniques?
Here we want to determine what effect the training has on the application of the technique. It makes sense that if there are differences in the training there could potentially influence the effectiveness of the inspection. We want to know what aspects of the training the users of the technique felt were important, which aspects that they found to be useful when they received training, and which important things were left out of the training.

EQ10 Are there ways to train more or less effectively?
This is a more specific piece of EQ7. Here we will be looking at qualitative results to surveys and questionnaires to determine what the users of the techniques felt were effective. We will also be looking for suggestions from them as to what we could change in the training to make it more effective. This is important because we will be looking at the feedback of the inspectors that actually received the training and the executed the techniques. This will provide some real concrete reactions to the training and the inspectors ability to use the techniques.

EQ11 Are there other issues that should be considered when planning the training?
In addition to the amount of time spent on training, and the individual that is doing the training, we want to determine if there are other things that influence the success or failure of a training session. This could include things like the time of day, the location, or other things that we have not thought of.

EQ12 If the training unequally emphasizes one aspect of the technique, does this skew the results when the technique is applied?
Here we are interested in understanding better what influence the training has on the output of the inspection. If there are parts of the inspection process that are not performed well, then possibly emphasizing those more in the training could positively affect the inspection process.

Domain Knowledge

EQ13 Does experience in a domain make the readers more or less effective? Is the technique too detailed for experts? Is the technique not detailed enough for novices?
Here we want to determine whether or not having domain knowledge is a good thing. There are two ways of thinking about this. The first says that the more domain knowledge an inspector has the better they will do in the inspection because they are familiar with the domain of the problem. The other says that because an inspector is not familiar with the domain, they will perform better in the inspection because they won't be biased by their previous knowledge. In conjunction with this we want to know if the techniques need to change based on the domain knowledge of the inspector. Does someone with a lot of domain knowledge need more or less guidance when doing an inspection? What about someone with no domain knowledge?

EQ14 Does the technique affect how domain experts perform?

This is similar to EQ11, but here we are asking whether or not a technique is helpful at all to an expert. The issue here is that when someone is an expert in the domain, they are generally comfortable with doing things a certain way, and effective using their own way. We want to see if having a technique or guide for the inspection will simply get in the expert's way and cause them to become less efficient, or if the technique will give them a new way of looking at the same thing and improve their efficiency.

EQ15 Does the technique affect how domain novices perform?
Again, this question is similar to both EQ11 and EQ12. The difference here is we want to know what affect an inspection technique has on inspectors who do not know a lot about the domain. We want to know if having a detailed procedure helps a novice since they do not generally know what they are looking for. The technique should be able to provide them with some guidance and enable them to uncover defects that they would not have found using some type of ad hoc process.

Software Development Experience

EQ16 Does experience in a specific development technology (e.g. Object Oriented Design) make a reader more or less effective? Are the techniques too detailed for experts? Are the techniques not detailed enough for novices?
Many inspection techniques use specific development technologies such as Object Oriented Design. What we are interested in here is determining how much of that knowledge is necessary when using and inspection technique at that lifecycle phase. For example, if someone is going to inspect and Object Oriented design using some type of reading technique, is it necessary that they be very familiar with Object Oriented design? Likewise, if a requirements inspection technique uses a specific testing technique to aid in defect detection, such as equivalence partition testing, is it sufficient to introduce the testing technique to the inspector during the training, or is necessary that the inspector already be familiar with the technique?

EQ17 Does the technique affect how development experts perform?
In the same way that we wanted to know how domain experts were affected by a reading technique, we want to know how Software development experts are affected. This means that if we are going to use testing in our requirements inspection, is it harmful to the expert to give them a specific testing technique to use? Would it be better to tell them to inspect the requirements as if they were going to use them for testing, and allow the expert to use whatever type of testing technique that they felt was appropriate.

EQ18 Does the technique affect how development novices perform?
The main point here is determining whether or not the techniques are of sufficient value that a software development novice could improve their performance on an inspection. Generally the time of a novice is cheaper than that of an expert, and if the techniques could sufficiently aid the novice in finding defects, then the cost of the inspections could be greatly reduced.

Experience in another Reading Process

EQ19 Do readers have experience?
Here we have to evaluate the experience that the inspector has. We must determine what is relevant experience and what is not relevant experience.

EQ20 Does this knowledge, or lack of knowledge, help or hinder them in the application of the reading technique?
We are interested in determining the value of inspection experience. We want to know if having performed inspections before, using techniques similar to the new one being introduced, is going to help the inspector or not. If the previous experience is helpful in using a new technique, then this will give the process manager some guidance as to which people they should choose for the inspection team when trying out a new inspection process.

Experience in this Reading Technique

EQ21 Do the readers become more or less effective after prolonged exposure to the technique?

While EQ17 and EQ18 were concerned with how previous inspection experience would affect the learning of a new technique, this question deals with the learning curve on a new technique. If we find that inspectors dramatically improve their performance by using the technique multiple times, then this would suggest two things. First, that the inspectors should be given chances during the training to use the technique once or twice so that when the performed their first real inspection they had already conquered most of the learning curve. Secondly, it would suggest that when forming an inspection team, the process manager should favor people who were familiar with the inspection technique that was going to be used over those that were not familiar with it.

EQ22 Do the readers tend to follow the techniques closer the longer they use them or the more experience they have with them?

This is a process conformance issue. We are interested in knowing not only whether the inspectors do better or worse the longer they use the techniques, but also how closely they follow them. As inspectors use a technique more they will begin to better understand the technique and what the technique helps them to do. If it becomes evident that as inspectors become more familiar and better understand a particular technique and its goals, that they tend to follow it less rigidly, then the techniques should evolve to capture this fact. The techniques should be able to become less rigid and more of a set of guidelines.

EQ23 Do the techniques need to evolve to become less restrictive as readers become more expert?

Like the previous question, we want to know what the effect of prolong exposure to a technique is. In order to keep the technique effective and useful to the inspector we need to know how the technique must change. Because a first time user does not yet understand the process or the goals of the technique, they need a more formalized rigid technique to follow. On the other hand, as an inspector becomes more and more familiar with the technique, they better understand the goals of the technique and how each step contributes towards accomplishing those goals. That reader is in a better position to reorganize the steps or perform them in a slightly different way that they feel still accomplishes the same goal. We want to have techniques that can remain useful even after the readers become experts. In order to do this we must understand how experts view the techniques and how those techniques should change to meet their needs.

Native Language

EQ24 Is the reader familiar with the language that the documents and instructions are written in?

Here we will investigate the variation caused by inspectors who are not native speakers of the language of the document being inspected. This would be especially important during a requirements inspection where the document is often written in natural language. But, it could also be an important issue during any inspection in which a technique must be followed. How well the inspector understands the technique could have a large impact on how they execute it.

Process Conformance

EQ25 Other than experience and domain knowledge, are there other ways to measure what causes readers to either follow or deviate from the reading technique?

Often, inspectors do not follow processes or techniques as closely as we would like them to. Many times, the inspector thinks that they can be more effective by using their own techniques. What we want to determine is the cause of the deviation from the process. In the previous sections we were concerned with how experience, both software development and inspection technique, and domain knowledge affected process conformance. Here we want to know what other factors cause inspectors to either closely follow a process or technique or to deviate from it. We also are interested in how to quantify or measure those factors.

EQ26 Is it always a bad thing for readers to deviate from the technique?

From the point of view of the researcher, deviation from the process is generally a bad thing, because when the researcher doesn't know exactly what process was followed, it is difficult or impossible to draw conclusions about the effectiveness of the technique. But, from the point of view of the inspector, it might be a good thing to deviate from the technique. If the inspector has found from past experience that they have a better way to accomplish the same goal, then they will want to use that better way. From the point of view of the development organization, if the individual inspector is more efficient using his or her own way, then the development organization has a difficult time arguing with that.

EQ27 Is it possible to modify the technique so that it gives a "bounding box" instead of a "straight jacket"? Would this make the inspector more or less effective?
Many inspectors see well-defined processes or techniques as a "straight jacket". As we have said in previous sections, this may not be a problem for a novice, because they don't know any other way to do it. But, for a more experienced user, they can see the process as a type of "straight jacket" that won't let them do what they know in their mind to be a more effective or better process. By creating a "bounding box", we would give the reader a set of high level guideline, such as goals to be accomplished or specific types of defects to look for. In this case, we want to know if this "bounding box" approach would improve the performance of the inspectors by giving them the freedom to do what they know, or if it would hurt the performance because they lose the well-defined process that has been studied and evolved into and effective technique.

EQ28 When the technique is not followed, can we determine the reason? (e.g. Do the readers think that they do not need it?)
In the earlier question in this section we have tried to address what factors may cause an inspector to deviate from the given inspection process. In some cases this deviation is done on purpose, as in the cases where the inspector thinks that he or she has a better way of doing something. Or, the deviation could be done by accident, the inspector has used the process so many times they don't look at the steps each time through and he or she forgets one or does them out of order. What we want to figure out with this question is when the process is deviated from, is it on purpose or by accident. We also want to know what reasoning the inspectors have for not following the process. The results of this investigation should help support the linkage between the other variables and success of inspections. For instance, if a highly experienced tester tells us that he or she didn't follow the technique of inspecting from a tester's point of view because he or she felt that their way of testing was more effective. This would lend support to the fact that development experience has some effect on the inspection process.

1.3.3 Relationships between the Background and Experience Variables and Defect Classes

After determining both the useful defect classification and the list of background and experience variables, the next step is to determine how they relate to each other. Here we want to investigate the relationship between the values of the different background and experience variables and whether or not inspectors find various classes of defects. To do this analysis, we will use the more general defect classification scheme mentioned earlier. It is the one that has smaller number of defect classes.

1.3.3.1 Related Work

Many of the papers presented in Section 1.3.2 also contained information suggesting a linkage between the variables they identified and the output of their inspections. In this section we will discuss the findings of other researchers as a starting point for our work. We are interested in determining whether a relatively high or a relatively low value for any of the identified background and experience variables can be shown to have a direct impact on some portion of the output of the inspection process.

[Chaar93] and [Chaar95] identify "defect triggers", also discussed in the Orthogonal Defect Classification (ODC) [IBM99]. These are not the defects themselves, but the situation that helps to uncover the defects. They also describe what they call the "skill level" needed by an inspector for each of the triggers. The triggers are:

1) **Design Conformance** – defect can be found by comparing design or code to the specification from the previous lifecycle stage.
2) **Understanding Details** – defect is found while trying to understand the structure or operation of a component.
   - **2.1 Operational Semantics –** defect uncovered while looking at flow of logic.
   - **2.2 Side Effects** – defect uncovered while looking at a side effect of an action.
   - **2.3 Concurrency** – defect uncovered while looking for serialization for controlling a shared resource.
3) **Backward Compatibility** – defect uncovered using knowledge of previous versions of the product to determine an incompatibility between the product under review and the existing version.
4) **Lateral Compatibility –** defect uncovered using broad knowledge to discover an incompatibility between the product under review and another system with which interface is required.
5) **Rare Situation** – defect uncovered using both product knowledge and domain knowledge to discover some system behavior that has not been described.
6) **Document Consistency/Completeness –** defect uncovered because of incomplete or inconsistent document.
7) **Language Dependencies** – defect uncovered because of language-specific details of implementation.

