ABSTRACT

Title of Dissertation: THE IMPACT OF BACKGROUND AND EXPERIENCE ON SOFTWARE INSPECTIONS.

Jeffrey Clark Carver, Doctor of Philosophy, 2003

Dissertation directed by: Professor Victor Basili
Department of Computer Science

This dissertation is an initial study into the relationship between an inspector's characteristics and his or her effectiveness in an inspection. Research has shown that improving the individual effectiveness of the inspectors improves the overall effectiveness of an inspection team. But, the performance of inspectors varies widely, even when using the same inspection technique. This variation is often due to the inherent differences among the inspectors who used the technique. In order to better understand this variation and provide guidance to inspection planners, this dissertation has focused on the background and experience of the inspector as the source of variation.

To study this issue I used a novel approach for software engineering, grounded theory. This methodology allowed hypotheses to be built both top-down, from the literature, as well as bottom-up, using data. The literature portion came from software engineering as well as education and psychology. The data portion came from both existing studies and newly designed studies. The data from existing studies allowed the initial hypotheses to become more concrete. Once some of the hypotheses had support from data, the final step was to design studies to test a subset of the hypotheses.

I designed and ran two studies to test the selected hypotheses. The goal of the first study was to understand the type and level of experience with the software inspection process that was necessary. The earlier data had shown that process experience was important, but the effect of the type and level of experience was still unclear. The goal of the second study was to understand the interaction between an inspector's software development experience and the level of detail required in an inspection process. The earlier data had shown some indications that for experienced inspectors too much detail reduced the number of defects found, while less experienced inspectors needed more detail to overcome their lack of experience and find more defects. This dissertation presents complete list of hypotheses and the results of these studies along with some specific suggestions for both researchers and practitioners.

THE IMPACT OF BACKGROUND AND EXPERIENCE ON SOFTWARE INSPECTIONS

by

Jeffrey Clark Carver

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2003

Advisory Committee:

Professor Victor Basili
Professor Atif Memon
Dr. Forrest Shull
Professor Claude Walston
Professor Marvin Zelkowitz

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. Introduction

As software continues to become more prevalent, we cannot escape computers or the software they run. Software systems influence everything from our automobiles, to our cellular phones, to the air traffic control system. From small systems to large ones, software is playing an increasingly significant role in our world. While many of these systems work as advertised, unfortunately there are many that do not. These systems can function improperly for various reasons, such as poor workmanship, lack of time to do a proper job, or unqualified software professionals.

It can be argued that many of the problems in software today occur because building software is a difficult task. As the need for properly functioning software systems grows, so does the need for people to produce those systems. Thus, we find ourselves in a situation where there are increasing numbers of people developing software who were not properly trained in software engineering principles. This situation results in producers of software who are inadequately trained to do this difficult task. Therefore problems show up in the final product.

To address the problem of poor software quality, President Clinton, in 1999, convened an advisory committee to study software and information technology in general and its impact on our society. The President's Information Technology Advisory Committee produced a report [PITAC99] with their findings and some recommendations for the future. The findings of the committee expose some severe weaknesses in our software products and processes. In the next few paragraphs I will highlight some of the relevant findings.

The committee found that much of the software we as a nation depend on is fragile, meaning that the software has a tendency to work incorrectly or not at all, to be unreliable, to lack security, or to have performance problems. This fragility presents a real problem because of society's dependency on software for its infrastructure. There are many large and complex systems whose behavior is less reliable than desired. In addition to these software product issues, the findings also showed that the processes for producing software are inadequate.

So, based on the findings of the PITAC report, a series of goals can be defined. First of all, product quality should be improved. Section 1.1 will discuss the meaning of product quality in more detail. Better processes need to be put into place to work towards improved software quality. Based on this need, Section 1.2 will give a brief discussion of software process quality. Finally, those processes need to be evolved in order to improve them. Section 1.3 will discuss some of the process improvement mechanisms that have been put in place to aid in this evolution of process.

1.1 Product Quality

Because the software products are the most tangible assets in the software industry, they provide a logical place to start. Software that is operating can be measured and characterized more easily than the intangible processes used to create that software. While product quality and process quality are intimately related, process quality will be discussed separately in a later section. This section will begin by looking at aspects of product quality.

There are many dimensions of software quality that could be explored. While these different dimensions are all related, they each have slightly different foci and associated metrics and they aim to capture different types of information about the software. For each of these dimensions, a series of characteristics describes what should be true about a new software product. While each of these characteristics can be important in different settings, they are often at odds with each other, meaning that to get an increase in one dimension of quality requires a decrease in another dimension of quality. So, in a given project or environment, the software engineers must decide which dimensions are most important to maximize.

These different dimensions of quality have been called the "ilities" for short and include: efficiency, flexibility, improvability, predictability, repeatability, robustness, sustainability, visibility, portability, reusability, usability, testability, understandability, adaptability, correctness, reliability, availability, scalability, maintainability, and time to market. These "ilities" will be described in more detail in Chapter 2.

While the quality properties of the software products may be the easiest to observe and measure, the quality of the product is tightly coupled with the quality of the processes used to create that product. The use of higher quality processes provides a better chance of a higher quality final product. The next section gives a brief introduction to software process quality and explains some of the different kinds of processes.

1.2 Process Quality

Because the processes used in the development of a product have an important effect on the resulting product, it is important to characterize those processes. Processes can be either used for construction of software artifacts or for analysis of already constructed software artifacts. Additionally, processes can have both managerial and technical aspects.

The first method for characterization of a process is whether it is used for construction or for analysis of software artifacts. In both the construction and analysis phases, it is important to have good processes so that the products have the desired quality properties. Construction processes help software engineers build the most complete artifacts possible while injecting the smallest number of defects possible. On the other hand, the processes for analysis aim to help software engineers uncover as many defects as possible in the software artifact so that they can be removed from the artifact before it proceeds to the next phase of the lifecycle.

Secondly, processes can be viewed in terms of their managerial and technical aspects. The managerial aspects focus more on organizing the development activities and selecting the right technical processes. Examples include the SEI's Capability Maturity Model (CMM) [Paulk93] and ISO9001 [Paulk95]. On the other hand, the technical aspects give aid to the software engineer while doing their job. Technical aspects include things like design patterns, requirements elicitation techniques, and testing techniques.

1.3 Process Improvement

As discussed in the previous sections, processes are an important driver in the quality of the resulting products. For this reason, it is important to use the best processes available and improve and tailor those processes for each local environment. An "off-the-shelf" or "plug-and-play" process might be a good starting point for an organization, but it is definitely not an ending point.

Because every environment and to some degree every software product is different [Basili88], we should not expect any given process to be equally effective, without modification, for all projects in all environments. To combat this problem, it is necessary to evaluate and experiment with each new process and then evolve it to be more effective in a given environment or discard it if not effective. This evaluation involves observing and measuring the process while it is in use, and then feeding back the results and experiences of the project teams to allow the process to be tailored. More specifically, an organization should learn from itself and its own experiences. This idea of continual organizational learning has been formalized in the Quality Improvement Paradigm (QIP) [Basili85].

The QIP, which is an evolutionary process, is instantiated to function in a software organization as the Experience Factory (EF) [Basili94]. The idea of measuring a process in a given environment and then drawing conclusions about that process and feeding those conclusions back into the organization is the main thrust of the QIP and EF concepts. An effective way of doing this evaluation and feedback procedure is through studies and experiments on a process. When running an experiment, the researchers can normally control the environment so that one or more specific hypotheses about a process can be evaluated. These experiments allow the researchers to feed back specific results and lessons learned about the process. Then based on these results and lessons, the process can be tailored, if necessary, and then experimentation can continue.

This idea of using experimentation to evolve a process is not a new one. Forrest Shull used this idea in his dissertation work [Shull98]. The main focus of his work was to run a series of experiments to evaluate and improve a specific technique for requirements inspections. The main goal was to evaluate the steps in the procedure and determine how effectively they accomplished their task. Based on the results of the experiments, the steps could be modified for better results.

This dissertation examines the correctness of various software artifacts (requirements documents, design documents) in the context of a series of specific analysis processes. I will also use experimentation for process improvement to understand the variables associated with the analysis environment and the people involved in the analysis. The next section will present a discussion of the problem and its relevance to the field.

1.4 Problem Statement

While each of the quality properties discussed earlier is worthy of study and has an impact on the final quality of the products, I have chosen to focus on *correctness* using the measure of *defects*. Furthermore, because the product quality is largely a result of the processes that are used to create that product, the product quality problem is quite related to a problem of poor, or poorly understood software development processes.

Earlier, processes for software development were discussed, both for the construction phase and the analysis phase. These processes are not inherently good or bad, but their usefulness depends on the environment. Therefore, I have chosen one specific subset of processes to study in order to better understand their environmental context. This work will center on the process of software inspections including specific techniques used in those inspections. Both of these topics will be discussed in more detail in the next chapter.

Previous experimental work has been done to understand and improve software review and inspection procedures [e.g. Basili96, Porter97, Gilb93, Martin90, Parnas85] with respect to defect detection. Inspections have been shown to be an effective tool for defect detection in software artifacts. It has been reported that inspections help find between 60 and 90 percent of the defects present in software artifacts [Fagan86, Boehm01].

In addition, studies often report large variation between the least effective inspector and the most effective inspector. For example, in [Basili96], results are reported that the least effective inspector found only 10% of the defects, while the most effective inspector found 90%

of the defects. While [Schneider92], looking at inspection teams, found the least effective team to find 22% of the defects and the most effective team found 50% of the defects. Finally [Laitenberger00] reports results of a study showing that the least effective inspector found about 20% of the defects while the most effective inspector found almost 70% of the defects. These wide variations could not be explained by variations in process.

Chapter 2 will present a review of a variety of studies in the literature that deal with analyzing the effects of inspections on improving product quality, specifically in identifying defects. Each of those studies focuses on a specific problem or issue in the context of the quality of the software product. While there is much discussion on the processes used, there is little analysis of the effects of environmental characteristics or the people on the effectiveness of the process. Therefore, there was a need to extend the coverage of these previous studies to look at the environment and the people involved in the software development process.

The problem addressed in this dissertation can be described as follows. Because the processes used for building software impact the quality of the resulting software and a process' usefulness varies depending on the environment in which it is used, understanding the environment is important. More specifically the relationship between software inspections and the environment within which they are conducted is an open issue. Because software inspections are a human-based rather than machine based activity, a large part of the environment for inspections deals with the people who are conducting the inspection. So, understanding the individual differences among the inspectors is an important part of understanding the inspection environment. The problem that I will address in this dissertation is to determine the relevant background and experience variables that can characterize inspectors, including beginning to understand what effect these variables have on inspections, in terms of the overall number of defects found.

1.4.1 Hypotheses

Based on the above problem description, there is a set of research questions that need to be investigated. The more specific hypotheses will be discussed in Chapter 4.

1) Does the environment, including the inspectors, affect the overall number of defects found by the inspectors during an inspection?
2) Does the environment, including the inspectors, affect the number of defects within specific defect classes that are found during an inspection?

1.4.2 Impact of Results

In order to show that the above research questions are worth studying, a brief discussion of the potential impact of the results is provided. If the results from the study show the answer to the above question to be 'yes', then the questions were worth studying. But, not only will there be a positive impact if the results show positive answers, there will also be a positive impact if the results provide negative answers.

I will begin by discussing the impact of positive answers. In order for the results to answer 'yes' to the questions, there would have to be a relationship between the variable and an inspector's performance during the inspection for at least one of the identified variables.

If a relationship with the overall number of defects is detected is present, then the following guidance can be given to inspection team leaders:

1) If the team leader has the flexibility to choose the inspectors for the inspection team, then the results can provide a set of characteristics for identifying a "good" inspector.
2) On the other hand, if the team leader does not have the flexibility to choose inspectors for the inspection team, then the list of identified variables will give an indication of the areas to focus the training efforts to be of greatest benefit to the members of the inspection team.

Conversely, if the results do not show a positive answer to the questions, then one of the following situations could exist:

1) There really is no relationship between an inspector's background and experience and his or her performance during the inspection
    - If this is true, then it would indicate to an inspection team leader that it does not matter which inspectors they choose for the inspection team, because all inspectors will have approximately the same effectiveness. Therefore, choose the most cost effective team regardless of individual characteristics.
2) There is a correlation, but either there was not enough data to uncover it, or the experiments were not designed in such a way that the variables were highlighted. I can hypothesize that this situation is true if the qualitative data seems to consistently indicate a correlation between the values of a variable and the results of the inspection, but the quantitative data does not always agree.
    - This result is more positive for the researcher than for the practitioner. This type of result would mean that further investigation is necessary, including creation of a new series of hypotheses based on the results, and redesigning of experiments to address the specific issues uncovered.