The skill levels are

1) **New/Trained**
2) **Within Product**
3) **Within Project**
4) **Cross Project**
5) **Very Experienced**

For checking Design conformance, the author suggests using people with knowledge **Within Product** and **Within Project**. Operational Semantics require **New/Trained** or **Within Project**, while side effects and Concurrency require more expert reviewers. Backward Compatibility requires inspectors with knowledge only **Within Project** knowledge, but Lateral Compatibility requires **Cross Product** knowledge. The Rare Situation requires the **Very Experienced** reviewers. Finally, the Language Dependencies require only **Within Product** knowledge. These are not exactly the same kind of cause and effect that we are looking for between the variables and the defect classifications, but they are a step in the right direction and can provide some valuable insight into these important relationships.

In the study by [Doolan92] that was discussed in Section 1.3.2, they analyzed the data from 11 requirements documents which totaled about 500 pages. Inspectors were chosen from various groups within the organization. These inspectors were chosen because of their perceived expertise in their area. The researchers at SSSG believed that having this varied expertise would better prepare them for finding the specific defects that were related to their area of expertise, and might have been missed by an inspector not as familiar with that area.

In the [Dow94] study also discussed in Section 1.3.2, they discovered an interesting interaction between product knowledge and the types of defects found during inspection. In their experiment, they had two groups, a control group and an experimental group. The subjects in the control group and the subjects in the experimental group were drawn from graduate level computer science students who were also practicing professionals. Each group also received comparable training in inspections. The difference is that the control group was taken from a multi-semester class, of which one part was to develop the software under inspection. The experimental group had no previous knowledge of the software under study. They found that in cases where the defects could be found by consultation with an "oracle", that having product knowledge was not helpful to the inspector. They defined an oracle to be a document that the artifact under inspection could be compared against, such as a coding standard, a design document, a maintenance document, comments in the code, or other parts of the code. In fact, lack of product knowledge allowed the experimental group to find defects that were missed by the control group. But, in cases where the "oracle"

is incomplete or incorrect, then the control group found defects that were not found by the experimental group. For example, in the design inspection, the oracle that could be used as a basis for comparison is the requirements document.

In the [Fagan86] paper described in Section 1.3.2, while he recommends that the inspectors be domain and language experts, he says that the moderator should be someone drawn from a different, but related project. The reason for this is to help maintain objectivity.

In a paper on Phased Inspections, described in more detail in Section 1.3.1.1, [Knight93] observed that as inspectors become more familiar with the particular checklist being used for the inspection, their inspection rate increases. They also found that as the inspector's knowledge of the C programming language increased so did their inspection rate. Both of these findings show some linkage between background and experience variables and inspection success. The familiarity with the checklist is similar to our variable of **Experience in this reading process**. The finding here was that this experience helped the inspectors. The knowledge of the C programming language can be considered part of the variable **Software Development Experience** since this was a code inspection. The finding here was that increased Software Development Experience also helped the inspectors.

[Porter97] is an investigation of the necessity or benefits gained from the inspection meeting as compared to just collecting individual inspection results. The paper discusses two different experiments, one performed at the University of Maryland and the other at the University of Hawaii. While background experience, or **Inspection Experience and Software Development Experience**, and knowledge, or **Domain Knowledge**, were not things being investigated in these experiments, they report an interesting result. Two experiments were run in different contexts and the results were compared. In the inspections performed in the first experiment, the subjects were inspecting the source code from an implementation of a program they had just completed. Because of this these subjects were both very familiar with the problem domains (Employee databases or two-pass assemblers) as well as the programming language (C or C++). On the other hand, in the second experiment, the reviews were done on formal requirements for an automobile cruise control and water level management. These subjects were neither as familiar with the specific problem being solved or the problem domain in general like the subjects in the previous experiment were. Even though we do not have conclusive evidence here because one set inspected requirements and the other inspected code, we can still see the continuation of the trend, the subjects that were more familiar had a greater defect detection rate. Thus, domain knowledge may have a positive impact on the performance of an inspector.

1.3.3.2 Our Proposed Approach

Our approach will be to look at the defects that appear in each class. We will also look at the inspectors that found those defects. Then, for each class of defects, we can look at which inspectors were the most likely to find that type of defect. We can then look at the values of the background and experience variables of those inspectors that found the defects. We will perform a statistical analysis to determine if there is any cause effect relationships present. If we can find a relationship between a background or experience variable and a defect class, then this information could be very useful in the planning of an inspection process.

For example, if a software development organization has been collecting defect data about its existing software products, it can determine the types of defects that are most commonly missed during inspection and test. These are the worst kind of defects, because they are the ones that the customers see. While they might not be the most severe in the eyes of the development team, the are the ones that cause customers to lose confidence in the product or the organization. If the organization determines that frequently it releases products that are missing pieces of functionality, and if the research into experience variables has determined that an inspector who has a high amount of expertise in the application domain is good at finding defects of omitted functionality, then in the future, this development organization will want to plan its inspections in such a was as to include more people with the necessary experience.

1.3.3.3 Methodology

After developing the defect classification schemes as well as the background and experience variable list, we will then proceed to investigate if there is any relationship between the two. We will take each background and experience variable one at a time. For the variable currently under investigation, we will group the inspectors into different groups based on their values for that variable. We will then look at the defects that are found by members of each group to see how they differ. First we will look to see if there is an effect seen in the overall defect detection performance. Then we will look at the performance of detecting defects of each class. If certain classes of defects appear to be more likely to be found by members of one group over the members of another, then we will have found a piece of information that will be useful to inspection planners when creating their inspection team. We will repeat this procedure for each of the variables.

The above procedure will be done for each experimental data set that we have. It will not be possible to group all of the data together because of the variations among experiments. After we have performed the evaluations described above for each individual data set, then we can begin to look at how the results from the different data sets compare with one another. And, if the environmental setting for two or more experiments is similar, then it may be possible to group that data together and rerun the analysis to see if the results support the conclusions that have been drawn.

1.3.4 Summary

After completing each of these three pieces we will be able to put them back together to provide a helpful approach to the problem described earlier. The list of background and experience variables will provide a process manager with the characteristics that are important when considering members of an inspection team. The findings about the relationships between the background and experience variables and the defect classes will provide the process manager with more guidance on the inspectors that should be chosen for his team. We will be able to provide a set of guidelines that help in this selection process. These guidelines will suggest the characteristics of an inspector that will help them in general in the inspection. In addition, if the organization is prone to making a certain type of defect, our guidelines will also suggest the characteristics of the inspector that may be most likely to find that type of defect during an inspection.

1.4 Other Related Research

One method of theory building that is popular in the Social Sciences is called *Grounded Theory* [Glaser67]. This approach to theory building is based largely in the data that has been collected in observation of the phenomenon under study. Instead of forming theories based on assumptions that the researcher has *a priori*, the theory is formed systematically from the data. They continue by listing the purposes of theory. Even though Glaser's work was in the field of Sociology, because we are studying how people use inspection techniques, these same ideas can be helpful. The stated purposes are :

1) To enable prediction and explanation of behavior.
2) To be useful in the theoretical advancement of Sociology.
3) To be useable in practical applications, meaning that practitioner should gain some control of situation because of theory.
4) To provide a perspective on the data, or a stance to take towards the behaviors exhibited.
5) To guide research on a particular behavior.

Based on this idea, [Gilgun92] gave a concrete way to implement this method of forming theories from the data. They give the same idea of *Grounded Theory* another name. They call it *Constant Comparison*. They define this as theory that is developed by "interweaving observations of phenomena of interest, abstractions from these observations, and previous research and theory." The main idea behind this, is that as the data is analyzed, the theories are continually modified and updated to take into account each piece of data. They provide a series of 21 specific steps to follow when doing this type of research. We will summarize and extract the parts of this that are important for our work here. First, once you have decided on a topic of investigation, a literature search should be performed. After this, the researcher should enter into the study with an open mind, willing to observe things that may go against his or her preconceived notions. The first case should be observed and described. Based on this information, one can begin to form theories and hypotheses. After doing this, the literature should again be searched to see if there is any other information on the specific findings from the first case that was not found in the previous literature search. The next step is to observe a second case. While doing this, the researcher will either confirm theories and hypotheses that were discovered in the first case, or will have to modify the theories and hypothesis from the first case so that they apply to both cases. This process of reviewing new cases and modifying the hypotheses and theories to take them into account should continue until some point of confidence. This could be either when one runs out of cases, or when each new case is causing very little or no change to the current theories and hypotheses. At this point, the theories and hypotheses are fairly solid.

Finally, [Day93] provides us with some rules for creating categories. This area of research applies to us in the creation of our defect classes. Each one of those is, in some sense, a category. So, the following rules are helpful to judge our defect classes.

1) Become thoroughly familiar with the data.
2) Always be sensitive to the context of the data.
3) Be flexible – extend, modify and discard categories.
4) Consider connections and avoid needless overlap.
5) Record the criteria on which category decisions are based.
6) Consider alternative ways of categorizing and interpreting the data.

We will use these pointers as we evaluate and evolve both our defect classification schemes as well as our list of experience variables.

While this method of theory building has come from social science literature, it is not novel to apply it to the field of software engineering. In a study on reading techniques for Object-Oriented framework learning [Shull00] this type of approach is used to construct theories about how subjects use frameworks. [Seaman97] also discusses the use of this approach when in the understanding of communication among members of a software development organization.

1.5 Evaluation Plan

Evaluation of the proposed approach in this chapter must be accomplished in 3 parts. First we must ensure that we have a useable set of defect classes. This requires the examination of the proposed defect classification schemes that will be described in Section 2.2.2 to ensure that the classification schemes are both complete and useful. More specifically the general classes should be complete such that for any new defect that is uncovered, it should be able to easily fit into one class. The general classification should also be useful such that a fair number of defects fall into each class. While the specific classes may not be complete, because a new defect that occurs may be of a different specific type than the ones we have previously found, they must be useful for improving the inspection process. If we find either of these cases to be false, then the defect classification scheme will be updated and revalidated.

Secondly, we must have a useable set of background and experience variables. To do this we will have to verify that the set of experience variables captures relevant background experience. Moreover, we must make sure that each background or experience variable that is in the set is one that actually has some effect on the outcome of the inspection. This means that we may add variables to the list if we uncover variation that cannot be explained by variables already in the list or remove variables from the list if they are found to be extraneous.

Finally, we must validate any cause and effect that we find between the background and experience variables and the defect classes. For our conclusions to have any validity, we must verify that for a given value of an experience variable, there is a correlation to the defects found from a given class, or the defects not found from a given class.

1.5.1 Evaluation of Defect Classifications

When we talk of defect classification schemes, many questions arise. One of the first questions that must be addressed is that of correctness. A straightforward approach would be to determine if the defect classification scheme is "correct". This is not such an easy task. Striving for "correctness" is not an adequate approach because the definition is subjective and can vary from situation to situation. Instead of striving for correctness, we are seeking defect classifications that are useful. This task is more feasible than the first one. In order to have such a classification scheme, it must meet a couple of different high level goals. The classification scheme must be useful both to the researcher and inspection manager as well as to the individuals who are developing software. These two goals lead us to ask a series of questions to support the validation.

EQ1 Are the two levels of the classification scheme at the right level of granularity?

This question really has two parts. We must verify both the general classification as well as the more specific classification. For the general classification to be useful, each of the classes must be general enough to contain a large number of defects. Those classes should also be orthogonal because if different inspectors put the same defect into different classes, then the classification does not provide much assistance for testing relationships between variables and defect classes. The classes should be described well enough that even without the specific classes inside of each general class, it is clear where to put a defect. The specific classifications should describe the defects in such a way that the inspection process can be improved if a specific class of defects is not being found at all. For instance, if there are instances where the data type of a particular attribute is different in different parts of the design, but this defect is not caught during the inspections, then a step should be added to the inspection process to check for this defect.

EQ2 Do all true defects easily fit into one of the general categories?

This addresses one of the issues from EQ1 more specifically. Namely, is the defect classification both clearly defined and complete. By clearly defined we mean that it is clear to the inspector which class each particular defect should be placed into. This involves both a well-defined set of classes, as well as providing a clear explanation during the training. By being complete, we mean that for each defect it is not only is clear which one class that it should go into, but that it makes sense to put that defect in that class. If a defect is found that does not easily fit into any of the general classes that have been defined, then it may be necessary to expand the defect classification scheme to include a new defect class.

EQ3 Do the specific categories reflect information that is useful to the process developers?

This addresses the other issue from EQ1 in more detail. Namely, when we have the specific categories for the defects, do the descriptions of those categories give us enough information to be able to improve the inspection process? If the descriptions are too vague, or too general then it will be difficult to add a step to an inspection procedure to address that issue.

1.5.2 Evaluation of Background and Experience Variables

The set of background and experience variables that has been proposed must also be evaluated to judge its usefulness. For the set to be judged useful, we must have a set of variables that is both complete and minimal. By completeness we mean that all of the potential background and experience variables that we have identified have been examined and any variation that we can identify in the performance of inspectors can be explained by some combination of these variables. By minimal we mean that only the variables that directly affect the performance during the inspection have been kept in the set. Any variable for which we cannot show some correlation must be removed from our set of variables.

EQ4 Are all of the variables really variables?

Here we will look at the values of the variables that correspond to each individual inspector. If for any of the variables in our list we find that there is no variation among the inspectors, then for our purposes this variable should be removed because it is not really a variable. This evaluation will be done based on the responses to the questions given for each variable in Section 1.4.2. If we do find a variable in our list that should be removed, we will not say that this variable has no effect, only that with the set of inspectors that we are studying it has no variation. A later experiment might be planned to test that variable specifically, although that is not part of this work.

EQ5 Is the list complete?

We want to know if we have described the variations among inspectors that we can identify by the variables in our list. This information is based on the data sets that we will have to study. There may be other background and experience variables that are not covered by our list, but we want to make sure that any variation we see in our data is explained by a variable in this list. If there is another dimension on which inspectors vary that has an impact on the inspection process, then we would also want to investigate that variable. Based on the qualitative data that we collect from the subjects of our studies, we should be able to get a good idea of what really occurred and be able to identify if there are other variables that we have not enumerated yet.

1.5.3 Evaluation of Interaction between Background and Experience Variables and Defect Classes

While the data from various experiments can be used together in creating the defect classifications as well as in developing the defect lists, special precautions will have to be taken when it comes to evaluating the interactions between variables and defect classes. Because the data that was collected from varied development situations, it will not be possible to naively group all of the data together to investigate these interactions. These evaluations will have to be performed on each data set separately. Then the results of those analyses can be used to either support or contradict one another. The only situation where grouping data from different experiments would be possible is if the environment and variables associated with the experiment are similar enough that it would not introduce potentially confounding factors into the data analysis. Based on these results we will be looking for heuristics that hold over multiple studies in varied environments. These heuristics could lead to further experimentation to expand the context in which they hold.

EQ28 Which of the variables contribute to finding the largest number of defects?

Once the variables that affect the inspection process have been determined (Section 1.3.2), we need to find the linkage between those variables and the success of the inspection. If we can determine that a particularly high or particularly low value of a certain variable is beneficial to an inspection, this information would be very useful to a process manager. The information would be useful in the planning of the inspections. It would provide guidance on which members of the development organization should be chosen for each particular inspection, based on the specific problem being solved as well as the background of each of the potential inspectors.

EQ29 Do some variables increase or decrease the likelihood of finding certain classes of defects?

Similar to the previous question, we not only want to determine which variables affect the overall success of the inspection, but we also want to know if particular variables are especially useful in finding each class of defects. Based on the defect classification scheme from Section 1.2.1, if we can find a correlation between the background and experience variables and those defect classes this information would be useful in the planning of inspections. If an organization wants to focus special attention on a specific class of defects, because they know that class generally makes it into their released products, or that class is very severe if not caught, then based on the correlations we find the best people can be chosen for the inspection team to find as many of those defects as possible.

2. Completed Work

We have created initial classification schemes for both Requirements defects and Object Oriented Design defects. We have also developed an initial list of background and experience variables. Each of these lists will provide us with a starting point for the proposed work. In addition to the literature reviewed, which was discussed in Chapter 1, we also performed an experiment. This experiment will be described in this section along with the initial lists. The initial lists are based on the background literature as well as the analysis of a single data set, and may need to be evolved as more data is analyzed.

2.1 The Experiment

Background

PBR for requirements and OORTs for Object Oriented design were both studied in this experiment. In this experiment we were interested in studying the techniques in isolation, rather than in the context of a full software lifecycle. The main reason for this was that we wanted to study the problem domains, and the background of inspectors, for which the techniques could be most useful.

To get the level of detail about the techniques that we wanted, we used an observational approach. Observation is not commonly used in Software Engineering. The observational approach is one that uses direct observation of a process executor while they are executing a process. In our case, we observed inspectors while they were inspecting the software artifacts. The observational approach provides the researcher with specific data about how the executor performed the process. The decision to use an observational approach was dictated by the desired results. We wanted to discover what improvements might be necessary at the level of individual steps. For example, whether subjects experience difficulties or misunderstandings while applying the technique (and how these problems may be corrected), whether each step of the technique contributes to achieving the overall goal, and whether the steps of the technique should be reordered to better correspond to subjects' own working styles. For a more complete description of observational studies see [Travassos99]. For more details about this experiment see [Shull00b].

Subjects

The 28 subjects were members of a graduate-level Software Engineering class at the University of Maryland during the Fall 1999 semester. The subjects were grouped into pairs to carry out the experiment. For each inspection, one person was the process executor (responsible for carrying out the inspection) and the other one was the process observer (responsible for recording observations about the execution of the process). Since each pair performed a requirements inspection and a design inspection, each person got to play both the role of executor and the role of observer. We believe that the background experience was only relevant for the process executor, since they were the one actually performing the inspection. Of the 14 subjects that performed the requirements inspection, 29% had previous industry experience with writing or reviewing requirements, 64% had classroom experience, and the other 7% had no experience. Of the 14 subjects that performed the OO inspection, 86% had previous industry experience with OO design and the other 14% had classroom experience. All students received training on both the requirements reading technique, the OO reading techniques and the observation process.

Materials

The materials under study during this experiment were PBR, described in Section 1.2.2, as well as OORTs, described in Section 1.2.3. They were applied to the artifacts of two systems. The first was a Loan Arranger (LA) system, which was responsible for organizing and selling loans held by a financial consolidation organization. The second was an automated Parking Garage Control System (PGCS), which was responsible for allowing drivers to enter and leave a parking garage and for keeping track of monthly parking tickets as well as the number of available spaces for general parking. The Loan Arranger requirements document consisted of 8 pages with 26 functional and 4 non-functional requirements. The LA design used in this study was relatively small, consisting of 7 classes in the high level design, 4 interaction diagrams and 3 state diagrams. The PGCS requirements document consisted of 17 pages with 21 functional and 9 non-functional requirements. The PGCS design was also relatively small, consisting of 6 classes in the high level design, 5 interaction diagrams and 2 state diagrams. The LA problem domain was selected due to its unfamiliarity to reviewers, while the PGCS domain was selected for familiarity.

Procedure

A quasi-experimental, factorial design was used. The 14 two person teams were broken up into 4 groups. During the requirements inspection, 6 teams inspected the PGCS requirements and 8 teams inspected the LA requirements. During the design inspection, of the 6 teams that inspected PGCS requirements, 3 inspected the PGCS design and 3 inspected the LA design. Likewise, of the 8 teams that inspected the LA requirements, 4 inspected the LA design and 4 inspected the PGCS design. Because half of the teams inspected requirements and designs from the same system, and half did not, we could look for any differences in the performance in the design inspection due to the reviewers' past familiarity with the system requirements or with the problem domain.

Before the study, subjects received training in the reading techniques to be applied and the observational methods. Training in observational methods was accomplished by presenting the roles of executor and observer and defining their specific responsibilities. One member of each team was assigned to be the executor and the other to be the observer. Each team was required to submit a report describing their experiences during the inspections. The data for this report was based largely on the observations during the execution of the inspections. The subjects were not instructed as to which specific pieces of data to include in their report. Rather, they had to come up with that on their own.

Data Collection

The data from this experiment was collected in two ways: analysis of artifacts, which includes the experimenters examining and analyzing the work products and defect lists submitted by the subjects, and observation. Analysis of artifacts was used for collecting some quantitative data, namely the time required for executing the techniques and the number and type of defects detected. However, observational techniques were the most important method used in this study. A rich array of qualitative data was collected through their use. As mentioned earlier, the teams produced an evaluation report, which included both a summary of the notes taken during observation as well as retrospective data determined after the execution of the process. The metrics collected from the observations were:

- Executor's opinion of effectiveness of technique
- Problems encountered with specific steps of procedure
- How closely executors followed the techniques
- Number and type of defects reported

The retrospective data (collected via open-ended questions) provided the following information:

- Time spent in review
- Usefulness of different perspectives
- Were the techniques practical, would they use some or all of them again
- The problems found using the techniques

As can be noticed, the retrospective data are better suited to global issues, rather than the critiquing of individual steps. Also, some of the metrics collected here were the same as in the previous feasibility study, allowing a comparison of results across the two versions of the techniques. This observational data provided us with information that will be useful in determining the cause-effect relationships between the experience variables and the defect classes.