2. State of the Art

Based on the discussion in Chapter 1, there are four major areas that impact this research. The first area is software quality. Because the overall goal of this work is to understand how to produce higher quality software, it is important to define quality and the aspects of quality we hope to achieve. Secondly, because the process used to produce software is one of the main drivers of the software's quality, a focus will be placed on the processes used to produce software. Third, not only is the study of processes important, but also the improvement and tailoring of those processes. Finally, validation techniques and processes are important for understanding the effects of the processes and the improvement to those processes. The validation methods help to understand, based on analysis of data, whether or not the hypotheses correctly describe the world of software engineering. In addition, the validation methods show the level of impact that the hypotheses are able to account for.

Section 2.1 will present some examples of the different aspects of quality and how researchers have measured them. It also contains a discussion of defect classification schemes. Section 2.2 then proceeds to discuss some different aspects of process that are important to software quality including a discussion of software inspections. Section 2.3 enumerates the methods for studying and improving software processes and some justification for why processes need to be improved. Finally Section 2.4 presents the techniques that are used to validate the results of studying processes.

2.1 Software Quality

As discussed in Chapter 1, *Software Quality* can be characterized by a series of properties referred to as the 'ilities'. Each of these quality properties captures a different aspect of the quality of a software system. While these properties all measure aspects of quality, they measure different and sometimes mutually exclusive aspects; therefore most quality models only capture one or a few of the properties.

The 'ilities' include: e*fficiency, flexibility, improvability, predictability*, *repeatability, robustness, sustainability, visibility* [McConnell02]; *portability, reusability, usability* [Dromey95]; *testability* [Voas95]; *understandability, adaptability, integrity*, *correctness* [McConnell93]; *reliability, security, availability, scalability, maintainability,* and *time to market* [Offutt02].

For each of the properties above, one or more models can be created to evaluate the given property. For example, *maintainability,* the ease with which a software product can be modified, has been modeled in different ways [Pearse95], [Takahashi97a]. Another important aspect of software quality, *testability*, dealing with the ease of creating test cases and the likelihood that successful testing indicates a defect free product, has been formalized into a model [Voas95]. Finally, *reliability* and *correctness* are also heavily studied quality properties [Hudepohl96], [Schneidewind97], [Takahashi97b], [Porter90], [Khoshgoftaar96]. Each of the models varies based on what is being measured. In addition, the level of human involvement and the skills required will also vary. Based on factors such as these, the level of confidence in the models can also vary. Next I will discuss two examples to demonstrate this variance. Also, I will show why some properties are more difficult to measure than others.

For instance there are some differences between the properties of *reliability* and *correctness*. First, in looking at *reliability*, the standard model to use is the Mean Time to

Failure (MTTF), which measures the average time between observed failures of the system. On the other hand, when looking at *correctness,* the model evaluates whether the artifact produced in one lifecycle phase is "equivalent" to the artifact produced in the next stage of the lifecycle.

While both of these models are based on *defects*, i.e. something in the product is incorrect, the MTTF model is based on *failures*, i.e. executing code that performs incorrectly, while the correctness model is based on *faults*, i.e. something written in a document that is not correct. This difference in the underlying metric displays one of the issues in measuring quality properties. While failures are the incorrect functioning of a running piece of software they are often easier to identify than to fix because they must be mapped back to the fault or faults that caused them, which is often quite difficult. On the other hand, to determine a fault, an estimation of the impact must be done. So, they are often harder to find but easier to fix once they are found because the location is known. Also, the numbers of faults and failures are not comparable because one fault may lead to multiple failures, or multiple faults may lead to the same failure.

Defect Classifications

Because defects are integral in determining the correctness discussed earlier they are central to the discussion of quality. In order to talk about and study defects, they are often grouped into classes. There are a few factors that influence the defect classification schemes: 1) the level of abstraction used in the classification scheme; 2) the document; and 3) the environment.

The level of abstraction deals with exactly what must be wrong in order to classify something as a defect. The IEEE standard defines three levels for this classification. The most concrete class is a *failure.* The next level, a little more abstract, is a *fault.* Finally, the most abstract class is the *error.* The error occurs when there is a misconception in someone's head [IEEE87]. The failures, which are used in the model for *reliability* discussed earlier, are the easiest to detect, because there is an observable problem in the execution of the software. Faults, which are used in the models for *correctness*, are more difficult to detect, because one must often speculate about the consequences of the fault, and cannot be sure if it is really a fault or not. Errors are the most difficult to quantify, because they require understanding where a person misunderstood something.

Secondly, the document being reviewed has an impact on the defect classifications. For instance, code defects are different from requirements defects. These differences occur because of the differing levels of specificity in the documents. While a code document is very specific, requirements documents, especially those written in natural language, are much more vague. In terms of concreteness, code defects are the easiest to identify because an inspector can more clearly see the result of a fault. This level of specificity makes requirements and design defects more difficult to identify because it is not always clear what the correct level of specificity is for that artifact. Therefore the inspector must often speculate on the consequences of the potential fault and make a determination, based on the assumed impact, whether or not a fault exists. In addition, because different types of information are represented in the requirements document, the design document, and the code, the defects will be different. Because of these differences, the defect classifications for requirements, design and code can look quite different.

The third aspect that impacts the defect classifications is the environment. Because of certain constraints in an environment the defect classification scheme might be different. In some cases, especially in safety critical situations, things that might not be defects in other settings become defects and vice versa. Additionally performance issues are environmentally dependent.

With all of these potential influences, different researchers and practitioners can easily describe defects in different ways, making it necessary to have a more formal way to talk about defects. Defect classification schemes are a more formal way of grouping defects together based on their characteristics. The reason for this grouping is that similar defects tend to be found in similar ways, or with similar frequencies and have similar types of fixes [Chillarege92]. There have been a series of studies that have shown that different classes of defects have different characteristics in terms of effort to repair [Basili84]. Therefore, in addition to discussing quality from the perspective of all defects in software, we can also discuss quality from the perspective of classes of defects present in software.

There are many different defect classification schemes present in the literature that vary depending on the specific goals, documents, and environment of the researcher who developed the scheme. Here some examples are discussed.

In research on requirements defects, Basili *et al.* [Basili96] developed a classification scheme consisting of five defect classes: 1) **Omission** – Important domain information has been left out; 2) **Incorrect Fact** – Information from the domain has been incorrectly recorded in the requirements document; 3) **Inconsistency** – The requirements document contains two or more statements that when taken together are not consistent or cannot both be true; 4) **Ambiguity** – Information in the requirements document can potentially have multiple interpretations all of which are not correct; and 5) **Extraneous Information** – Information has been included in the requirements document that while not incorrect, is not required. Other researchers also used similar schemes [Porter95], [Schneider81]. This scheme was designed to be abstract enough that it could also be instantiated for different software artifacts, such as design and code. These same five defect classes have also been tailored to Object Oriented design [Travassos99].

Other defect classification schemes have been developed for varying goals. One popular one, the Orthogonal Defect Classification (ODC), was created by IBM to focus on design and coding defects by classifying the defects based on when they are found and how they are fixed [IBM99]. Another defect classification scheme for requirements defects specifically was defined by [Ackerman89]. When a new inspection process is developed a defect classification scheme can be defined to understand the specific effects of the new process [Schneider92]. Finally, the defect classification scheme can be defined to help focus the attention of the reviewers on specific issues [Parnas85].

Each of the above defect classification schemes attempts to capture and quantify the important aspects of quality in a given environment. Looking back at the MTTF and correctness models discussed earlier, they both use a defects classification that allows similar fixes or similar prioritization to be given to all of the defects in a given category. But, because the focus of each model is different, the defect classification scheme will be different. The MTTF model classifies the failures based on severity, while the correctness models classify defects into classes based on the artifact being inspected. These classes typically deal with either what was done incorrectly in the document to create the defect or how the defect should be fixed.

In addition to the difference in defect classifications, users of different models might require a different skill set depending on the model of the defect classification scheme. In terms of the MTTF model, the following skills are required:

1) Knowledge of continuous mathematics;

2) Application domain knowledge, in order to understand if a failure has occurred the user needs to know what the software was supposed to do;
3) The ability to map failures back to faults.

On the other hand, for the correctness models, the following skills are required:

1) Knowledge of the process used to conduct the requirements, design or code inspection;
2) Application domain knowledge in a requirements inspection because of the lack of a document from a previous lifecycle phase for comparison; but unlike the MTTF model, domain knowledge is not as important for a design or code inspection because of the presence of an 'oracle' with which comparisons can be made;
3) Understanding of notation (code, design) for either inspection of that document, or for comparison during inspection of another document.

When these factors of defect classifications and necessary skills are taken together they can provide a better understanding of the confidence we should have in the results of any given model. For instance, because the MTTF model is based on observing actual incorrect execution of a piece of software, if someone has knowledge of the domain and how the software is supposed to function, then we can be more certain that identified failures are really failures. So, the confidence level is high. When looking at the correctness model, the level of confidence in the results varies depending on the lifecycle phase. In the coding phase, if an inspector identifies a defect in the code, there is high confidence that it will lead to a failure in the executing software. On the other hand, in a requirements inspection, the confidence level is much lower because the requirements artifact is much further away from an executing system than the code document.

Because defects underlie many aspects of quality, the remainder of this work will focus on defects. In order to consistently talk about defects, the focus will be on defects and their classifications. For the studies that have already been run, I will use the existing defect classification scheme. For the new studies the Basili *et al.* classification will be chosen as a base and evolved as necessary. This choice was made because it focuses on requirements and design, so it provides a good starting point for evolution.

This classification scheme identifies the types of defects that are studied in this work. These defects do not include cosmetic issues such as grammatical errors. The types of defects that are included are situations that would prevent the system to be built correctly. These situations include places where important information has been left out of the artifact, information has been recorded in the artifact that is incorrect, information in one part of the artifact is inconsistent with information in another part of the artifact, information in the artifact has not been specified at the right level of detail, or information has been included in the artifact that is not necessary.

2.2 Process

As mentioned earlier, the process of producing software greatly impacts the software product that is produced. So, in looking at software quality, it becomes necessary to understand the related software processes.

2.2.1 Construction and Analysis Processes

In the literature, processes can be differentiated by their foci. One set of processes is designed to aid in the construction activities of software. That is, they give guidance to the software engineer in various software development tasks (e.g. how to write a requirements document, create a design, code a module, create a test plan). On the other hand, there is a set of processes whose goal it is to provide guidance on how to examine software artifacts for the presence or absence of some set of quality properties, which will vary depending on the document and the setting. The analysis processes are normally done after the software artifact has been created, while the construction processes lead to the creation of the software artifact.

2.2.1.1 Construction Techniques

In order to effectively construct software, researchers have developed a number of techniques that encompass the construction phase of the lifecycle, e.g. development methodologies like the Spiral model [Boehm88], or the Waterfall model [Royce87]. There are also techniques that exist for specific phases of the lifecycle, elicitation and authoring of requirements (e.g. [Potts94]), creation of design, including Object Oriented [Rumbaugh91], Structured [Yourdon89], and Client-Server [Pfleeger98], the coding of the software, and finally the creation of the test cases [VanVeenendaal02]. The goal of the different construction techniques is to achieve a specified quality property in the software product. While these techniques are interesting and worthy of study, the remainder of this work will focus on techniques from the next set, analysis techniques.

2.2.1.2 Analysis Techniques

There are two major types of analysis techniques, testing techniques and inspection techniques. Testing techniques are a class of techniques in which the program or system is executed on the computer. The test cases provide the inputs to the program or system, and the results of those test cases can be examined either manually or automatically to verify that the proper actions were taken and the correct results produced by the system.

The second option for analysis is a class of static analysis techniques called inspections. While testing techniques operate by executing the program or system on the computer and analyzing the results, inspection techniques ask humans, individually or in teams, to review the software artifacts (e.g. requirements, design, code) without executing any code. The goal of the review is to verify the quality of the artifact. The faults the reviewer is looking for will vary depending on the specific goal of the inspection [Fagan76], [Parnas85]. Because inspections are more dependent on humans than testing, the rest of this work will concentrate on inspection techniques rather than testing techniques.

2.2.2 Managerial and Technical Aspects of Processes

Aside from being a process for construction or analysis, each process will typically have both a managerial and a technical aspect. The managerial aspect involves the way in which the process fits within the development environment. Additionally, managerial aspects might include such things as staffing requirements, time estimates, and constraints. On the other hand, the technical aspects deal with the actual process followed by the software engineer while doing his or her job.

For example the Software Engineering Institute's Capability Maturity Model (CMM) [Paulk93] is basically a managerial model. It instructs a software organization on which classes of process it should use to improve its software process, but focuses more at a high level rather than specific details of implementing the steps. Most of the processes are management process, e.g. use of configuration management, but the CMM does include one technical process fairly prominently, the process of inspections. In order for a software organization to gain a higher maturity rating (which is the goal of CMM) they must implement and begin to evaluate the inspection process within the organization. Again, as inspections figure so prominently in mature software organizations, they will be the focus of this work.

2.2.3 Inspections

There are many software review models that have been developed over the years. These models vary where the main defect detection activities take place. On one end of the spectrum is the *walkthrough* where the defect detection is done wholly during a team meeting. It is distinguished from the other types of reviews in that the author of the document is present in the review and "walks" the review team through the document. These types of reviews can often be at a higher abstraction level than the other types of reviews. The drawback to this type of review is that the results depend heavily on the quality of the presentation by the document author [Freedman90]. On the other end of the spectrum is a process that places the emphasis on individual reading of the software artifacts in preparation for a team meeting. More information on this type of review follows later in this section. Other inspection models fall on the spectrum between these two extremes.

For example, a *formal inspection* still places the emphasis of defect detection during the team meeting, and during the individual preparation inspectors are told to focus on familiarizing themselves with the artifact rather than looking for defects. The basis for the work on this type of inspection is a paper by Fagan describing a process for formal inspections as implemented at IBM [Fagan76]. According to Freedman, there are three characteristics that must be true of a Formal Inspection: 1) a written report; 2) active participation of everyone in the review group; and 3) all members of the team are responsible for the review [Freedman90]. On the other hand, the process of *Active Design Reviews* was proposed to address the shortcomings of traditional design reviews [Parnas85]. Instead of conducting one large meeting to find all types of defects, the review is divided up among the inspectors such that each review is focused on a different aspect of the design documents. Each reviewer has a checklist to use. Each reviewer then meets with the designer to resolve the issues uncovered during the review. This type of review adds a focus for each individual, rather than simply reviewing the document looking for all potential problems.

The term *inspection* has different meanings depending on the context. In this work, an inspection will be a static analysis technique used on some software artifact with a particular goal in mind. Inspections have been found to be of value in industrial contexts and have been highly studied by reasearchers. Michael Fagan first defined inspections as a way to aid in the detection of defects in design and code documents [Fagan76]. As defined in that paper, the focus of the inspection process was a team meeting in which the artifact to be inspected was read by the team, and defects were recorded based on discussions among the team. Members of the team were assigned a specific role during the meeting, and also spent some time individually, prior to the team meeting, acquainting themselves with the document to be inspected.

Based on the above description of an inspection, there are two major steps: individual preparation and the team meeting. The inspection techniques can be split into those two groups. In the original formulation of the inspection process, an inspection was a process whose goal was to aid a team to detect defects during a team meeting [Fagan76]. Since the initial paper on inspections, other researchers have called into question the need for a team meeting. Studies have shown that while the team meeting may add other value to the software development process, the number of defects detected during the meeting that were not detected during individual preparation time was relatively small [Votta93], [Porter97]. Based on these findings and the lack of work that had been done on techniques for individual defect detection, this work will focus on the individual preparation phase and techniques for individual defect detection.

Previous researchers have studied the process of software inspections, both from a technical and from a managerial point of view. Much of the research work in inspections has focused on how to structure the team meeting most effectively. Examples include the *N-fold* inspection process [Martin90] and the *Phased Inspections* [Knight93].

Because inspection techniques are static techniques, they do not require an entire functioning system to present in order to be used. Therefore, inspections are useful at any stage of the lifecycle process on any of the artifacts that are produced. Because each artifact has a different format (textual or graphical), and a different level of abstraction, the techniques for each artifact will be different. In the literature there are techniques that exist for inspection of requirements documents [Basili96], [Gervasi00], [Zhu02]; design documents [Laitenberger99], [Laitenberger00], [Travassos99]; code documents (e.g. [Linger79]); and even User Interfaces [Zhang99]. Code inspections have been studied more than any of the other types, and User Interface inspections are specialized for certain applications. Additionally, because of the benefits of early defect detection, this work will focus on inspections for Requirements documents and for Design documents.

In his dissertation, Harvey Siy points out that rather than focusing so much attention on the overall inspection process, researchers should focus more on the technical aspects of the individuals involved [Siy96]. In his work he points out that the inputs to the inspection, including the code (his work focused on code inspections) as well as the inspectors had as much of an impact on the results as the overall inspection process did. His work also uncovered the fact that the presence or absence of specific reviewers at an inspection made a large impact.

Based on the fact that the technical processes for the individual inspectors are showing increasing importance in the overall scheme of things, this work will focus on two specific inspection techniques. Additionally, while Siy found that the presence or absence of particular inspectors had an impact on the overall inspection, I will begin to understand not only if the presence or absence of certain inspectors affects the use of the inspection techniques, but also what characteristics about those inspectors are helpful or detrimental to their performance during the inspection. Because early defect detection has been shown to provide a cost savings, this work will focus on inspections of requirements and design inspections only.

2.2.4 Defect Reduction Studies

Because the goal of most inspections is to reduce the number of defects in a software product, a review of some of the relevant defect reduction studies that have been conducted is fitting. An article published in *Software* magazine [Boehm01] summarizes the results of multiple defect reduction studies into a Top 10 List of what researchers have learned about defect reduction.

One of the standard industrial methods for defect detection is to use a checklist. This checklist contains a series of questions, tailored to the environment, which guide the inspector in finding defects. Porter, et al. [Porter95] undertook a study to compare the checklist-based method with an ad hoc method and with a method based on using scenarios, which make an inspector responsible for only a subset of the defects. The study showed that focusing the efforts of the inspectors improved both the overall team effectiveness as well as the individual's effectiveness. It also showed that the industry-standard checklist method did not do any better than an ad hoc method. Finally, they determined that having the team members meet to compile a defect list was no more effective than simply compiling the individual lists. This last result indicates the importance of aiding individual inspectors in improving their effectiveness.

Another study was conducted to understand if modifying the size of the inspection team or increasing the number of inspections performed could increase the effectiveness of the inspection. The results showed that modifications to the process had very little effect on the outcome of the inspection. In addition, the researchers discovered that there was a large amount of unexplainable variation in the results, which led to further investigation into the inputs to the process rather than the process itself. The conclusion was that the inputs to the process, including the artifact and the inspectors themselves, had the largest impact on the results of the inspection [Siy96].

Continuing with the exploration of methods for individual inspectors, a study was run to explore the effectiveness of inspecting requirements using a specified technique versus an ad hoc procedure. A family of scenario-based techniques called Perspective Based Reading (PBR), which will be discussed in detail in Section 4.2.1 of this paper, for inspecting requirements documents, was evaluated, in the context of NASA, to determine whether or not there was any benefit to have a structured focused techniques for reviewing requirements over the normal technique used by the NASA engineers. The results of this study showed that defect detection effectiveness measured by the number of defects found was increased by using the PBR technique over the usual technique. In addition, there was some indication, though not statistically significant, that the experience of the inspector has an impact on the number of defects found [Basili96].

Based on the research showing the benefits of structured processes, one aspect of the process that has been hypothesized to contribute to the presence or absence of defects in the software product is conformance to the process guidelines. One process that has been specifically studied is the CMM [Krishnan99]. The argument is that if a software process is consistently followed, then the quality of the products will increase. The results of the study showed that process conformance significantly reduced the number of defects. In addition, they found that the personnel with higher capabilities also reduce the number of defects. In this case personnel capability is defined as only the technical capability of the team members.

Using these studies as a backdrop, an important focus has been identified for this work. It is clear that defect detection by individual inspectors is important. Therefore, support needs to be provided to aid those individuals in their task. However, because of the variance in the performance of individual inspectors, simple modifications to process models, and even to techniques for individual inspections will not be enough to fully understand and improve the inspection process. To better improve the defect reduction process, this variation among inspectors must be characterized and understood.

2.3 Process Improvement

Software processes are not plug-and-play or one-size-fits-all entities. For any given process to be effective, it must be tailored to its environment. Moreover, the process must be continually improved in order for its effectiveness to be fully realized. One successful method for doing process improvement is the Quality Improvement Paradigm (QIP) [Basili94b].

The QIP provides a framework, based on the scientific method, for systematically evaluating and improving a software process. It consists of six steps: 1) Characterize the environment – before any improvement can be done, an organization first has to understand their current situation through collection of representative metrics; 2) Set improvement goals – based on deficiencies in the processes or products of the characterized environment, researchers choose what aspect should be improved; 3) Choose process – after deciding what to improve, an appropriate process must be chosen that will help accomplish this improvement; 4) Execute process – the process must be executed in the environment and data collected; 5) Analyze results – the data from the executed process are analyzed to determine if the desired improvement occurred; and 6) Package and store experience – in order to help future improvement initiatives, historical records keep researchers from repeating studies without learning from past results. This set of steps is then continually repeated as the process is improved. This QIP process has been formalized in the context of a software development organization with the concept of the Experience Factory [Basili94].

So, the main idea behind the QIP is to make controlled improvements to a process or environment based on empirical data rather than speculation. After choosing the process that will be studied, the process must be executed. This execution must be done in an environment such that the process and the results can be measured. The measurement is necessary so that analysis can be performed and then a decision can be made concerning how to proceed with the process.

Empirical Studies in Software Engineering

One of the main tools for doing controlled studies about software process is the empirical study. Empirical studies in software engineering are different than in other laboratory sciences where the same study can be repeated many times with small changes. The approach to empirical studies in software engineering has to be different, because multiple replications are not always possible. In addition, software engineering studies involve software engineers or software developers whose time is expensive and therefore the studies are costly.

In software engineering, the steps involved in designing and running an experiment are as follows. First, the experiment must be defined, in terms of what is to be studied, the goals of the study, the domain, the scope, and so on. Secondly, the type of study must be chosen based on the goals and setting. Third the experiment must be executed. Finally the results must be analyzed [Basili86]. This methodology closely follows the first fives steps of the QIP described earlier.

The first choice that must be made is level of formality required for the study. The least formal, and easiest to perform is a simple survey, where subjects are asked a series of questions to collect information about current practices. The other two options are more formal, case studies and formal experiments. There are trade-offs between these two that provide guidelines on when each should be chosen. A *case study* sits in between a survey and formal experiment in the level of control over the environment. Case studies are easier to plan than formal experiments, but the analysis is more difficult, and the results are harder to generalize beyond the specific environment. Typically a case study is performed to evaluate a technique within an industrial or some other fixed setting. While the researchers have less control over the environment, the benefit is evaluation of the technology in a "real-world" situation. On the other hand a *formal experiment*, which is normally performed in a laboratory, and often with students as subjects, provides the researchers with the opportunity to control more variables within the environment. The drawback is the loss of the "real-world" aspect to the study [Kitchenham95]. So, depending on the environment and the specific goals of the researchers, one or more of these types of studies can be chosen.

After choosing the type of study, the researcher must choose the scope of the study, in terms of number of projects and number of subjects evaluated. The first option is a *single project* case study. This type of study examines one team on one project. The second type of study is the *multiproject variation.* This type examines one set of subjects across a set of projects. These first two study types are quasi-experiments or pre-experimental designs because they typically do not involve a control group. Third is the *replicated project study*. This type of study examines a single project across a set of teams. Finally, the *blocked subject-project* study examines multiple teams across multiple projects. The blocked subject-project type of study has a few variations. A *full-factorial* study means that all subjects use all treatments, and that every possible ordering of treatments is used by at least one subject. On the other hand, a *partial-factorial* design eliminates some of the treatment orderings for various reasons. Examples of these types will be given below. These replicated project and blocked subject-project study types are often run as controlled experiments, with a control group and an experimental group [Basili86]. These scopes are summarized in Figure 1. Each study type can be useful in different situations.

# Teams per project	# Projects	
	One	Many
One	**Single Project**	**Multi-project Variation**
Many	**Replicated Project**	**Blocked subject-project**

Figure 1 – Experimental scopes (from [Basili86])

Regardless of which type of study is chosen, data must be collected in order to determine the outcome of the experiment. It is as important to choose proper measures, as it is to choose the proper type of study. Not only must the right measures be chosen, but also the right scale for those measures must be chosen [Fenton94]. It is important to choose between the interval, ratio, and ordinal measures in order to collect the most accurate data, to get the most accurate results. Next we will revisit some of the studies discussed in the previous section to highlight the experimental aspects.

Experimentation has been successfully used in the context of inspections and reading techniques. The following paragraphs will highlight some of the related experiments, showing how the knowledge and state of the practice has progressed based on the results of these experiments.