2.2 Developing Classification Schemes

2.2.1 Introduction

There are many reasons for classifying defects. They range from a simple one of characterization all the way up to a more complex one of process improvement. A defect classification scheme that is meant only for characterization can be relatively simple with only a small number of classes. The main goal for this type of classification is to allow the development organization to see where they make their defects. And even to see at a limited scale how various changes to the process and/or personnel may affect that characterization. On the other hand, a defect classification scheme that is meant to help drive process improvement will have to be more complex, as the classification scheme will have to point the process manager in directions that can improve both the software development process and more specifically the inspection process.

2.2.2 Initial Work on Defect Classifications

The first step in our work was to begin to create defect classification schemes that we could use as a starting point, and then improve. Because of the goals for the defect classifications which were discussed in Section 1.3.1, we decided that it would be more beneficial to create separate classification schemes for requirements and for design defects (we don't examine code inspections in this work). The classification schemes discussed in the previous section provide some good ideas and different ways to think about the problems. We decided that in addition to that information some concrete data of our own would help us to flesh out the schemes for our purposes. The data that was used was drawn from the experiment described in Section 2.1.

2.2.3 Creation of Initial Defect Classifications

Based on the results of the above experiment, we created the initial classification schemes for the defects. Because the defects that are found by inspectors in one isolated project may not be representative of the entire population of defects, this experiment explored two different problems. The first was the Loan Arranger problem, which was in the financial domain, (where most subjects had low expertise), and it was a static problem. The second was the Parking Garage problem, which was in a much more familiar domain, but it had more of a dynamic aspect to it, meaning the state machines and sequence diagrams were more extensive. We felt that because of the differences in the two problems, we would have a good diversity of defects as our base.

There are two types of defects that made up the list from which our initial classification schemes were created. Based on their past experience with inspections and the common types of defects, the experimenters seeded many of the defects, especially in the requirement phase. While this does not provide as realistic a sampling as simply taking defects that occur on a real project, it was necessary in order to maintain control. The second type of defect that occurred mainly in the design phase, were those defects that were made by the experimenters, who were the document authors, in their regular process of creating

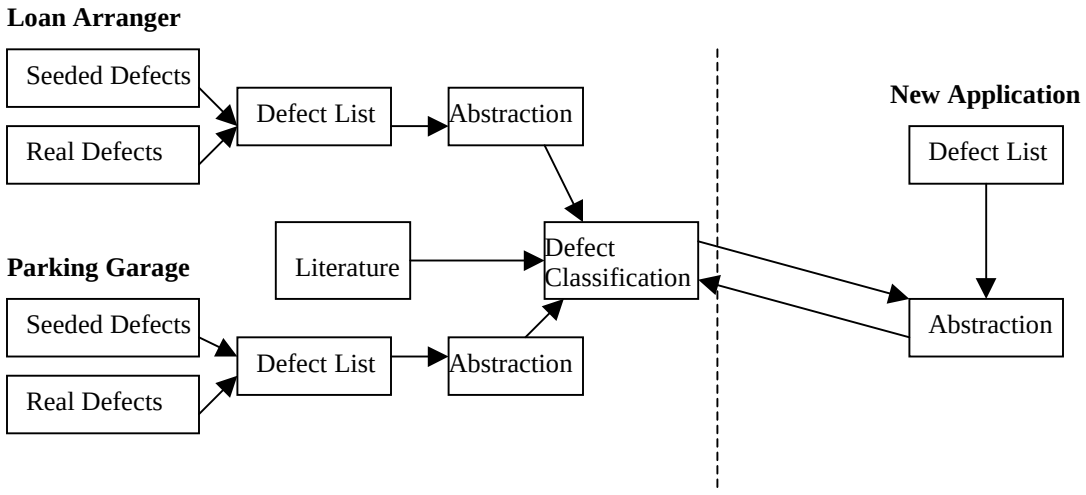

Figure 3 – Methodology for Defect Classifications

the documents. We think that these two sources of defects, when coupled with the information found in the literature, provided us with a good source for creating the initial defect classification schemes.

The defect classification schemes discussed in the next two subsections came from performing the tasks on the left hand side of the dashed line in Figure 3. We looked at each defect that was present in the two problems. Each defect was then abstracted so that it was no longer domain specific, so that we could compare defects across the two domains. After this we grouped defects that appeared to be similar. This grouping was aided by the defect classification schemes that had been described in the literature. The results of these tasks are the two defect classification schemes presented below in Sections 2.2.3.1 and 2.2.3.2.

2.2.3.1 Requirements Defect Classification

I. Interface and Access – These defects deal with the way that users or other systems gain access to this system. They deal both with the mechanisms of access (the interface) as well as the restrictions placed upon the access.
   A. Interface not described
   B. Wrong Actor used
   C. Access to system not adequately described including access requests at inappropriate times.

II. Data – These defects deal with the actual data items that are to be maintained by the system. This includes inconsistent data types, validation of new data, and access restrictions to for the data.
   A. Input/Output format not given
   B. Data either checked against wrong constraint or not checked at all
   C. Not clear what data value is being checked
   D. Data access is not properly restricted for a process that needs exclusive access
   E. Information deleted from the system will cause loss of necessary information for other parts of the system.
   F. Not defined who can add/edit/delete a data item.
   G. When one data value is updated, all associated data items do not contain correct values
   H. Data type is described but never used
   I. Domain specific type conversion not handled
   J. All possible data types not handled.

III. States – These defects deal with system, object or data states. While it is at the requirements level, the concepts of states is still present.
   A. Passive state change not described
   B. The truth of a condition puts an object in a particular state, but the set of conditions are not orthogonal.
   C. The system is supposed to act when something happens, but all conditions, including errors, are not handled.
   D. Error condition handled without user notification.

IV. Terminology – These defects occur when terminology is incorrectly or inconsistently used in such a was as to confuse a later user of the document.
   A. Term not defined
   B. Wrong term used
   C. Terminology not consistent

V. Document Formatting issues – These defects deal with problems in the placement of requirements within the document.
   A. Requirement in the wrong section
   B. Reference to the wrong requirement

VI. Other Inconsistencies – These defects deal with situations where the requirements document says contradictory things in two different parts of the document.

2.2.3.2 Design Defects Classification

I. Messages – These defects deal with problems involving messages that appear in the design.
   A. Messages out of order on sequence diagram
   B. Message name on sequence diagram not the same as on Class Diagram
   C. Extraneous message on a sequence diagram
   D. Message on a sequence diagram has different parameters compared to the Class Description.
   E. Message on a sequence diagram is not on Class Description

II. States – These defects occur when the states of the objects are misunderstood in some way. This includes the addition or omission of an entire state, or incorrect transitions.
   A. Misunderstanding of what the start state is
   B. Misunderstanding of the navigability between states (transition exists between states where it should not)
   C. Misunderstanding of condition for a transition between states or a self-transition
   D. State diagram depicts a state transition, but class has no attribute or combination of attributes that indicates this.
   E. Method for a state change was left off of Class Description but included everywhere else in the design
   F. Extraneous transition or self-transition in the design.
   G. Method for a state change (passive or active) on state diagram was omitted from the rest of the design
   H. Description of condition for transition does not match on State Diagram and Class Description.

III. Attributes – These defects deal with problems in the way that attributes have been described in the design.
   A. Extraneous attribute added to the design
   B. No way to adjust the value of an attribute
   C. Attribute on Class Diagram but not on Class Description
   D. Attribute has different types in the class description and the class diagram

IV. Actors – These defects deal with the misuse of the actors within the design.
   A. Actor is represented as a class on sequence diagram
   B. Actor is of the wrong type

V. Constraints – These defects occur when constraints are omitted or incorrectly used.
   A. Constraint on attribute left off of Class Description
   B. Constraint added to Class Description that is not true
   C. Constraint on System Updates (performance) omitted from the design

VI. Relationships – These defects occur when classes are incorrectly related within the design.
   A. Wrong type of relationship between classes on the Class Diagram
   B. Two entities related on the Class Diagram and they should not be.
   C. Cardinalities for classes are reversed on Class Diagram for two related classes.
   D. Classes that exchange information on Sequence Diagram are not connected on Class Diagram

VII. Class Hierarchy – These defects describe problems within the class inheritance structure.
   A. No behavior in design to change from one sub-class or state to another

VIII. Other – Design defects that do not fit in other categories, but are not major enough to create a new category.
   A. Sequence diagram has different name than corresponding Use Case

2.3 Identifying initial list of Experience Variables

2.3.1 Introduction

As was stated earlier in this paper, there are variables besides the process being used or the document being inspected that can have an effect on the outcome of the inspection process. The assumption has generally been that the most effective way to predict the results of an inspection process are to examine the process being used or the documents being inspected. But, this is not always the case. In fact, in some cases the presence or absence of certain inspectors can have a greater effect on the outcome of the inspection than the inspection process does [Porter98], [Siy96].

2.3.3 Initial Work on Experience Variables

There are many potential factors that can influence the outcome of an inspection. In order to come up with our initial list, we explored two sources of information. First, we examined the different background and experience variables implicitly or explicitly presented in the literature. Secondly, we used our own experiences with inspections and what had been factors in those situations. From these, we were able to generate an initial list of background and experience variables. Based on the data from the experiment described in Section 2.1, we fine-tuned that list. We wanted to create a list of variables that was both complete enough to describe the variation in inspectors, as well as not having any variables that were extraneous. This list provides us with a starting point. Upon the study of more inspection data, discussed in Section 3, this list may be expanded or contracted so that every variable is useful and interesting. For example, if all of the inspectors have the same value for a given variable, or it does not appear that variable will affect the outcome of the inspection, then that variable would be removed from the list. This describes the tasks that are represented on the left-hand side of the dashed line in Figure 4.

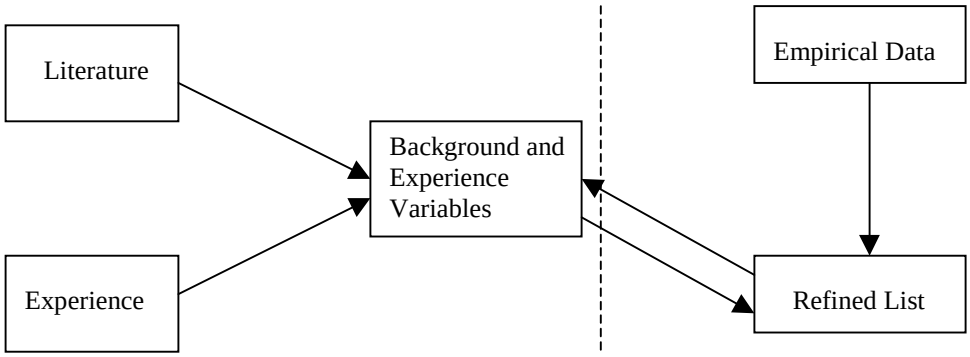

Figure 4 – Methodology for Background and Experience Variables

Here is the initial list of variables:

- **Environmental Variables** – Constraints placed on the inspection or reading process by the development organization, classroom setting, or other setting within which the inspection is taking place.