The first study was conducted to understand the benefits and drawbacks of an inspection meeting over other methods of defect collection [Porter97]. In this study, two experiments were conducted and the results were compared and analyzed together to reach a conclusion. The first experiment was a *full-factorial blocked subject-project* study with only one factor being tested, the use of the group meeting. So, each team of subjects did a code inspection both with and without a group meeting, and the order of these two treatments was varied across the groups. In the second experiment the goal was to test whether it was better in a requirements inspection to a) individually prepare, but not record defects, and then detect defects during the team meeting; b) individually prepare and record defects, then have a team meeting to continue defect detection; or c) individually detect defects, and in place of the team meeting to spend more time individually detecting defects, and simply combine the lists of the team members to get the team defect list. Because of the large number of treatments a *partial-factorial blocked-subject project* design was used such that each team only performed two of the three treatments. When the results of the two experiments were combined, the results showed that having a team meeting did not significantly help or hurt defect detection effectiveness and that individual defect detection detected more defects than team defect detection. Based on this result, researchers have focused experiments on improving individual defect detection effectiveness.

In the study that compared a focused, scenario-based technique for individual inspections to checklist based and ad hoc inspections [Porter95] a *partial-factorial* design was used. This design is a type of the blocked subject-project study in which each subject is evaluated while using each of the experimental treatments but not necessarily all orderings. The use of partial-factorial design rather than a full-factorial design was necessary because the less formal methods could only be used prior to training in the more formal methods. For example, subjects could not be evaluated in ad hoc after being taught how to do scenario-based reading, because there is a danger that subconsciously the subject will use the scenarios as part of their ad hoc inspection. This design is useful for studies examining the use of a new technique to perform a task this is already being done in an ad hoc or unsystematic way. It allows researchers to compare the performance of each subject on the new method to their performance on the ad hoc or unsystematic (control) method. This study showed that a structured process for performing an inspection was more effective than an ad hoc or unstructured one.

In another example, Basili et al. used a similar *partial-factorial blocked subject-project* design to evaluate a specific type of scenario based inspection technique, Perspective Based Reading, against the normal technique used by professionals at NASA [Basili96]. Based on the same reasoning as the previous experiment, the subjects could only do PBR after doing their normal method. So each subject was able to act as his own control by doing the NASA method first (control) and then doing the PBR method second (experimental treatment). This study showed that PBR allowed teams to have a better coverage of defects in requirements documents than the traditional NASA process.

Forrest Shull, in his dissertation work, used a series of experiments to measure and improve the Perspective Based Reading (PBR) techniques for requirements inspections [Shull98]. In this work, he used a series of experiments to evaluate the PBR technique described in Section 4.2.1. This series of experiments were from a subset of the *blocked subject-project* type, called *repeated-measures*. In this type of study, each subject is evaluated under a series of treatments allowing for the results to be compared. The main focus of these studies was to evaluate the series of steps that make up the PBR techniques. The goal was to determine how the steps could be reordered or improved to allow for better results to be obtained when using the techniques.

Therefore, based on the fact that experimentation has been successfully used to evaluate and improve the steps in the reading techniques, I have chosen to employ a similar philosophy to explore the environment associated with the use of a reading technique. I will focus on the individual inspectors who are participating in the inspection. The idea will be to understand the individual differences among the inspectors and what impact those differences have on the inspection process. While the output of Shull's work was an improved set of steps for the techniques, the output of this work will be guidance on selecting the right people to perform an inspection, and also advice on how to train those people.

2.4 Validation Literature

Because of the expense of running controlled experiments in software engineering, the sample sizes are often small. These small sample sizes and limited opportunities for experimentation often make traditional statistical methods difficult or even impossible to use successfully. Therefore, other methods of data analysis must be explored in conjunction with the traditional quantitative statistical analysis. In addition to data analysis methods within software engineering, there are also useful methods that can be taken from the social sciences. This section will describe some quantitative and qualitative data collection and analysis techniques applicable to this work.

2.4.1 Data Collection and Analysis

In terms of data analysis, there are two main types of analysis that can be performed on software engineering experiment data, qualitative analysis and quantitative analysis. Quantitative data analysis focuses on statistical analysis of numerical data while qualitative analysis focuses on textual and numerical data that does not lend itself to the standard statistical testing.

The types of data analysis that can be performed on a data set also depend heavily upon the scale with which the data was measured. The data can be *nominal,* which allows data to be grouped based on equality. Secondly, the data can be o*rdinal*, which adds the ability to order the groups from least to greatest. Thirdly, data can be *interval*, which adds the ability to determine the equality of intervals or distances between groups. Finally, data can be r*atio*, which adds the ability to determine the equality of ratios [McClave95].

Quantitative Collection and Analysis

Some standard statistical tests can be used to compare data from different experimental treatments. The goal is to determine if one of the experimental treatments caused a statistically significant difference in the results. In order to use this type of data analysis, the independent variables in the experiment must be controlled so that the result being measure can be attributed to the experimental treatments.

There are a series of statistical tests that can be run depending on the data set and on the experimental design. The most common statistical test is the *parametric t-test* [Box78]. In this test, the results of two experimental treatments are compared against one another to determine if there is any significant difference between the two. In order to use this type of test, the data must be of at least interval scale and the observations must be independent of one another.

A related test to the t-test is the *Analysis of Variance (ANOVA).* This test extends the concepts of the t-test to more than two groups. So, three or more groups can be compared to determine if a statistical difference exists among them. The same requirements for the data exist here as for the t-test [Box78].

Qualitative Collection and Analysis

Many software engineering sample sizes are small and therefore difficult to show statistical significance so we have to employ qualitative data collection and analysis in order to supplement to the more common quantitative methods. Two popular methods of qualitative data collection that have been transferred from other domains for use in software engineering are protocol analysis [Singer96], and ethnography [Shneiderman98]. These methods involve collecting data about how subjects perform processes. The data collected includes information about what the subjects did as well as what the subjects' thought processes were as they solved problems.

In order to collect this type of data, researchers must employ two types of methods. The first type of methods is *retrospective*. These methods involve data collection after the process is complete, thorough post-mortems or questionnaires. The benefits of retrospective techniques are that they do not interfere with the execution of the process under study. Because they collect data after the fact, the process executor is allowed to focus on and finish the task at hand and then provide data. On the other hand, because the data is not collected until the end, there is an issue with the reliability of the information. Subjects have time to think about and formulate responses, so their response may not give a totally accurate reflection of what went on [VanSomeren94].

The second type of methods are *observational* and are used to collect data while the process is executing. These methods normally involve observation by the researcher of the subjects [Singer96]. The benefits of observational methods include more accurate data because the data is collected while the process is executing rather than after it is over. Also, there is no time for the subject to 'clean up' his answers before the researcher collects the data. On the other hand, there is the potential that the observation could cause the process to be altered. Some subjects might feel uncomfortable and act differently than they would if they were not being observed. This phenomenon is known as the Hawthorne Effect. The observational techniques are useful when the level of specificity of the data is at the level of individual steps in the procedure, rather than global information. One of the main techniques for observation is called *thinking aloud* [VanSomeren94]. A researcher instructs the subject to recite his thinking process out loud. Then the researcher can take notes to understand what is going on.

This type of data does not lend itself to the same types of statistical analyses that can be performed on the quantitative information mentioned earlier. In analyzing qualitative data, researchers must examine the mostly textual data to look for patterns. One of the important methods of doing the analysis is *grounded theory*, which will be discussed in Section 2.4.2. There are also a series of non-parametric statistical tests that can be useful if the textual data can be coded and grouped into discrete categories [Siegal56].

2.4.2 Grounded Theory

Based on both qualitative and quantitative results, hypotheses and theories can be built. 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 top-down based on assumptions that the researcher has *a priori*, the theory is formed bottom-up systematically from the data. This method came from the field of Sociology, and because this work concerns how people use inspection techniques, this technique can be useful. The stated purposes [Glaser67] are:

1) To enable prediction and explanation of behavior.
2) To be useful in the theoretical advancement of the field 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 the ideas of grounded theory researchers have created methods of qualitative research [Gilgun92]. Grounded Theory is defined 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 method 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. The steps that are important for this work will be summarized.

First, after a topic of investigation has been chosen, 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 this observation, 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 confidence could come 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 some rules for creating categories. This area of research applies to the creation of defect classes in this work. Each one of those classes is, in some sense, a category. So, the following rules are helpful to judge the 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.

These pointers will be used for evaluating and evolving both the list of experience variables as well as the defect classification schemes.

Because this method of theory building has come from social science literature, it is relatively 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.

3. Methodology

3.1 Motivation

Researchers often develop new and better techniques for software engineering. While these new techniques are often theoretically beneficial, in many cases little empirical support is given. In addition, it is rare to find researchers who investigate not only their new technique, but also the context in which the technique is to be used. There are many reasons for the lack of the second type of research, one being that it is very difficult to do.

This section will describe the methodology used to address the issues described above. The methodology will be based on a series of empirical studies that will provide data to back up conclusions drawn about the usefulness of various techniques within a given context.

The process of developing a list of relevant context variables is not a straightforward experimental problem. It is not possible to design and run one experiment, analyze the results, and draw a conclusion. Because of the complexity of the issues, a more involved methodology must be used. So, rather than simply running one controlled experiment, the results of a series of studies must be combined. The studies will be both studies designed specifically for the purpose of testing the research hypotheses and studies that were run for other purposes but which collected the necessary data to be evaluated in this context. The grounded theory methodology described in the previous chapter was used and is described below.

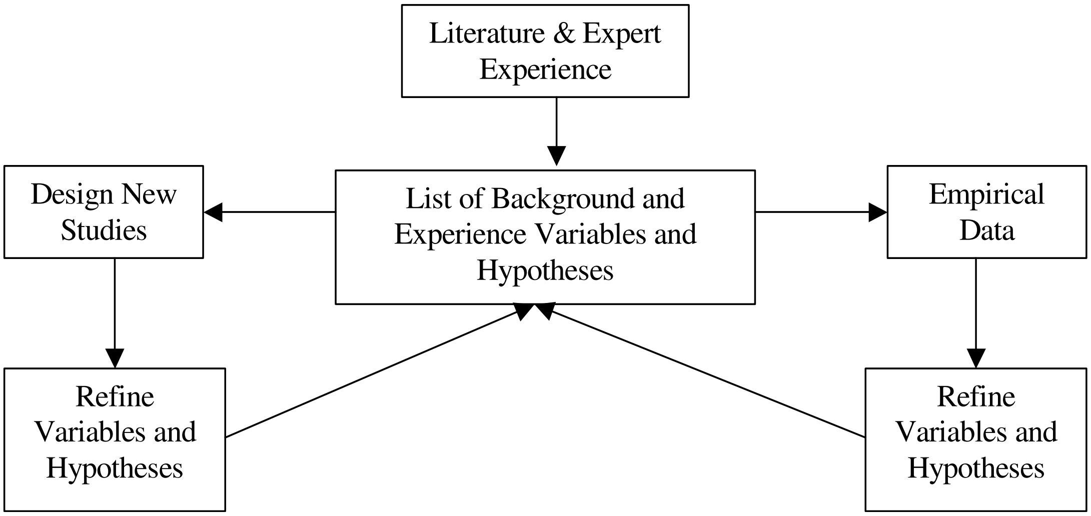

Literature & Expert Experience
Design New Studies
List of Background and Experience Variables and Hypotheses
Empirical Data
Refine Variables and Hypotheses
Refine Variables and Hypotheses

Figure 2 – Methodology for developing list of Background and Experience Variables

3.2 Developing a list of Context Variables

The first step in the process is to identify the potential context variables that have an impact on the inspection process. Because the people performing the inspection are an integral part of the inspection process, most of these variables will focus around the people. Because software inspectors all bring different types of knowledge and varied experiences, this work investigates the background and experience of the software inspector.

Figure 2 illustrates the process I followed to create the list of background and experience variables. The development of the list of variables had three major steps. Those steps consisted of both theoretical information as well as empirical information. I chose to use both theoretical and empirical information so that the list would combine the results of both approaches. By using this type of approach, I get the benefit of the many theories produced by other researchers, but those theories are balanced out by real data from actual studies. The theory helps to guide the studies, and the results of the studies help to refine the theory.

3.2.1 Literature and Theory

The starting point was the theoretical side gathered from conclusions drawn in studies reported in the literature. Some of these studies were backed with empirical support, but in general, the conclusions were arrived at as a byproduct of some other course of study. I looked for papers dealing with studies where the researchers discussed that individual differences in inspectors had some impact on their results. From this set of studies I focused on those studies in which the researchers were able to characterize or at least hypothesize about what the important differences among the inspectors were.

The second set of interesting literature was from outside the software engineering domain. Because I was looking at how people performed a step-by-step process, and how their background and experiences influenced that performance, other fields that study the application of similar processes by humans were prime candidates. The areas of sociology, psychology and education are fields that perform this type of study. While I did not expect to find information in this literature about how people would do software engineering processes, the software inspection processes are very similar to processes used in other places. Therefore, I thought that the sociologists and the psychologist might be able to shed some light on important characteristics for people performing a step-by-step process. There was also information in this literature about the ways people learn how to do new processes, which is also important and relevant to what I am studying, as one of my background and experience variables deals with the issue of whether or not the inspector was using a brand-new process, or using one that he had used before.

If the theoretical information gathered from the literature does not provide a rich enough set of variables with which to proceed, then that information must be augmented. One potential source of addition information would be to interview experts. Experienced researchers who have participated in the planning and running of a large number of experiments as well as those who have interactions with researchers in the field are a source of possible background and experience variables.

3.2.2 Data

The second major input to the list of variables is empirical data from studies. The data will provide the bottom up portion of the list. This portion of the list construction was done based on conclusions I was able to draw from the data using the concepts of *Grounded Theory* discussed in Chapter 2. Based on the results of each study, the existing hypotheses can be refined so that they continue to fit the observed data. While the hypothesis or theories are often initially formed based on the literature or expert opinion, they are refined based on the data from studies.

The data that will be used in this work will be of two types. First, a series of experiments have been conducted exploring other (but related) research questions. In many of these studies, the necessary metrics were collected to allow for a reanalysis of the data to fit into this work. This information makes up the right loop of Figure 2 (Empirical Data).

The second type of data used is from newly designed experiments. Based on a hypothesis or a small set of related hypotheses, an experiment can be designed. During this experiment, data will be collected and analyzed to draw conclusions. This information makes up the left-hand loop of Figure 2 (Design New Studies).

Each of the two loops concerning data can be iterated as many as times as necessary. Any time a new data set becomes available that allows the hypotheses to be evaluated, we can proceed around the right-hand loop. On the other hand, any time that we have the opportunity to select a hypothesis and design a study in order to specifically test it, then we can proceed around the left-hand loop.

3.2.3 Merge new data with existing list

In the "Refine Variables and Hypotheses" steps from Figure 2 the new data has to be merged back into the existing list of variables and hypotheses. A few different situations can exist: 1) The data and the existing list can agree; 2) The data and the existing list can disagree; 3) Items from the existing list can remain unsupported by the data; and 4) The data can lead to new additions to the existing list. I will discuss each of these situations below.

3.2.3.1 Data and Existing List agree

This situation is the easiest to deal with. When the data from a study or studies supports a theory or hypothesis that a particular background or experience variable has an impact on the number of defects found in the inspection, the hypothesis or theory is *supported.* This support can be at different levels, it can be anywhere from very weak to very strong. For instance, one study, that showed significant results with a relatively high alpha level, would provide a weakly supported hypothesis or theory. On the other hand, data from multiple studies, run in different environments with different subjects, all having the same significant results at lower alpha levels, would provide a much more strongly supported hypothesis or theory.

3.2.3.2 Data and Existing List disagree

This situation is a little more difficult to deal with than the first one. This category is characterized by the following: 1) The existence of a theory or a hypothesis correlating a context variable to the number of defects found in an inspection; and 2) Empirical data from a study that contradicts this theory or hypothesis. When this contradiction occurs, there are multiple potential explanations. First, it is very possible that the theory or hypothesis is incorrect. Secondly, the theory or hypothesis could be correct, but too broad, and based on the data from the study, the theory or hypothesis needs to be refined with some qualifiers. Lastly, the study could have been poorly run and not controlled, allowing other factors to potentially cause the variation in the results. The way to proceed here depends on which of the above situations is true. If the study was well designed and well run, then most logical way to proceed is to run another study to verify the results, and begin to refine the theory or hypothesis based on the resulting data.

3.2.3.3 Existing List is unsupported by the Data

In this case, there is simply a theory or hypothesis developed based on some experience or literature, but there is yet no data to support or refute it. The hypotheses in this category remain good candidates for future study, but have yet to be supported. To move a hypothesized context variable from this category to the supported category would require designing and running a study in order to collect empirical data. Then based on that data, the hypothesis can move either to the first or second category described above.

3.2.3.4 Data produces new additions to Existing List

Finally, the data collected from a study could produce a new context variable that had not been considered before. This type of information would normally come from the analysis of qualitative data collected on a study. If a large percentage of subjects made similar comments that would explain variation in the results, then this explanation might be the basis for creating a new context variable. In this situation the process is similar to the previous one. Although in the previous situation, we have no data to support the hypothesis, here we have data and have developed a hypothesis from that data. But, generally this new hypothesis will not have been the focus of the study it was created from. Therefore, a new study needs to be designed in order to specifically evaluate the newly hypothesized context variable.

3.3 Defect Classification Scheme

As mentioned in Chapter 2, in addition to correlating context variables with the overall number of defects found during an inspection, I also want to determine if the context variables have any correlation with the number of defects found of specific types. In order to evaluate the impact of the variables on specific defect types, a classification scheme for defects needs to be developed. This classification scheme will be based on previously used defect classifications, and will be augmented by some naturally occurring defects from some of my experiments.

The defect classification scheme used in this work was chosen based on the defect classification scheme that had been used in a series of previous studies. The data from those studies provides an important input into the development of the background variables, so it made sense to use that classification scheme, rather than create a completely new one.

Based on Figure 3, there were two software systems, the Loan Arranger and Parking Garage used as inputs. The Loan Arranger is a financial domain system that is responsible for bundling loans for resale to lending organizations. The Parking Garage system is a system that is responsible for the operation of a parking garage that has both monthly and daily tickets. The Parking Garage system keeps track of how many open spaces are available in the garage and only lets cars enter if there are spaces. More details on those systems can be found in Chapter 4.

For each one of these systems, defects were seeded into the artifacts. These seeded defects were created based on the experience of the researchers. The researchers chose defects that they believed were commonly occurring defects and ones that would be important for people to find. These defects are called “seeded defects”. The second set of defects from each of these systems that were used are called “real defects”. These “real defects” are defects that were inserted into the Parking Garage and Loan Arranger artifacts unintentionally by the researchers when the documents were created. This situation was possible because we were dealing with requirements and design documents rather than code documents. There would most likely have been fewer unintentional defects in a code document that could have been executed and its output examined. As each of these systems was used in different experiments, the subjects often reported defects that were not on the master defect list. Each additional defect found by the subjects was then examined and it was determined whether or not it represented a real problem with the system. If a newly reported defect turned out to be real problems with the system, it was added to the master defect list.

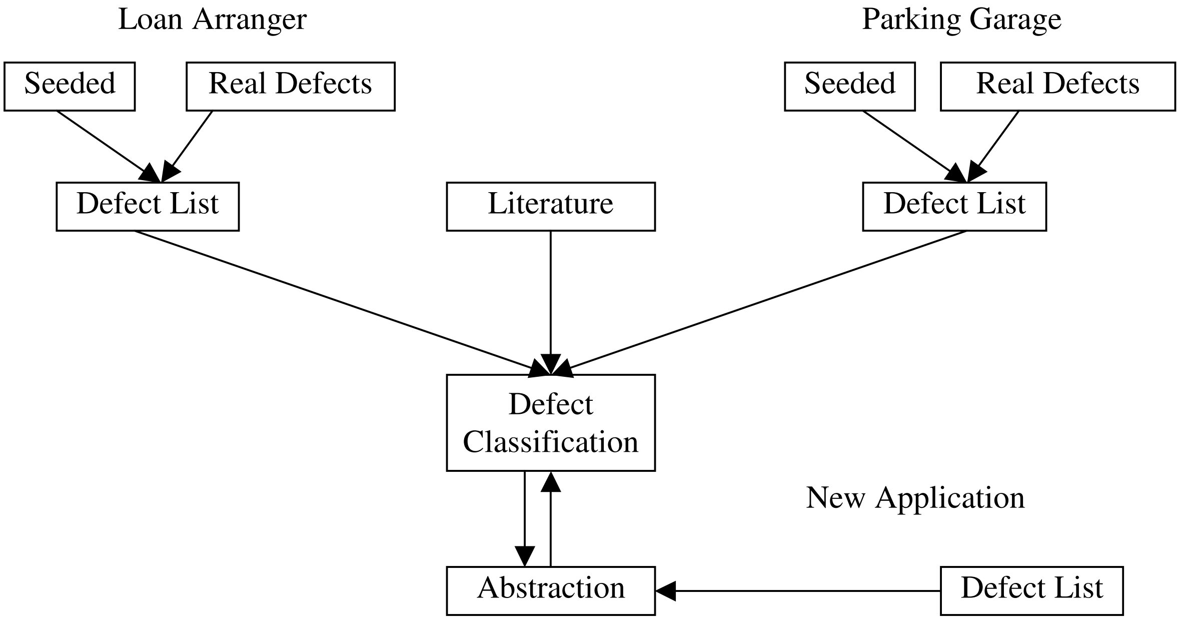


Figure 3 – Methodology for Defect Classification

The second input to the defect classification scheme was literature. In various reported studies, researchers classified defects in various ways depending on the goals of their studies. Because many of the studies had different goals, the defect lists were different. The focus of many of these studies was not to identify a defect list, but rather the defect lists were a necessary byproduct of the other research goals.

From these two sources came an initial defect classification scheme. The next step was to understand if the defect classification scheme was robust enough to cover naturally occurring defects in real systems. In order to evaluate the robustness of the defect classification scheme, data was available from studies on real projects at the University of Southern California (USC). These data sets were taken from the projects of the students enrolled in a graduate-level Software Engineering class. These students were working with real clients within the university to create real products, mostly dealing with systems to be used in the university libraries. The requirements and design documents were produced from scratch by the students, so all of the defects uncovered were naturally occurring (i.e. there were no seeded defects).

Based on the lists of defects produced during inspections, adjustments can be made to the defect classification scheme if necessary. The first step when analyzing the defects will be to abstract away as much of the domain specific information as possible to identify what the issue is. The next step is to examine the existing defect classification scheme and determine if this new abstracted defect fits in one of the defect classes in the scheme. If the new defect is covered, then nothing further has to be done. If it does not fit, either one of the existing classes can be extended to cover the new defect, or a new defect class can be created.

3.4 Validate context variables

For each of the variables that are identified as potential context variables, their impact must be validated. This validation needs to be done on two different levels. First each variable must be correlated with the overall performance of the inspectors during the inspection. This correlation is done by grouping the subjects based on their value for the given variable. Then, a statistical test is run to determine if being a member of one group or another has a significant effect on the number of defects found during the inspection. The second type of validation that must be done is based on the defect classes. The same type of evaluation described for overall effectiveness will be performed, but instead of correlating the context variable to the overall number of defects found, correlation will be between the context variable and the number of defects of a certain class that were found.

For each variable, there are different ways to perform the validation. Because the validation must be done based on empirical data, the two methods discussed here are different ways of getting access to such data. The first way is to design and run a new study. This approach is the preferred approach because the experimenter has more control over which variables to hold constant and which to vary to test the hypothesis. While this approach is preferred, it is also very expensive and not always possible to get a pool of subjects to run the study. Therefore, there also needs to be a secondary way to gain access to data. The second approach is to examine data from existing studies. Data from a study dealing with inspections where the experimenters have collected the necessary background and experiences on the inspectors, can be used to perform the type of validation described above.

3.5 Threats to validity of the overall series of studies

As is true with any series of studies, there are always threats to the validity of the results. This methodology is no different. Because the results are based on the analysis of many separate data sets, some interesting issues arise. From an external threat to validity viewpoint, there is the issue of whether or not the results apply outside the specific set of studies from which the data was taken. This threat is present here, but its impact has been limited by using a variety of studies from a variety of settings. The main threat here is the relative lack of data from professional subjects compared to the amount of data from student subjects. In terms of internal threats to validity, any time data from an existing study is used in order investigate a different hypothesis from the original study there are potential problems. These problems arise from the fact that the original study was designed to control certain variables to investigate the proposed hypotheses, in reanalyzing the data, a researcher cannot be sure if the assumptions made during the design of the study will confound the new analysis. In this work, existing data sets have been used as well as data from newly designed studies. By using both existing and new data, this threat can be addressed. If the results from the newly designed studies and from the existing data agree, then this threat has been minimized. In addition, the existing studies were all studying inspections, and necessary background information was collected on the subjects, so by re-using the data, I was not straying too far away from the original intent of these studies. Finally, by using data from multiple studies, set in different environments, with different subjects, often run by different experimenters, the threats to validity of each individual study are covered by another study within the group.

4. Background and Experience Variables

Chapters 4, 5 and 6 will describe how the grounded theory approach, described in Chapter 3, was used to develop and evolve the background and experience variables. Variables were initially identified from the literature, and were then continually refined based on data from empirical studies. Each new data set, or study run to evaluate one of the variables, increased the knowledge and helped to refine the list of variables.