- **Training** – This deals with how the training for the process or technique was performed. It includes things such as who performed the training, how long it lasted, and the material that was used.

- **Domain Knowledge** – This deals with the amount of knowledge about the domain, such as banking or satellite control, which contains the problem being solved by the software being developed.

- **Software Development Experience** – This contains both experience in whatever development phase the inspection is occurring, e.g. Requirements, Design, Code, etc., as well as with specific technologies being used in the development process, such as SCR or Object Oriented Design.

- **Experience using another reading process** – This deals with the experience using another similar reading technique, but not the one that will be used in the current inspection. For example, if an inspector is familiar with a reading technique for inspecting a requirements document, then we would like to know how that affects them when using a new reading technique for inspecting an Object Oriented design.

- **Experience in this reading process** – This deals with the inspectors experience with the specific reading process that is going to be used in the current inspection.

- **Natural Language** – This deals with how familiar an inspector is with the language the document is written is, as well as the language the procedures or instructions are written in.

- **Process Conformance** – This deals with how closely the inspector follows the process that they have been given to aid in the inspection.

Once this list has been properly evaluated, it will be improved to contain only the variables that really have an effect on the reading process. This may mean that some of the above items are removed, some new ones are added, or some are combined.

2.4 Evaluating Interaction between Experience Variables and Defect Classes

2.4.1 Introduction

The previous section discussed what the different variables are that might affect the outcome of an inspection, and Section 2.2 discussed different defect classification schemes. While these two pieces of information are interesting in and of themselves, they become really useful when combined together. It would be a really helpful piece of information for a software process planner to know how the value of one of the variables in Section 2.3 affected the classes of defects, from Section 2.2 that an inspector tended to find. This would allow for better planning of inspections based on the history of the organization or the specific requirements of the present project.

2.4.2 Initial Work on relationships between Experience Variables and Defect Classes

We have not yet done a lot of work in this area. Based on the data from the experiment described in Section 2.1.3.1 we have some initial results in this area that can help to guide the future work. We have results for both requirements inspections as well as for Object Oriented design inspections. Those initial results will be described below. Neither the requirements results nor the Object Oriented results are at the level of individual defect classes, they are both at the level of total number of defects found.

2.4.2.1 Requirements

The main experience variables that were under study in this experiment were **Domain Experience** and **Software Development Experience**. When we examined **Domain Experience**, we compared the inspectors that inspected the Loan Arranger requirements, low domain experience, to the inspectors that inspected the Parking Garage requirements, high domain experience. We found, somewhat surprisingly, that having domain knowledge did not significantly help in the requirements inspection. If this relationship is confirmed, it would mean that it would not be necessary to always use domain experts as inspectors during a requirements inspection. We did find, as expected, a significant effect caused by the **Software Development Experience**. As the inspectors were more experienced in software development, they found more defects in the requirements they inspected. Also, part of the software development experience, was the fact that inspectors were using different perspectives (see Section 1.2.2 for an explanation of perspectives). One set of inspectors had to inspect from the tester point of view and the other from the user point of view. We also found that as the inspectors were more experience in whichever perspective they inspected from, the performed better in the inspection.

2.4.2.2 Design

In the design inspection, we also looked at **Domain Experience**, but we looked at it from two views. The first was the same as was used in the requirements inspection with Loan Arranger being low domain experience and Parking Garage being high domain experience. The second view was that of experience with the requirements. We compared the performance of teams that had previously inspected the requirements for the design they were inspecting to those teams that had not inspected the requirements. Here again, we had a surprising result, the readers who had low domain knowledge did better in the design inspections than the readers with high domain knowledge. On the other hand, having inspected the requirements previously did not help or hurt when performing an inspection of the design.

2.4.3 Conclusion

While neither of these results provides us with concrete evidence of a cause-effect relationship, they provide us with some evidence that we should continue to investigate. One reason is to investigate the cause-effect relationships at a deeper level, not only do we want to know if the variable causes an inspector to find more or less defects overall, but we want to know how those variables affect the individual classes of defects we have defined. Secondly, we have some counterintuitive results, that if they can be verified could provide some very interesting advice for someone planning an inspection.

3 Proposed Work

3.1 The Data and Experiments

The defect classification schemes as well as the experience variables and their linkage will be studied through the use of data from a set of experiments. Each of the experiments to be used will be described below. Some of the experiments will be used to improve and fine-tune the lists, while the others will be used to verify that the lists are correct. In each of the sections following, it was described which experiment would be used for which purpose.

3.1.1 PBR Experiment at NASA

Background

The goals of this experiment were to determine if the use of PBR was beneficial to NASA professionals over their usual unspecified technique. The experimenters wanted to understand if a structured and focused set of techniques like PBR could help the inspectors' performance. They wanted to explore this performance both on a requirements document that was not NASA specific as well as one that was NASA specific. This was done in order to determine if the techniques were helpful to inspectors that were highly experienced in the domain. The data from this experiment will provide us with insights into both performance by inspectors using a reading technique, as well as using and ad hoc procedure.

Subjects

The subjects were NASA professionals.

Materials

There were two requirements documents from non-NASA specific domain. A requirements specification for a Parking Garage (PG) system was used, as well as a requirements specification for an Automated Teller Machine (ATM). In addition there were two NASA specific documents that were reviewed, NASA A and NASA B. The subjects used the Perspective Based Reading techniques described in Section 1.2.2.

Procedures

The subjects performed the inspections individually. Teams were simulated by combining the defects found by individual reviewers. In the PBR portion the individual reviewers that were combined represented all of the perspectives. After training, the readers performed an ad hoc inspection of either the NASA A or the NASA B document (half of the groups inspected each one). Then the subjects performed an ad hoc review of either the PG or the ATM document (half of the groups inspected each one. The subjects were then taught PBR. They then reviewed the other non-NASA specific document (PG or ATM) than they had reviewed in the first part. Finally, they reviewed the other NASA document (NASA A or NASA B) than they had reviewed earlier.

Data Collection

Data was collected from questionnaires before the study to understand the subjects background. The defect lists produced by each subject when inspecting each document provided the qualitative data about the number of defects found.

3.1.2 CMSC435 Fall 1998

Background

This is a study that has already been completed, but the data has yet to be fully analyzed. It was conducted as part of a senior-level Software Engineering course at the University of Maryland. The class consisted of mostly upper-level undergraduates as well as some graduate students. The main goal of this study was to test the feasibility of newly developed Object Oriented Design reading techniques as well as to continue the investigation of and improvement of the Perspective Based Reading [Basili96] techniques for inspecting a requirements document. For complete details on this experiment see [Shull99], [Shull00b].

Subjects

The subjects in this experiment came from an undergraduate software engineering course at UMCP during the Fall 1998 semester. The 44 students in the class had a mix of previous software experience. In terms of writing or reviewing requirements, 23% had prior industrial experience, 52% had experience in a class, and the other 25% had no experience. In terms of design, 32% had some prior industial experience in software design from requirements and/or use cases, 59% had classroom experience, and 9% had no prior experience. However, all students were trained in OO development, UML and OO software design activities as a part of the course. The subjects were randomly assigned into 15 teams (14 teams with 3 students each and 1 team with 2 students) for the experiment.

Materials
The materials under study during this experiment consisted of an initial version of the OORTs, as well as a more mature set of PBR. The PBR techniques were applied to a requirements document consisting of 9 pages, with 5.5 pages of background and definitions and 3.5 pages of requirements. There were 5 high level functional requirements, 13 data constraints, 20 data operations, 7 interface requirements, and 4 nonfunctional requirements. The OORTs were applied to the software artifacts of a "Loan Arranger" system discussed in Section 2.1. It was a small system, but contained some design complexity due to non-functional performance requirements. The version of the design inspected in this experiment consisted of 11 classes, 3 state diagrams and 5 sequence diagrams.

Procedures
The first thing that the subjects did was to receive the Loan Arranger requirements and perform an inspection using PBR. In addition to providing us with data about a requirements inspection, it allowed the subjects to gain a better understanding of the system. After all teams had completed the requirements inspection, the experimenters corrected the defects in the requirements document. This corrected requirements document as well as a set of use cases were distributed to the subjects. From these documents they were asked to create an OO design for the system.

After all of the designs had been collected, the best one was selected and distributed to each team except for the team that created that design (they were given another design to ensure that each team had a design created by another team). The teams were then instructed to perform an inspection on the design they had been given.

Each subject applied one of the perspectives for PBR during the requirements inspection phase, and a subset of the OORTs during the design phase. The techniques were assigned such that when taken as a whole, every technique had been performed by at least one member of the team. In both the requirements inspection as well as the design inspection, each team member did an inspection individually. Then the team met and compiled their defect lists into one final one for the team.

Data Collection
The data for this experiment was collected in three ways: analysis of artifacts by the experimenters, questionnaires, and interviews. The questionnaires used to collect the qualitative data were distributed both before and after the experiment. The artifacts that were examined included defect lists and all of the system artifacts. Each team was interviewed individually after completion of the experiment.

3.1.3 CMSC435 Spring 2000

Background
This experiment is also one that has already been run, but the data has not been analyzed. It was conducted as part of a senior level software engineering course at The University of Maryland. The main goal of this study was to evaluate the effectiveness of using inspections with reading techniques as performed within the context of a software process. Both Perspective Based Reading (PBR) for requirements and OORTs were evaluated.

Subjects
The subjects in this experiment came from a senior level undergraduate software engineering course at UMCP during the Spring 2000 semester. The 42 students had a mix of previous experience. In terms of

writing or reviewing requirements, 24% had industrial experience, 40% had experience in the classroom, and the remaining 36% had no experience. In terms of design creation or reviewing, 14% had some industrial experience, 45% had classroom experience , and the remaining 40% had no prior experience. All the students were trained in requirements elicitation, requirements review, OO development, OO software design activities, UML, and OO design inspections as part of the course. The subjects were grouped into high, medium, and low expertise categories, and one person from each group was randomly assigned to each of the 14 3-person teams.

Materials
The materials under study in this experiment were the PBR techniques and the OORT techniques. They were applied in the evolution of the PGCS system, described Section 2.1.3.1. The students were given a list of enhancements and had to create a new requirements documents, and a new design. The enhancements included functionality that allowed customers to reserve tickets and pay bills over the Internet.

Procedure
Before this experiment was run, a version of the PGCS system was designed and implemented based on the requirements that had been used in previous experiments. This consisted of a requirements document, and Object Oriented design, and executable modules, all of which the subjects had access to. The subjects used a waterfall development process, to create an enhanced version of this system. First of all, they were given the set of requirements that had been implemented, as well as a list of enhancements that were required for the new system. The subjects then created a new requirements document combining the existing requirements with the newly added ones. They then performed an inspection on this document.

After correcting the defects found during the requirements inspection, each team created a design for the system. The teams first performed an initial inspection to ensure that their designs were consistent, and corrected any defects that were uncovered. After this, the teams traded designs. Each team then performed a more detailed inspection of the design they had been given. The list of defects found by the reviewers was then returned to the authors of the design for correction.