The methodology from Figure 2 was followed to examine the background and experience variables. The first step, discussed in this chapter, was to examine the literature, both from software engineering and from other related disciplines, in order to identify an initial list of variables. The output of this literature review was a list of proposed variables based on theory, but not yet supported by empirical data. Section 4.1 discusses the process of identifying the initial variables. Section 4.2 discusses historical data from a series of studies and shows the first step of evolving the variables. Chapter 5 discusses the analysis of data from a series of studies that were run by a colleague in Brazil who allowed me to reanalyze his data to continue refining the hypotheses. Chapter 6 discusses two new studies that I designed and ran to test specific hypotheses generated during the work discussed in Chapters 4 and 5.

As discussed in Chapter 1 researchers have identified a strong relationship between a software process and the characteristics, including the quality, of the artifact produced by that process. It has also been established that variations in process do not completely explain the variation in the results of a process. In this chapter I will identify and discuss some of the other factors that affect use of a process to produce a product. Figure 4 shows the major actors in an inspection, the process, product and the human inspector. As the literature is discussed, different variables will be identified that relate to either the process being used, the product being developed or the human who is performing the process. As a conclusion to the next section, Figure 4 will be updated with the newly identified variables.

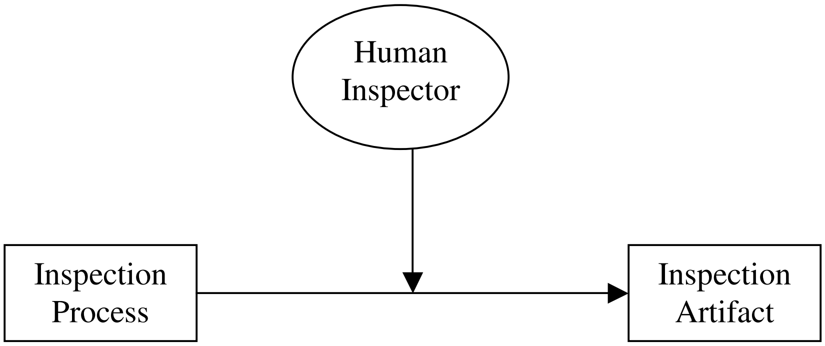


Figure 4 – Actors in the inspection process

4.1 Literature and Theory

In other areas of software engineering, the relationship between the characteristics of software developers and the quality of their final product has been established. In the definition of one of the most popular cost-modeling tools, COCOMO, Boehm et al. include the characteristics of the developers as a factor that influences the effort estimate produced by the model. The COCOMO model includes 1) Analyst Capability; 2) Programmer Capability; 3) Applications Experience; 4) Platform Experience; 5) Language and Tool Experience; and 6) Personnel Continuity [Boehm95]. By allowing these metrics to influence the effort estimate, Boehm et al. demonstrate that both general experience, such as analyst or programmer capability, as well as application specific experience are important issues to consider. Furthermore, if these variables impact the effort, then surely they also impact the quality.

A second example that indicates the relationship between developer characteristics and effort deals with project productivity estimation [Walston77]. In this paper, a series of metrics were defined that influenced the effort required for building a system. In the analysis, there was an almost linear relationship between effort and delivered lines of source code. But, there was considerable variation among the data points. In order to better understand the causes for this variation, the researchers investigated a series of metrics that were related to productivity. These metrics included technical issues such as access to development assets as well as people issues. Some of the important human characteristics that were defined were: 1) Overall personnel experience and qualifications; 2) Percentage of programmers doing development who participated in design of functional specification; 3) Previous experience with programming language; and 4) Previous experience with application of similar or greater size and complexity.

Based on the two well-known examples above, there was enough evidence to search the literature for more specific cases in which the background or experiences of an inspector had an impact on the results of a study. Because the focus was on software inspections, the literature search began with the software engineering literature. The goal of this literature search was to discover either conclusions related to individual characteristics of inspectors reached by other researchers, or conclusions that could be drawn from discussion of results by other researchers. After completion of this survey of the literature, a second literature search was performed in the education and psychology literature. The goal of this search was to determine if the conclusions drawn by software engineering researchers about characteristics of inspectors were true in the general case of people performing a process. In addition, I wanted to determine if any other characteristics had been reported in the education and psychology literature that had not yet been identified in the context of software engineering. Section 4.1.1 describes the variables that were found in the software engineering literature, while Section 4.1.2 describes the variables that were identified in the education and psychology literature. Because researchers from both groups identified some of the same variables, those variables appear in both sections.

4.1.1 Software Engineering

The literature search began in the software engineering domain, because the insights gained by researchers about the effects of human experience on software engineering processes will be most closely related to this work on the effects of background and experience of software inspectors on the software inspection. The main goal of examining this set of literature was to search for studies dealing with defect detection in software where variation among the performance of different inspectors was not explainable by variations in the processes used.

4.1.1.1 Application Domain Knowledge

One issue that has been recognized as having a potentially important impact on an inspector’s performance is the amount of application domain knowledge he or she has. The idea of application domain knowledge arises in various studies as a potential explanation for variability in the number of defects detected by different inspectors.

In a study conducted at AT&T, researchers noted that inspectors performing a requirements inspection do not have a document from a previous lifecycle phase to use as a reference. They contrast that with a design or code inspection where inspectors do have another document to use as a reference. Because of the lack of reference document in a requirements inspection it was concluded that **Application Domain Knowledge** can be an important variable because the inspector needed some knowledge of the domain in order to determine the correctness of the requirements document [Fowler86].

In a study conducted at NASA the researchers had a goal of determining how the pool of potential inspectors could be expanded. In their previous history the NASA organization only placed people who had application domain knowledge in their pool of potential inspectors. Their desire was to determine whether or not the pool of potential inspectors could be expanded to include those who did not have application domain knowledge. The results of the study showed that subjects who had less knowledge about the product being developed brought a fresh perspective to the inspection in the cases where there was an "oracle" document to use for reference, (e.g. code inspections referenced design documents and design inspections referenced requirements specifications) [Dow94]. Similar to the previous study, the results indicate that having **Application Domain Knowledge** might be important in the cases where an "oracle" is not present (i.e. a requirements inspection). On the other hand, application domain knowledge might not be important to have in the cases where an "oracle" is present (i.e. design or code inspections).

In the previous examples the researchers spoke directly of application domain knowledge. In the remaining studies the researchers did not specifically draw conclusions about application domain knowledge, but provided enough information in their discussion for conclusions about application domain knowledge to be drawn independent of the researchers. In the first such study, researchers suggested that because of the differences among applications and application domains each individual software project is really its "own little world". Because of this fact, they argue that for inspectors to find defects they must be able to "penetrate" this world. Therefore to do effective job, inspectors must be chosen from among those who are creating "similar worlds" to that of the project being inspected [Ackerman89]. The issues these researchers seem to be identifying, while not specifically stating it, is that the application **Domain Knowledge** of the inspectors is an important variable to be considered when choosing inspectors for an inspection team. This application domain knowledge allows the inspectors to focus their attention on defect discovery rather than spending a lot of time trying to understand the "world" that they are being asked to inspect.

In another study, conducted at the Shell Software Support Group (SSSG) at Shell Research, software inspection teams were using the Fagan method to inspect requirements specifications for large production systems. Due to the nature of this environment, both software domain experts (software engineers from SSSG) and application domain experts (geophysics) were working together on the same projects. A study was conducted in this environment, such that inspectors were drawn from 4 groups: 1) application domain experts (geophysicists); 2) The Quality Assurance group (testers); 3) the end users of the package; and 4) the software development experts (SSSG members who developed the software). This selection process was done such that each group of experts would be able to identify defects related to their expertise in the requirements document [Doolan92]. Again, the researchers here did not specifically state that application domain knowledge was an important issue, but by making two of the four groups of potential inspectors to be those who had application domain expertise (geophysicists and the end users) it is clear that the researchers must have believed that **Application Domain Knowledge** was an important issue.

In a similar example, the active design review method identified a series of inspections, each with a slightly different focus to be conducted on each software product. Four types of reviewers were identified as being important for these inspections: 1) Specialists (domain specialists); 2) Potential users of the system; 3) Developers who are familiar with the design methodology used, even if they are not familiar with the specific application being inspected; and 4) Those skilled at finding logical inconsistencies [Parnas85]. Because of the fact that one of the classes of reviewers proposed consists of domain specialists and another class of those who will be end users of the system, it appears that **Application Domain Knowledge** was an important issue.

Finally, in a follow-up to the initial study on inspections [Fagan76], it was reported that in most code inspections the inspectors were drawn from programmers on the project. On the other hand, the researchers suggest that the moderator of the inspection should probably be someone from another, but closely related, project [Fagan86]. This result was not based on a study conducted to validate the method for choosing inspectors; rather it was simply a report on the state of the practice. One of the potential reasons why inspectors were often chosen from the same project is that these inspectors would have more **Application Domain Knowledge** than inspectors who had not been working on the project.

Based on the results of these studies, both from researchers who specifically identified application domain knowledge as an important issue and tested it, and those who simply reported results that allowed for conclusions to be drawn about the importance of application domain knowledge, there was enough evidence to support the addition of **Application Domain Knowledge** to the list of potential variables to be studied in the remainder of this work. The impact of application domain knowledge was not clear from the series of studies presented above. It appears that in some cases application domain knowledge was an important asset for the inspectors to posses, but in other cases it did not provide any help to the inspector. While this variable seems to be important, due to the lack of certainty and consistency in the results, it is important to conduct further study on this variable.

4.1.1.2 Software Development Experience

The next issue that can potentially have an impact on the performance of an inspector during a software inspection is the level of software development experience they have. Depending on the situation, software development experience can include anything from experience writing requirements to experience creating designs, to experience as a tester, and many other things in between. The following studies are examples where software development experience had an effect on the outcome of the study.

A study was run to compare two requirements inspection techniques, an ad hoc requirements inspection technique and a scenario-based technique. In the ad hoc technique, the subjects were allowed to develop their own inspection methods. In the scenario-based requirements inspection method, the subjects were provided a scenario based on function point analysis. The results of the study showed that when using an ad hoc method, the subjects who had greater computing or information systems experience found significantly more design related defects than those with less experience [Cheng96]. The result from this study implies that, at least for one class of defects, the impact of an inspector's **Software Development Experience** on their performance should be considered.

Another group of researchers performed a replication of an experiment dealing with the N-fold inspection method previously discussed in Chapter 2. During the planning of this replication the researchers noted the fact that in the original study there was no control for the ability of the subjects. To combat this issue, subjects filled out a form to provide a list of computer science courses and grades, and the number of years of work experience [Schneider92]. By deciding to control for the ability of the subjects, the researchers seem to be indicating that they believed that **Software Development Experience** was an important issue to consider when studying an inspection process.

The active design review method mentioned in the previous section identified four types of reviewers as important to include in software inspections. One group was developers who were familiar with the design methodology used, even if they are not familiar with the specific application being inspected [Parnas85]. Because of this fact it appears that **Software Development Experience** was considered to be an important issue.

Similarly, in the study from the Shell Software Support Group (SSSG) discussed in the previous section, the researchers identified four groups from which inspectors would be chosen. One of those groups consisted of software development experts who were members of the SSSG and took part in the development of the software [Doolan92]. Again by making software development experts one of the sources for their inspectors, the researchers here implied that **Software Development Experience** was an important issue to consider.

Finally, in the study by Fagan discussed earlier, he concludes that any artifact that can be created can also be inspected. His basis for this argument is that the inspection of an artifact uses a similar skill set as the development process for that artifact [Fagan86]. Because the set of skills necessary to create an artifact are software development skills, in pointing out this similarity of skill sets, he seems to be indicating that **Software Development Experience** was an important issue to consider.

The studies discussed in this section provide a cross-section of cases where researchers have either specifically studied the impact of software development experience on their results or have indicated that software development experience was an important issue when composing teams of reviewers. Because of the information provided in these studies, **Software Development Experience** was considered an important variable to study in this work.

4.1.1.3 *Inspection Process Experience*

Another issue that appears to have an effect on the performance of inspectors is experience working with the inspection process. This type of experience can be experience in performing inspections, or it can be more specific experience in a particular inspection method or technique. Inspection process experience could be considered as a sub-piece of the software development experience described in the previous section. But, I have chosen to separate it out because the specific software process that I am studying is that of software inspections. So, it makes sense to look at this type of experience separately from the other software development experiences.

The first example is a study that dealt with the N-fold inspection method in which the researchers discussed a series of flaws in the experiment they were replicating. One of these flaws was mentioned earlier in the section dealing with software development experience. Another flaw that the researchers addressed in their replication was the non conformity of the experience the subjects had in conducting inspections. In order to address this issue the researchers tried to bring all subjects to the same level of knowledge through training sessions [Schneider92]. By recognizing that a subject's prior experience with performing inspections could have an impact on their results, these researchers appear to be indicating that **Inspection Process Experience** was an important issue to consider.

In another replicated study that compared a checklist based inspection technique to a scenario-based inspection technique, subjects were assigned one of the two techniques to use on two inspections. In their results, the researchers do not report a statistical analysis comparing the defect detection rates from the first inspection to the second inspection. On the other hand, based on the post-experiment questionnaires, the subjects did feel more comfortable using the scenario-based technique the second time they used it [Fusario97]. Because a scenario-based technique typically has more process involved in following it than a checklist-based technique, this result suggested that **Inspection Process Experience** was an important issue to consider.

An experiment was conducted at ORACLE Brazil using OORTs, discussed in Section 4.2.1, to inspect the design of a system to control the taxes collected by the Mato Grosso State's Secretary. In this study, one team of 5 developers used the OORTs to inspect the design created for the system. There was a wide variation in the number of defects reported by the inspectors. The inspector who was least familiar with the OORTs found only 12 defects, while the inspector who was most familiar with the technique found 57 defects [Melo2001]. While this result was from a small population and did not show any statistical significance, it did give some further indication that **Inspection Process Experience** was an important issue to consider.

A study was run at Bull HN Information Systems in which data was collected on more than 6000 inspection meetings conducted over a three-year period. One of the results was that in the second and third years of the study, the number of defects detected by the inspections increased. Comparing the overall number of defects found to the overall number of inspections conducted, it was found that the number of defects found increased faster than the number of meetings or the size of the inspected product. Based on this result, it was concluded that by the second or third year of the study the inspection process had stabilized within the organization [Weller93]. Based on these conclusions, it appears that **Inspection Process Experience** was an important issue because the number of defects detected increased as the inspectors got more comfortable with the process.

In the NASA study mentioned in the Section 4.1.1.1, there were some environmental conditions defined that were necessary to conduct the study. One such condition was that the subjects should not have been exposed to inspections prior to the course in which the study took place. They contrast this situation to the normal industrial inspection setting where inspectors will often have performed one or more prior inspections [Dow94]. By explicitly comparing the experimental setting and the industrial setting in terms of experience with performing inspections, the researchers indicate that **Inspection Process Experience** was an important issue to consider.

Finally, in the AT&T study also mentioned in Section 4.1.1.1, the researchers noted that because it is especially difficult to inspect a requirements document, the inspection relies heavily on the expertise of the inspectors [Fowler86]. Identifying this need for requirements inspectors to be experience in the inspection process suggests that **Inspection Process Experience** was an important issue to consider.

The studies in this section have provided some examples where researchers have encountered **Inspection Process Experience** as an important factor in the outcome of their studies. In some cases process experience was directly addressed and evaluated, although only with a small population. In other cases the discussion of the results of the studies provided some indication that a subject's level of process experience was a factor that needed to be considered. Because process experience appears to have an impact on the outcome of these various studies, and there do not seem to be any studies that directly address the issues with a large enough population, **Inspection Process Experience** was an interesting variable to study. Therefore, it was added to the list of variables studied in this work.

4.1.1.4 Training

Another issue that appears to have some impact on the performance of an inspector is the training received in the inspection process. Training includes both classroom lectures that explain the theory behind and use of the method and laboratory time where inspectors can practice the new technique and ask questions of an expert.

In the study by Ackerman et al. that was discussed Section 4.1.1.1, the researchers emphasized the importance of the training that the inspections receive. In fact, the suggestion is that as part of the training procedures for inspectors a practice inspection should be used rather than only lecture training [Ackerman89]. While there was no evaluation to judge the effectiveness of training with a practice session versus training without a practice session the fact that the point was raised in the discussion indicates that the **Training** inspectors receive probably has some impact on their performance during the inspection.

Also, in the study comparing a checklist based inspection method to a scenario-based method that was discussed in Section 4.1.1.3, the researchers discussed the importance of subjects receiving training in the exact technique they would use during the inspection process. As part of the training the subjects were allowed to do a practice inspection using only an Ad Hoc procedure. The subjects reported that they would have rather had time to practice the scenario-based technique during their training because they did not feel comfortable with the technique until the second time they used it [Fusario97]. This result suggested that the type of **Training** the subjects receive was an important issue to consider.

The studies discussed above highlight the fact that training is an important issue for inspectors when learning a new technique. In both cases, the researchers identified the importance of allowing the subjects to do a practice inspection prior to the experiment. Because the different types of training were not evaluated, i.e. comparing subjects who were allowed to practice to those who were not allowed to practice, there can be no concrete conclusions drawn at this point. But, because the **Training** issues appears to be important and has not been adequately studied, it was added to the list of variables being considered in this work.

4.1.2 Education and Psychology

After reviewing the software engineering literature to determine some variables that affect the performance of inspectors during a software inspection, it also became clear that the learning and execution of processes was not specific to software engineering. In order to better understand the problem, literature from education and psychology was reviewed looking for conclusions that had been drawn about characteristics that might affect the way someone learns or executes a process. The insights gathered here were more general than those from the software engineering literature and will either supported those discovered in the software engineering literature, or offered new directions to consider within software engineering.

4.1.2.1 Process Experience

In a paper discussing a study done with novice and expert programmers, Soloway et al. note that expert and novice programmers behave in different ways. The experts tend to have two major pieces of knowledge that the novices lack. First, the experts understand *programming plans*, which are commonly used fragments or excerpts of code. Secondly, experts understand the *rules of programming discourse*, basically the means of communicating with one another, including programming conventions, and variable naming schemas [Soloway88]. While this research deals with programmers, the ideas raised are applicable to inspectors. In order to gain the same types of knowledge that benefit expert programmers, inspectors would need increased **process experience** in the inspection process, and possibly even increased technique experience with the specific techniques being studied.

4.1.2.2 Application Domain Knowledge

In a paper from the psychology literature researchers hypothesize that there are differences between the way application domain novices and application domain experts design software. When the experts from the study were examined it was found that there were some activities that were common among them that were not present among the novices. First of all, the experts tended to make mental models before proceeding with the design. Second, the experts tended to have the same levels of abstraction at different points in the procedure. Third, the experts took notes about issues that need to be addressed at a later point. Finally, the experts did mental simulations of the partially completed designs [Adelson88]. The specific activities the experts performed are not as interesting as the fact that the domain experts performed similar activities that helped them complete their task. The commonality supported the argument that **Domain Experience or Knowledge** has an impact on the effectiveness of inspections.

4.1.2.3 Training

As would be expected, the education and psychology literature have more to say about the issue of training people in new processes. The literature here will get into some more specific ideas of how the training should be conducted. While some of those ideas may be applicable to training in the software engineering domain, the others will serve to show that there is little doubt that the type of training received has an impact on the process.

The Socratic method of teaching is an informal process where an instructor works one-on-one with a student. This method of teaching has been formalized into a procedure based on a set of rules. Those rules then are divided up into those that abstract general principles from known cases and those that take general principles and apply them to unknown cases [Collins77]. These two methods provide different ways of training for an inspection process. One method would be to start with a series of specific cases and abstract the inspection procedure from those cases. Another method would be to start with the inspection procedure and then use specific cases to exemplify what is taught. While it is not clear which of these methods would be more effective, the fact that they can be discussed suggests that the **Training** procedures students are given when learning a new task or a new class of tasks can have an effect and need to be studied further.

Another aspect of training is the transfer of knowledge from experts to novices. The goal is to determine how novices can be more quickly trained and become experts. One method for doing this training is to use an apprenticeship model. In this model of training, the expert first performs a task while the student or novice observes. Then the student or novice performs the task while being coached by the expert. As the student becomes more proficient in the task, the expert coaches less. Finally, the student no longer needs the expert [Collins89]. A similar concept was also noted by [Pintrich96]. The discussion in this paper shows yet another method of conducting **Training**. Again, by studying the problem of transferring knowledge from experts to novices, the researchers have identified an important issue.

Another paper discussing a set of studies about the way young students learn in the classroom presents several interesting differences between students who learn effectively and those who do not. The studies discussed in the paper deal mostly with 2nd-5th grade students, but many of the findings can be generalized to any students learning new material. The results show that the types of training students receive can be very important. The following suggestions are made when training students: 1) Set clear goals for the students, including linking the material to previously studied material; 2) Monitor the activities of the students; and 3) Provide feedback, positive or negative giving the students an idea of how they are progressing [Berliner77]. While these results arise out of studies with children, some of the conclusions have broader application in terms of training. These results again point to the fact that **Training** was an interesting and relevant variable to study.

4.1.2.4 Motivation

A new issue that became evident when examining the education and psychology literature was that there was more to learning than just the material presented or even the format of the material presented. A paper discussing ways in which people acquire knowledge reviews a series of relevant work and finds that learning is not merely a cognitive activity. There appears to be a strong component of effective learning that deals with the interest the student has in learning [Alexander00]. This idea of **Motivation** becomes relevant when inspectors are learning either the process of inspections or a specific technique for doing the inspection. Based on the findings in this paper, it appears that studying how to properly motivate inspectors could have an impact on how well they are able to learn a new inspection technique.

4.1.3 Initial List of Variables

After combining the variables collected from the above sources, an initial list of background and experience variables can be compiled. Figure 5 below is an updated version of Figure 4, showing how these variables relate to the inspection process, inspection artifact, and human inspector. In general, all of these variables are related to the human inspector, but in the figure the variables are attached to the actor that is impacted the most by changing the variable. For example, while application domain knowledge is an attribute of the inspector, its impact is seen only in relation to the artifact that is being inspected. This list is only the starting point.

At this point none of the variables on this list had been supported with additional data, not appearing in the original studies. In the subsequent sections, these general hypotheses will be refined into a series of specific hypotheses that deal with collectable metrics. Data will be provided to support and expand some of these hypotheses.

The initial list, along with a brief definition and the source of each, is:

3) ***Application Domain Knowledge (DK)* –** This variable deals with the inspectors' knowledge, experience, and familiarity with the problem domain of the software system being inspected.
   a. Software Engineering Theory Literature
   b. Education and Psychology Theory Literature
   H.DK - Inspectors who have more domain knowledge will find more defects during an inspection than inspectors who have less domain knowledge.

4) ***Software Development Experience (SD)* –** This variable deals with the level and number of years of experience the inspectors' have in various aspects of software development, ranging from general experience, to experiences in specific lifecycle phase activities.
   a. Software Engineering Theory Literature
   H.SDE – Inspectors who have a higher level of Software Development Experience will find more defects during an inspection than inspectors who have less experience.

5) ***Inspection Process Experience (PE)*** – This variable examines the experience that the inspectors' have in performing the process of inspections.
   a. Software Engineering Theory Literature
   b. Education and Psychology Theory Literature
   H.PE – Inspectors who are more experienced in the inspection process will find more defects than inspectors who are less experienced.

6) ***Training (T)*** – This variable deals with the amount and type of training that the inspectors receive prior to performing an inspection.
   a. Software Engineering Theory Literature
   b. Education and Psychology Theory Literature
   H.T – The amount and type of training inspectors receive will affect the number of defects that they find during an inspection.

7) ***Motivation (M)* –** This variable deals with the desire of the inspectors to learn a new inspection process and to perform that process well.
   a. Education and Psychology Theory Literature
   H.M – The level of motivation that the inspectors have will positively affect the number of defects that they find during an inspection.

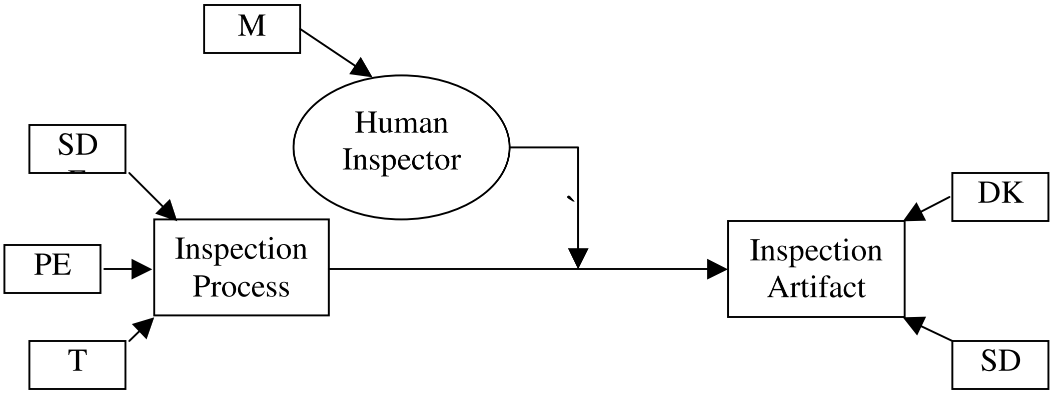


Figure 5 – Experience Variables

4.1.4 Expert experiences

Because there was enough information available from the literature to create a initial list of variables, it was not necessary to seek out additional experts. Had the literature not yielded a set of interesting, relevant variables, some experts in inspections and defect detection would have been interviewed to increase the richness of the initial list of variables and hypotheses.