In the overall scope of the software development process there was no control group here. This occurred for two reasons: first, the design inspection was one small part of a larger experiment, and the overall experimental design did not allow for a control group. Secondly, in a classroom environment, it was not possible to provide instruction on a topic to only a portion of the class.

Data Collection
The data for this experiment was collected in two ways: questionnaires and analysis of created artifacts. The questionnaires were used throughout the development cycle to collect both qualitative and quantitative data. The quantitative data collected include both background information, used to classify the subjects as having high, medium, or low expertise, and the amount of time taken to use the techniques, used to evaluate feasibility of use. The qualitative data collected by the questionnaires concerned:

- Opinions of the helpfulness of the techniques.
- Problems encountered using the techniques, or extra knowledge that was needed to use the techniques.
- Opinions of effectiveness of training.

Analysis of the defect lists provided quantitative data about the number and types of defects found by the teams. The data was useful in determining if the output of the reading process uncovered defects and was useful for continuing the development process.

3.1.4 USC Experiment
This experiment will allow us to further explore the background and experience variables. Specifically, the subjects in this experiment will not only be reviewing a requirements document, they will be first eliciting those requirements and writing the document. This provides us with the ability to see if inspectors who were involved in the elicitation and creation of the requirements perform inspections any differently than those that did not. Secondly, the projects that will be created in this experiment will be from the library systems domain. This will allow us to see if the defects made in that domain fit into our classification scheme, or if we need to expand it because different types of defects are made that we have not

encountered before. In this experiment we will be able to collect data both on subjects who use techniques for the inspection as well as subject who perform an ad hoc inspction.

Method
This is an experiment that will be conducted in the context of a graduate software engineering course at the University of Southern California. The course is part of a two semester sequence in which a software product is built, beginning with requirements elicitation and ending with a delivered product. One main benefit of this experiment is that the product being built has a real customer and contains real requirements. Therefore the defects that are made will be real defects. In the past we have mainly been able to work with artifacts created by us, with defects seeded by us.

Subjects
The subjects will be members of a two semester software engineering course at the University of Southern California. The students will be grouped into teams consisting of approximately 5 members. There will be approximately 20 teams in the class. Each team will be working on a different project.

Materials
The materials under study in this experiment will consist mainly of the PBR techniques for requirements inspections. These techniques will be applied in the inspection of the requirements document created by each team for their individual projects.

Procedure
During the course of the semester, the subjects are trained in the *Fagan Inspection* method. In addition to that, the subjects will receive additional training in PBR. The students will be in teams of 5 members each. Each team will have a real customer, probably someone in the library school, with a real product, to be used by the library school after completion. Each team will be responsible for meeting with their customer and eliciting the requirements for their project. After the requirements document has been created, the teams will be responsible for performing a *Fagan Inspection* on that requirements document. The 5 members of the team will play the roles in the *Fagan Inspection* that were described earlier. The PBR perspective that makes the most sense for each role will be used. The Author and the Reader will use the Customer perspective to gain an overall understanding of the requirements document. There will be two inspectors playing the Tester role and they will use the Tester perspective. Finally, the Moderator will use ad hoc reading to prepare for the meeting in whatever way he or she thinks is most appropriate.

Data Collection
Data will be collected in three ways: questionnaires, interviews and analysis of created artifacts. The questionnaires will provide us with information about the subject's background knowledge and experience that will be helpful in exploring the cause-effect relationship among experience variables and performance in an inspection. The analysis of artifacts will mainly deal with the defect lists that the subjects create during the requirements inspection, as well as the pre-inspection and corrected versions of the requirements document. Interviews with subjects after the experiment will provide us with qualitative data about the inspection process. Specific background variables can be explored as well as the subjects opinion of the inspection process. This type of information will be valuable in understanding the relationships between the background and experience variables and the defect classes.

3.1.5 Other Experiments
In addition to the experiments described above, we will also examine some other data sets as well as some other experiments that have either been planned or are in the process of being planned. The other data will come from replications of experiments described above that occurred both within a university setting as well as within places like NASA. The other experiments will provide us data for the validation of our new defect classifications and our cause-effect relationship between those classes and the experience variables. Each experiment will provide us with a different perspective on the defect data that has been collected in the past.

In the future studies there is potential for the collection of industrial data which would add greatly to the value of the data we already have. If data from industry can confirm what we have found based on data

from more controlled settings, then we can be more certain of our conclusions. We have potential industrial contacts both in Brazil through Dr. Guilherme Travassos, as well as with the Fraunhofer Center-Maryland. These settings would not be places to perform controlled experiments. Rather they would be sites where the inspection techniques that we have can be inserted into their process. While doing this, data will be collected and analyzed in the same way that we have described before through questionnaires, interviews and analysis of artifacts. In these experiments the interviews will continue to provide us with data about the background variables as well as helping us gain an understanding of their relationship to the defects found during the inspection process.

3.1.6 Other Data

In addition to the data sets specifically described above, we will make use of any other related data sets that we are able to gain access to. These include a replication of a Perspective Based Reading experiment that was done in Bari, Italy. Another replication of that experiment was done at the University of Maryland, Baltimore County.

3.2 Validation and Improvement of Defect Classifications

One of the major parts of this work is to improve the defect classifications we have so that they are useful for their stated purposes. This evaluation will be done using the data provided by the experiments described above. What we want to do is to ensure that the two levels of our defect classification fulfill their stated goals. The more general level should have categories that contain enough defects across many different domains that we can use them to evaluate the impact of the experience variables. If any of the stated categories has too few defects in it, then the impact of the variables on it will not be very interesting. Likewise, we want to make sure that in the more specific defect classification, that enough information is provided about the defect that a step could be added to a reading technique if that defect is never or rarely found.

To do this we will have to examine the defects that are reported in each of the experiments described in Section 3.2. We will need to examine each defect that has been reported to see if it fits into the classification scheme that we have described. If a given defect does not fit into any of the general classes of defects, then we may have to expand our classification scheme to include another general class. On the other hand, if we can find a general class to place the defect into, but it doesn't match any of the specific descriptions currently there, then we can add a specific description. This would give us an idea of something that may need to be added or modified in the inspection techniques. This is represented by the tasks on the right-hand side of the dashed line in Figure 3.

We will use the grounded theory approach and the steps described in Section 3.1. We have started by forming a defect classification scheme that is complete and useful for the list of defects that we started with. As we evaluate the data from each subsequent experiment or study, we will extend, modify and delete classes as necessary. After taking into account the defect lists from multiple separate studies, we should have good defect classification schemes. In doing this we will also keep in mind the guidelines for creating categories provided by [Day93].

EQ1 Are the two levels of the classification scheme at the right level of granularity?

EQ2 and EQ3, which are discussed below, deal with part of this question. One important thing that we much measure here is :

- Number of defects that occur in each major class. Done by counting them across all of the defect lists from all of the artifacts.

If we discover that there is a general category that has only a few defects relative to the other categories, then we must either combine it with one of the larger categories or eliminate it in some other way. The goal here is to end up with general categories that have enough defects in them that we can perform a statistical analysis. If the categories are too small, then we will not be able to discover anything interesting through our statistical analysis. We will run a statistical test to determine if any of the classes is an outlier with respect to the rest of them. If we find an outlier, then we must combine or split it so that we have relatively uniform data.

EQ2 Do all true defects easily fit into one of the general categories?

This is part of the question that is addressed in EQ1. Here we really want to determine how orthogonal the defect classifications are. If inspectors cannot easily classify a defect into its proper class, then our success in finding a correlation between a background or experience variable and any particular defect class will be greatly diminished.

- The way that we will measure this is to look at each defect that was found by multiple teams. If different teams describe a defect in such a way that it falls into different defect classes then our general classes are not sufficient. The classes will either need to be modified, or more clearly described so that each defect clearly falls into one and only one defect class.

Because the experiments have already been performed using a slightly different defect classification scheme, we can not look explicitly at how the subjects will classify the defects. To answer this question, we will have to take each set of defects that we have and classify them ourselves. Once we have multiple people having performed this classification, then we can compare the results and evaluate our defect classes.

EQ3 Do the specific categories reflect information that is useful to the process developers?
This is the other part of the question EQ1. We must examine each of the specific defect classes and determine if we can easily convert them to a question on a reading technique or to an item on a checklist if they don't already exist. If we have trouble converting the specific class to a useable step in a technique, then the class is not adequately described. Future work would involve empirically validating that the new step or the new checklist item had the desired result. This could be done through and experiment. In such an experiment, the same set of artifacts would be given to an inspector. This inspector would then be given the new technique. We would examine the defects found by the new inspector to determine if he or she found the defects that were missed in the previous inspection while continuing to find the defects that had been found in the previous inspection. While this experiment is not part of the proposed research, it would be a direction for a future study if we uncover enough specific defect classes that are not addressed by the current techniques.

3.3 Validation and Improvement of list of Experience Variables

Secondly, we want to make sure that the list of experience variables that we have defined is both complete and useful. We have to ensure that for each variable we have defined that we have a range of values. If all of the inspectors have the same value for a given experience variable then that variable is not very interesting. Or, it would be a potential topic for further investigation with another series of experiments. Because the experiments that are described in Section 3.2 come from a variety of domains, and also have variety of subjects, from students to professionals, we should have a good cross-section. This will help us to identify which of the experience variables really are important to study.

Based on the initial list of background and experience variables described in Section 2.2.3, we have some specific questions that need to be addressed. Each of the initial variables has a list of questions. They will be discussed in the following sections along with the metrics that will be used to measure them. In the case of each variable below, if we discover that the values reported by the subjects for that variable are all the same or have very little variation, then we must eliminate the variable from our consideration. The variable may still be an interesting one, and a future study may be planned to explore that variable directly. But, for our purposed, we will remove it from the list. This is represented by the tasks appearing on the right-hand side of the dashed line in Figure 4.

EQ4 Are all of the variables really variables?

This will be measured by examining the responses to the questions described in the following sections that deal with each individual variable.

- We will strive to have a normal distribution of inspectors within each value for each given variable.

If all of the inspectors have the same value for a variable, it clearly will not be interesting to evaluate. If we do not quite have a normal distribution but are close enough to get statistically significant results, then we will keep the variable in the list.

EQ5 Is the list complete?

This will be measure by evaluating the qualitative data provided by the subjects in each study. They will be required to answer questions to help provide us with insight:

- Did you find the technique useful? Why or why not?
- Were there other ways that you could have done this better? What are they?
- Do you feel that you were adequately equipped to perform the inspection? What else could have been done?

The responses to these types of questions will give us information about what the subjects in the study felt was important. If some subjects found the techniques they used to be helpful while others did not then their reasons for this will provide information into other potential variables between inspectors. If many of the subjects tell us that they were not adequately equipped and suggest something that would help, then this is a good candidate to add to our list of background and experience variables.

3.3.1 Environment Variables

EQ6 What is the environment like in which the inspection is being performed?

The answers to this question will be based on the following metrics that can be collected about the environment.

- Setting of the individual inspections
  - In their office with the normal noise and interruption of phone calls and coworkers
  - In a quiet room free of interruptions
  - Other

- Setting of the collection meeting (if it exists)
  - Someone's office with the normal workday interruptions
  - Quiet Room
  - Other

- Importance of inspections within the organization
  - Mandated by management for every project
  - Optional for each project but supported by management
  - Optional for each project but not supported by management
  - Discouraged by management

EQ7 Are there environmental constraints that affect how the inspection is being performed?
This question involves the answers to the following questions:
- Time/effort allowed by management for inspections?
  - None
  - < 10% of the development cost
  - between 10% and 25% of the development cost
  - >25% of the development cost
  - unlimited
  - unspecified

- Is there a standard defect report form for the organization that must be used regardless of the inspection process?
  - Yes
  - No

EQ8 Can the constraints from EQ7 change
This address how rigidly enforced the constraints from EQ7 are enforced
- Is the level of effort allowed for inspections a hard limit? I.e. do the inspections have to stop after a certain amount of effort has been expended regardless of their progress?
- If there is a standard defect report form, can it be modified as long as all the relevant data is still collected?

3.3.2 Training

EQ9 How does the training affect the application of the technique?
This is addressed by the following subquestions:
- How much time was spent during the training?
- Who performed the training?
- Did the subject attend the training?

EQ10 Are there ways to train more or less effectively?
This question will be based more on qualitative responses than quantitative ones.
- Do the experimenters see based on the results that a large portion of the subjects misunderstood one or more concepts from the training session?
- In the follow-up surveys or questionnaires do the subjects suggest things that they feel would have been helpful to have during the training session?
- Also in the follow-up surveys or questionnaires, did the subjects report that there were parts of the training that were not helpful, or they felt were a waste of time?

In order to address issues raised by one of the previous two questions, we would have to see the same answer or similar answers appearing in at least 1/3 of the subjects' responses.

EQ11 Are there other issues that should be considered when planning the training?
This question does not specifically address the application of the training variable to the finding of defects, but it does provide us with valuable insight into the training process and how it can be improved. If the training can be improved to a level where being present or absent from the training session has a significant impact on the outcome of the inspection, then this is valuable information.
- What time of day was the training performed? Morning, Afternoon, Evening

- Where was the training performed? In a quiet room where the subjects could concentrate, or in a more public place?
- Are there any other issues mentioned by subjects in their post-experiment reports that we have not yet thought of?

EQ12 If the training unequally emphasizes one aspect of the technique(s), does this skew the results when the technique is applied?
Again, this question does not directly address the main issue at hand, but the experimenters can use this information when planning and evaluating the training. This data will come from the information provided by the subjects in their post-experiment questionnaires or surveys.

3.3.3 Domain Knowledge

EQ13 Does experience or knowledge in a domain make the inspectors more or less effective?
The following questions will help to address this issue:
- What is the domain?
- How much experience does the inspector have in that domain? (None, Read in a book about it, Solved a problem in it before, Solved multiple problems in it before, Common domain)
- How familiar is the inspector with the product being developed?
  - **Requirements**
    - Did the inspector help create the requirements?
    - Is the inspector experienced with previous versions of the product?
    - If there are similar projects, is the inspector experienced with any of those?
  - **Design**
    - Did the inspector create the requirements?
    - Did the inspector inspect the requirements?
    - Did the inspector create the design?
    - If there are previous versions of the product, is the inspector experienced with those?
    - If there are similar projects, is the inspector experienced with any of those?

EQ14 Does the technique affect how domain experts perform
- Can we notice any difference in the performance of experts who use the technique from those who do not use the technique? The main idea is whether or not the technique gets in the way of the experts who would be more effective using their own method.
  - Number and type of defects detected

- Does the technique allow for the freedom of the expert to perform the same task in a way that they feel is better or more productive?
  - Qualitative data – domain expert may feel constrained and perform worse.

- Is the technique too detailed for domain experts?
  - This information will come from qualitative data collected in post-experiment questionnaires and surveys. If a large number of the experienced subjects report that the technique is difficult to use, and feel they would be more effective with their own way, then we know the technique may be too detailed.

EQ15 Does the technique affect how novices perform?
- Do the novice inspectors find more defects and different defects than they would have using an ad-hoc technique?
  - Number and type of defect reported by inspectors using a technique and inspectors using ad-hoc

- Is the technique not detailed enough for novices?

- Again, this information will come from qualitative data collected in post-experiment questionnaires and surveys. If a large number of the novices report difficulty understanding what to do, then the technique is not detailed enough.

3.3.4 Software Development Experience

EQ16 Does experience in a specific development technology (e.g. Object Oriented Design) make an inspector more or less effective?

- Inspector's experience in the lifecycle phase
  - None
  - Classroom
  - Industrial

- Inspector's experience in the specific technology, such as OO design, being used
  - None
  - Classroom
  - Industrial

- If the requirements inspection technique uses a technology such as Equivalence Partition Testing, inspectors experience in that technology
  - None
  - Classroom
  - Industrial

EQ17 Does the technique affect how development experts perform?

- Can we notice any difference in the performance of experts who use the technique from those who do not use the technique? The main idea is whether or not the technique gets in the way of the experts who would be more effective using their own method.
  - Number and type of defects reported by development experts using the technique
  - Number and type of defects reported by development experts using their own technique

- Does the technique allow for the freedom of the expert to perform the same task in a way that they feel is better or more productive?
  - Qualitative data – development experts may respond that the technique was too restrictive or not

- Is the technique too detailed for domain experts?
  - This information will come from qualitative data collected in post-experiment questionnaires and surveys. If a large number of the experienced subjects report that the technique is difficult to use, and feel they would be more effective with their own way, then we know the technique may be too detailed.

EQ18 Does the technique affect how development novices perform?

- Do the novice inspectors find more defects and different defects than they would have using an ad-hoc technique?
  - Number and type of defects found using inspection technique
  - Number and type of defects found using ad-hoc technique

- Is the technique not detailed enough for novices?
  - This information will come from qualitative data collected in post-experiment questionnaires and surveys. If a large number of the novices report difficulty understanding what to do, then the technique is not detailed enough.

3.3.5 Experience in another similar inspection process

EQ19 Does the inspector have experience in another similar technique?

- If the technique under study is a reading technique, Inspector's experience with reading techniques
  - None
  - Used a technique in an inspection in the same lifecycle phase in class
  - Used a technique in an inspection in another lifecycle phase in class
  - Used a technique in an inspection in this lifecycle phase in industry
  - Used a technique in an inspection in another lifecycle phase in industry

EQ20 Does this knowledge or lack of it, help or hinder the inspector in the application of a technique?

- How do the readers feel their experience or lack of it affected their application of the technique?
  - This will be collected from qualitative data provided by the subjects in post-experiment questionnaires and surveys.

3.3.6 Experience with this inspection process

EQ21 Do the inspectors get better or worse after prolong exposure to the technique?

- How much experience does the inspector have with this technique?
  - None
  - In Class
  - Industry

- Based on qualitative data, do the inspectors feel that they become more effective at using the technique by repetition?
  - Became more effective
  - Effectiveness remained the same
  - Became less effective

- Does the performance of the inspector improve, stay the same, or get worse each successive time the use the technique?
  - Number and type of defects detected during each inspection

EQ22 Do the readers tend to follow the techniques better or worse the longer they are exposed to them?

- Based on intermediate artifacts created, and qualitative answers from the subjects, do they continue to follow the technique after they become familiar with it, or do they assume that they understand the process and try to execute without following the instructions?

EQ23 Do the techniques need to evolve to become less restrictive as readers become more expert?

- Do experts feel, based on qualitative data, that they would be more effective if the technique told them the goals of each step, but allowed the inspector the freedom to accomplish that step in the best way they could think of?

3.3.7 Natural Language

EQ24 Is the reader familiar with the language that the documents and instructions are written in?

- Based on data collected in background questionnaires, we can know how familiar the inspector is with the language being used.
  - Spoken only
  - Written only
  - Spoken and written
  - Very familiar
  - Native speaker

3.3.8 Process Conformance

EQ24 Other than experience and domain knowledge, are there other reasons that inspectors deviate from the technique?

- Based on qualitative data from the post-experiment questionnaires and surveys, do a large number of inspectors state a reason for not following the process other than experience and domain knowledge?

EQ25 Is it always bad for readers to deviate from the process?

- Based on the qualitative data from the post-experiment questionnaires, and the quantitative data about the performance during the inspection we can determine if inspectors that deviate from the process justify it based on past experience and perform better than those that follow the technique?

EQ26 Is it possible to modify the technique so that it gives a "bounding box" instead of a "straight jacket"? Would this make things better or worse?

- Same idea as the use by domain and technique experts. Based on qualitative data, do the experts feel that the technique would be more useful if it gave them guidelines to follow instead of strict steps? If a large number of experts feel this way, then this hypothesis could be evaluated in a future experiment that is not part of this work.

EQ27 When the technique is not followed, can we determine the reason?

- Based on qualitative data, do the inspectors state why the deviated from the technique? If so, what is the reason? For example, did the inspector think the technique was not useful?
- Was the deviation done on accident?

3.4 Understanding and Validation of Linkage between Variables and Defects

Once we have a clear understanding of what our defect classification schemes are and what our experience variables are, we can begin to understand the correlation between the two. The assumption should not be made that a high value in a variable will always be helpful or beneficial. We have seen some instances in the literature where high domain knowledge can actually hinder the finding of defects. Sometimes someone with a fresh perspective can see things that someone who is very familiar with the material overlooks. It is conceivable that we could either confirm or deny this finding. It would also be possible for situations like this to occur with the other variables.

The research published in the literature has hinted at relationships between the variables we have defined and overall defect detection effectiveness. We want to confirm or deny the findings of other researchers published in the literature. The relationship between our defect classes and the variables are more difficult to predict. Indications were made in the literature that experts in certain aspects of the software lifecycle or the domain might perform better in that area of the inspection. This means that development experts may do a better job of uncovering defects dealing with object states, because they better understand what a state should or should not be. On the other hand, a development or domain novice might do better at uncovering some of the defects concerning data. Because they are looking at the artifact with a fresh perspective, they will not have some of the implicit assumptions that experts might have.

EQ28 Which of the variables contribute to finding the largest number of defects?

For each variable, we will split the inspectors up into groups based on their values for that variable. We will perform a 2-tailed t-test or ANOVA between the groups to see if any one finds significantly more or less defects than the other. This will be repeated for each of the background and experience variables that are in our list. There will be no new metrics collected to answer this question, but we will use the metrics discussed in the previous questions to get the number of defects as well as the groupings for the inspectors.

EQ29 Do some variables increase or decrease the likelihood of finding certain classes of defects?

For each variable, we will split the inspectors into groups based on their values for that variable. Then for each group, we will look at the number of defects from each of our defect classes that that group found. We will again perform either at 2-tailed t-test or ANOVA between the groups to see if any group finds significantly more or less defects in each of our defect classes. Like the previous question, no new metrics need to be collected here. We can reuse the data that was discussed earlier.

4 Summary

In this paper we have shown the necessity for inspections within the software development process. Software inspections are widespread. The environments in which they are used are varied as well as well as the inspectors. This variation leads to some discrepancy as to the results of the inspections. Researchers have pointed out that the variations among inspectors have an impact on the outcome of the inspection. While the researchers realize this fact, very little work has been done to determine what qualities or background and experience in an inspector has an impact on the outcome of the inspection. We have proposed an approach to this problem. We have discussed our method for evaluating the relationship between the background and experiences of an inspector to his or her performance in an inspection.

To do this, we have discussed three steps that must be taken. The first step we identified was to create defect classification schemes into which we could group related defects from different application domains. These schemes give us a way to measure not only inspectors performance in finding defects in general, but more specifically finding defects in a given defect class. Secondly, we must identify what variations between inspectors are important and useful to study. To do this, we have developed an initial list of background and experience variables. This list will be evaluated and possibly modified so that it contains the important differences among inspectors. Once we have this list, we can then explore the relationship between the variables and the defects. We will be exploring how the various background and experience variable affect the outcome of an inspection in two ways. First, we will examine each variable's impact on defect detection in general, i.e. how many defects are found. Secondly, we will examine how each variable impacts the discovery of defects from each of the defect classes that we have identified.

The results of this work will provide guidance to the inspection planner. We will provide a set of heuristics or guideline with which the inspection manager can choose inspectors for his or her team. Based on the organization's historical profile of defect types, we can suggest the background and experience that inspectors should have that will give them the best chance of finding those important defects.

References

[Ackerman89] Ackerman, A.F., Buchwalk, L.S., and Lewski, F.H. "Software Inspections: An Effective Verification Process." *IEEE Software*. May 1989. pp. 31-36.

[Basili81] Basili, V. R., Weiss, D. M. "Evaluation of a Software Requirements Document By Analysis of Change Data." In *Proceedings 5th Int'l Conference On Software Engineering*, IEEE CS Press, Los Alamintos, Calif., Mar. 1981, pp. 314-323.

[Basili84] Basili, V.R., Weiss, D.M. "A Methodology for Collection Valid Software Engineering Data." *IEEE Transactions on Software Engineering*, SE-10 (6): 728-738, Nov. 1984.

[Basili87] Basili, V.R., and Selby, R.W. "Comparing the Effectiveness of Software Testing Strategies." *IEEE Transactions on Software Engineering*, SE-12 (7): 1278-1296, Dec. 1987.

[Basili88] Basili, V.R., Rombach, H.D. "The TAME Project: Towards Improvement-Oriented Software Environments." *IEEE Transactions on Software Engineering*, SE14 (6): 758-773, June 1998.

[Basili96] Basili, V. R., Green, S. ,Laitenberger, O., Lanubile, F., Shull, F., Sorumgard, S., Zelkowitz, M.V., "The Empirical Investigation of Perspective-Based Reading"; *Empirical Software Engineering – An International Journal*, vol. 1, no. 2, 1996.

[Bush90] Bush, M. "Improving Software Quality: The Use of Formal Inspections at the Jet Propulsion Laboratory." *Proc. 12th Int'l Conf. Software Eng.*, IEEE CS Press, Los Alamintos, Calif., 1990, p 196-199.

[Chaar93] Chaar, J.K., Halliday, M.J., Bhandari, I.S. and Chillarege, R. "In-Process Evaluation for Software Inspection and Test." *IEEE Transactions on Software Engineering*, SE-19 (11): 1055-1070. November 1993.

[Chaar95] Chaar, J.K. "On the Evaluation of Software Inspections and Tests." In *Proceedings of the 8th International Quality Week*, Software Research, Inc., San Francisco, CA, June, 1995.

[Chillarege91] Chillarege, R., Kao, W., and Condit, R.G. "Defect Type and its Impact on the Growth Curve." In *Proc. Of 13th International Conference on Software Engineering*, May 1991, p. 246-255.

[Day93] Day, I. *Qualitative data analysis: A user-friendly guide for social scientists.* New York: Routledge. 1993

[Doolan92] Doolan, E.P. "Experience with Fagan's Inspection Method." *Software-Practice and Experience*. 22(2): 173-182. Feb. 1992.

[Dow94] Dow, H.E., and Murphy, J.S. "Detailed Product Knowledge is not a Prerequisite for an Effective Formal Software Inspection." In *Proceedings of the 7th Software Engineering Process Group Meeting*, Boston, MA, May, 1994.

[Fagan76] Fagan, M. E., "Design and code inspections to reduce errors in program development." IBM Systems Journal. 15(3). 1976. 182-211.

[Fagan 86] Fagan, M. E., "Advances in Software Inspections." *IEEE Transactions on Software Engineering* 12 (7): 1986. 744-751.

[Fowler86] Fowler, P.J. "In-process Inspections of Workproducts at AT&T." *AT&T Technical Journal* 65(2): 102-112, March-April 1986.

[Glaser67] Glaser, B. G., and Strauss, A.L. *The Discovery of Grounded Theroy: Strategies for qualitative research*. New York : Aldine de Gruyter. 1967.

[Glass99] Glass, R. L. "Inspections-Some Surprising Findings." *Communications of the ACM*, Apr. 1999, pp. 17-19.

[Gilb93] Gilb, T, and Graham, D. *Software Inspections*. Workingham, England: Addison-Wesley. 1993.

[Gilgun92] Gilgun, J.F., "Definitions, Methodologies, and Methods in Qualitative Family Research." *Qualitative Methods in Family Research*. Sage, 1992. pp. 22-29

[Grady94] Grady, R.B., and Van Slack, T. "Key Lessons In Achieving Widespread Inspection Use." *IEEE Software*. July 1994. pp. 46-57.

[IBM99] IBM, Center for Software Engineering. "Details on Orthogonal Defect Classification for Design and Code." IBM Research Whitepaper. 1999. Available at: http://www.research.ibm.com/softeng/ODC/DETODC.HTM

[Johnson94] Johnson, P.M. "An Instrumented Approach to Improving Software Quality." *Proc. 16th Int'l Conf. Software Eng.*, IEEE CS Press, Los Alamintos, Calif., 1994, p 113-122.

[Kelly92] Kelly, J. C., Sherif, J. S., Hops, J. "An Analysis of Defect Densities Found During Software Inspections." *Journals of Systems and Software*, Feb. 1992, pp. 111-117.

[Knight93] Knight, J.C., Myers, E.A. "An Improved Inspection Technique." *Communications of the ACM*. 36(11): 51-61, Nov. 1993.

[Laitenberger99] Laitenberger, O. and Atkinson, C. "Generalizing Perspective-based Inspection to handle Object-Oriented Development Artifacts." *Proceedings of 21st International Conference on Software Engineering*, 1999, p 494-503.

[Laitenberger00] Laitenberger, O., Atkingson, C., Schlich, M., and El Emam, Khaled. "An experimental comparison of reading techniques for defect detection in UML design documents." *Journal of Systems and Software*, 53 (2), August, 2000, pp. 183-204.

[Parnas85] Parnas, D.L., and Weiss, D.M. "Active Design Reviews: Principles and Practice." *Proc of 8th International Conference on Software Engineering*, 1985, p132-136.

[Porter 94] Porter, A.A., Votta, L.G. "An Experiment to Assess Different Defect Detection Methods for Software Requirements Inspections." Proc. 16th Int'l Conf. Software Eng., 1994, p103-112.

[Porter95] Porter,A. A., Votta, L.G. Jr., and Basili, V. R. "Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment." *IEEE Transactions on Software Engineering* 1995, 21(6): 563-575.

[Porter97] Porter, A.A., and Johnson, P.M. "Assessing Software Review Meetings: Results of a Comparative Analysis of Two Experimental Studies." *IEEE Transactions on Software Engineering*. Vol. 23, no. 3. March 1997. pp 129-145.

[Porter98] Porter, A.A., Siy, H., Mockus, A., and Votta, L. "Understanding the Sources of Variation in Software Inspections." *ACM Transactions on Software Engineering and Methodology*, Vol. 7, No.1, Jan. 1998, pp. 41-79.

[Russell91] Russell, G.W. "Experience with Inspection in Ultralarge-Scale Developments." *IEEE Software*. January 1991. pp. 25-31.

[Sauer00] Sauer, C., Jeffery, D.R., Land, L., and Yetton, P. "The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research." *IEEE Transactions on Software Engineering*. SE-26(1) : 1-14.

[Schneider92] Schneider, G.M., Martin, J., and Tsai, W.T. "An Experimental Study of Fault Detection In User Requirements Documents." *ACM Transactions on Software Engineering and Methodology*, April 1992, Vol. 1, No. 2. p188-204.

[Seaman97] Seaman, C.B., and Basili, V.R. "An Empirical Study of Communication in Code Inspection", in Proc. *ICSE '97*, pp. 96-106.

[Shull98] *Developing Techniques for Using Software Documents: A Series of Empirical Studies*. Ph.D. thesis, University of Maryland, College Park, December 1998.

[Shull99] Shull, F. Travassos, G.H., Carver, J., and Basili, V.R. "Evolving a Set of Techniques for OO Inspections." University of Maryland Technical Report CS-TR-4070, October 1999.

[Shull00] Shull, F., Lanubile, F. and Basili, V.R. "Investigating Reading Techniques for Object-Oriented Framework Learning." Accepted for publication by *IEEE Transactions on Software Engineering*. Available at: http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/postscript/framework_reading.ps

[Shull00b] Shull, F., Carver, J., and Travassos, G.H. "Process Evolution: A Toolkit and Example for OO Design Inspections." Submitted to ICSE. Available at: http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/pdf/icse2001.pdf

[Siy96] Siy, H.P. "Identifying the Mechanisms Driving Code Inspection Costs and Benefits." *PhD Dissertation, University of Maryland*, 1996.

[Travassos99] Travassos, G.; Shull, F.; Fredericks, M.; and Basili, V. R. Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Increase Software Quality. In *Proc. OOPSLA'99* (Denver CO, Nov. 1999), ACM Press, 47-56.

[Travassos99b] Travassos, G. H., Shull, F., Carver, J. "Evolving a Process for Inspecting OO Designs." XIII SBES: *Workshop on Software Quality*. Florianopolis, Curitiba, Brazil. October 1999.

[Votta93] Votta, L. G. Jr. Does Every Inspection Need a Meeting? *Proc. of ACM SIGSOFT '93 Symp. Foundations of Software Eng.* Assoc. for Computing Machinery, Dec. 1993.

[Weller93] Weller, E.F. "Lessons from Three Years of Inspection Data." *IEEE Software*. September 1993. pp. 38-45.

[Zhang99] Z. Zhang. *The Design and Empirical Studies of Perspective-Based Usability Inspection*. PhD. Thesis, University of Maryland, College Park, June 1999.