4.2 Data from Existing studies

The second step in the grounded theory methodology presented in Chapter 3 was to begin to refine variables on the initial list that were discovered during the literature search. This procedure allowed variables to be defined in terms of more concrete metrics. The empirical data from the studies discussed in this section will either support, refute, or refine one or more aspects of the variables presented above. An output of this analysis of data was a series of hypotheses about the relationships between the variables, and the more specific metrics, and the defect detection rate of an inspector during a software requirements or design inspection.

4.2.1 Brief description of reading techniques

In many of the studies discussed in this section, specific techniques were used to help the inspectors find defects. Two such techniques, generally referred to as "reading techniques" because they provide guidance for an inspector to read a software artifact, are briefly described in this section.

4.2.1.1 Perspective Based Reading (PBR)

The Perspective Based Reading (PBR) techniques were developed to support the inspection of a requirements document. The main idea in PBR is that for a requirements documents to be correct, it must satisfy the needs of each of the stakeholders who will use the requirements document. Therefore, PBR identifies each of these stakeholders and assigns an inspector to inspect the requirements from the perspective of each stakeholder. Each PBR perspective consists of a model of abstraction of the requirements that is relevant to that perspective. For example, the perspective of a tester would use the model of test cases, while the perspective of a designer would use the model of a high-level design. After identifying the model, each perspective then includes a series of questions aimed to detect the various types of defects that are relevant in a given environment. The steps of creating the abstraction model and the defect detection questions are then put together into a step-by-step technique to be followed. More information about PBR can be found in [Shull00b].

4.2.1.2 Object Oriented Reading Techniques (OORTs)

The Object Oriented Reading Techniques (OORTs) were developed to aid in the inspection of the various artifacts in an object oriented design. Because there are multiple artifacts in the OO design, and each of these artifacts views the system from a different perspective, it is not always obvious how to compare the artifacts. There are two sets of OORTs. The first set provides guidance to the inspector on comparing the various design artifacts to each other to ensure they all describe the same system. The second set of techniques provides guidance on comparing the various design artifacts to the requirements to ensure that they describe the right system. More information about OORTs can be found in [Travassos02].

4.2.2 Brief description of the artifacts used in the studies

The experiments and results that will be described in the next few sections are all based on inspections of artifacts from a small set of problem domains. This section provides a brief description of each of those domains.

4.2.2.1 Parking Garage Control System (PGCS)

The PGCS was responsible for controlling the various entries and exits of a parking garage. The document included requirements to deal with the purchase and use of both monthly parking passes, as well as daily (pay for each use) parking passes. Both requirements and design artifacts from this domain were used in the studies.

4.2.2.2 Loan Arranger (LA)

The LA system was responsible for organizing and bundling loans for resale by a financial organization. The system allows organizations to purchase loans from banks, classify them based on risk and other characteristics, and then create bundles based on provided criteria for resale to other investors or banks. The system was relatively small, but had some non-functional requirements that made it more complex. Both the requirements and design artifacts from this system were used in the studies.

4.2.2.3 Automated Teller Machine (ATM)

The ATM system was responsible for control of the cash machine and its interface to the customer. Complexity of the system is reduced by focusing only on requirements for the user. The proper functioning of bank computers and other associated systems is assumed and is not dealt with in this system. Only the requirements document from this system was used in the studies.

4.2.2.4 NASA Artifacts

In the two NASA studies discussed below, in addition to the PGCS and ATM artifacts, two NASA domain specific requirements documents were used. These documents were created in collaboration with experts from NASA to ensure that the domain was proper and the requirements were realistic. These systems focus more on computational problems involving a large amount of mathematics. Only the requirements documents from these systems were used in the studies.

4.2.3 Brief description of each study

Because data from each study was used to draw conclusions about more than one background or experience variable, a brief description of each of the studies is provided here. In the subsequent sections the study will only be referred to by name with no more details provided. More information about each study can be found in Appendix A.

4.2.3.1 NASA Studies (1994/1995)

Two studies were conducted at NASA's Goddard Space Flight Center in Greenbelt, MD using professional software developers as subjects. The goal of these studies was to compare the effectiveness of a PBR inspection method to that of the standard method used by NASA software practitioners. Subjects first inspected a generic (non-NASA related) and a domain specific (NASA related) requirements document using their normal procedure. Then they were trained in perspective-based procedure, and inspected a second generic and domain specific set of requirements. Quantitative data was collected via defect report forms. Qualitative data was collected via a post-study questionnaire [Basili96].

There were some small differences between the 1994 and the 1995 runs of this study. The main difference involved the artifacts inspected. Requirements documents from the PGCS, ATM, and NASA domains were used in both studies, but after the 1994 study, the researchers discovered that the artifacts were difficult to understand. So, for the 1995 study, the requirements documents were made more readable and easier to understand. Because of this fact, the data from 1994 and 1995 cannot be combined together into one data set.

The data was analyzed for each of the 4 documents separately, then by grouping together the generic documents (ATM, PG) into one group and the domain-specific documents (NASA) into another group. Finally, all four documents were analyzed together. Each of the above analyses was done separately for the 1994 replication and for the 1995 replication. When the results are reported below in Section 4.2.2, they will be reported corresponding to the document and technique used.

In all the other studies, the data was grouped such that those subjects who had industrial experience were in one group and the subjects who did not have industrial experience were in the other group. But, for these studies, all the subjects were working in industry, so all subjects would have fallen into the group that had industrial experience. Therefore, for the analysis of these studies, the subjects were grouped into low and high experience groups based on the number of years of experience they had. A natural break was found in the data, and this is where the cut off between high and low experience was placed.

4.2.3.2 CMSC735/MSWE609 University of Maryland (Fall 1997)

This study was conducted with graduate students in the Computer Science program as well as students from the Master's of Software Engineering program at the University of Maryland. The goal of this study was to compare the effectiveness of a checklist-based requirements inspection to a perspective-based requirements inspection in isolation from a real project. Subjects were instructed in the use of the checklist and then given a requirements document to inspect. After the completion of this inspection, the subjects were then trained in the perspective-based reading (PBR) techniques. They then inspected another requirements document using the PBR techniques. Quantitative data was collected via defect collection forms. Qualitative data was collected via post-study questionnaires [Lanubile98], [Shull98]. In this study the subjects were grouped for each variable based on whether or not they had industrial experience. The high experience group for each variable consisted of those subjects who had industrial experience while the low experience group consisted of the subjects who did not have industrial experience.

4.2.3.3 CMSC435 University of Maryland (Fall 1998)

This study was conducted with senior level undergraduate students enrolled in a software engineering project course at the University of Maryland. The goal of this study was to again compare the checklist-based requirements review with the PBR requirements review, this time in the context of a project. Subjects received training in either the PBR technique or the Checklist-based technique. The subjects then performed a requirements inspection on the project they were developing. Quantitative data was collected via defect report forms. Qualitative data was collected via post-study questionnaires. In this study the subjects were grouped for each variable based on whether or not they had industrial experience. The high experience group for each variable consisted of those subjects who had industrial experience while the low experience group consisted of the subjects who did not have industrial experience.

4.2.3.4 CMSC735 University of Maryland (Fall 1999)

This study was conducted with graduate students enrolled in a graduate-level software engineering course at the University of Maryland. The goal of this study was to understand both PBR and OORTs at a more detailed level than in the past. To accomplish this level of detail the subjects were paired together, and observed each other during the inspection process. Subjects were first trained in the PBR procedures, then one subject used PBR on the assigned set of requirements while his partner observed and took notes about the execution of the technique. After receiving training in the OORTs procedures, the partners switched roles, and the second inspection was done using the OORTs on an assigned design. Quantitative data as collected via defect report forms and background questionnaires completed by each subject. Qualitative data was obtained from reports written by each team based on their observations during the inspections [Shull99, Shull01]. In this study the subjects were grouped for each variable based on whether or not they had industrial experience. The high experience group for each variable consisted of those subjects who had industrial experience while the low experience group consisted of the subjects who did not have industrial experience.

4.2.3.5 University of Southern California (USC Fall 2000/Spring 2001)

This study was conducted with graduate students enrolled in software engineering courses at the University of Southern California under the direction of Dr. Barry Boehm. The goal here was to determine if the PBR and OORT techniques could be tailored for use in the spiral development model [Boehm88] and the MBASE document standard [Boehm99]. Subjects were trained in the techniques and how they interacted with the spiral model and the MBASE document standard. Subjects then used the techniques to aid in the inspection of the artifacts for the projects they were developing for real customers as part of the class. Quantitative data was collected via defect report forms. Qualitative data was collected via questionnaires as well as post-study interviews [Shull03]. In this study the subjects were grouped for each variable based on whether or not they had industrial experience. The high experience group for each variable consisted of those subjects who had industrial experience while the low experience group consisted of the subjects who did not have industrial experience.

4.2.3.6 Summary of data from studies

As another indication of the importance of understanding the individual variations among inspectors, a summary of the data will be presented here. For each of the studies that will be discussed in this section, Figure 6 reports the percentage of defects found by the most effective and least effective inspector for each artifact used. As the table shows, in most cases there is a large variation between the least effective inspector and the most effective inspector. This result again shows the importance of understanding the cause of these variations.

Study	Phase	Artifact	Lowest	Highest
NASA 1994	Req.	NASA A	5%	28%
		NASA B	0%	40%
		PGCS	6%	41%
		ATM	11%	57%
NASA 1995	Req.	NASA A	13%	86%
		NASA B	6%	93%
		PGCS	18%	68%
		ATM	7%	55%
CMSC735 1997	Req.	ATM	7%	77%
		PGCS	7%	75%
CMSC435 Fall 1998	Design	LA	0%	31%
CMSC 735 Fall 1999	Req.	LA	6%	33%
		PGCS	3%	34%
	Design	LA	16%	34%
		PGCS	5%	21%

Figure 6 – Variation in Effectiveness

4.2.4 Historical data support of proposed variables

Because the variables presented in Section 4.1.5 were all taken from a survey of the literature, and are at a relatively abstract level, in most cases they could not be directly measured. Therefore, in order to measure each of the variables, one or more metrics was defined such that those metrics could be collected in a study. Defining a set of metrics that accurately represent each variable is a difficult task.

The following subsections present the metrics defined for each variable defined by a series of historical studies. The data from the metrics in these studies is discussed and new hypotheses about the relationship of the metric and the inspection process are proposed for each of the variables. The hypotheses are characterized in the following way, along with a summary of the mapping to high and low experience (for all studies other than NASA, the mapping for the NASA studies was described in Section 4.2.3.1. The complete mapping can be found in Appendix C):

1) Each hypothesis is related to one of the variables defined above:
    a. DK = Domain Knowledge
        i. High experience = "very familiar"
        ii. Low experience = anything else
    b. SD = Software Development Experience:
        i. RE = Requirements Experience
        ii. DE = Design Experience
        iii. TE = Testing Experience
        iv. PE = Perspective Experience
        v. For each of these variables:
            1. High experience = "industrial experience"
            2. Low experience = "no industrial experience"
    c. PE = Inspection Process Experience
        i. High experience = "industrial experience"
        ii. Low experience = "no industrial experience"
    d. T = Training
    e. M = Motivation
2) Each variable's hypothesis are numbered 1-N
3) If the hypothesis refers to a requirements inspection specifically, the number will be followed by a ".Req". If it applies to a design inspection, the number will be followed by a ".Des".

For example, H.DK.1.Req is the first hypothesis about the Domain Knowledge variable and it applies to a requirements inspection. H.SD.RE.4.Des is the fourth hypothesis about the sub-variable of Software Development Experience, Requirements Experience, and it is applied to a design inspection.

In addition to this numbering scheme, I also present the hypotheses in a predicate calculus form. This representation will use the traditional predicate calculus operators ($\wedge$ = "And"; $\vee$ = "Or"; $\neg$ = "not"; $\Rightarrow$ = "implies"; $\neg\Rightarrow$ "does not imply"). For each hypothesis, in addition to its description in words, I will also show the hypothesis in this predicate calculus form. The predicate calculus representation will also allow the hypotheses to be more easily compared against each other and merged together later on. The merging and comparisons will be done in Chapter 7.

There are a few assumptions made throughout this work. The first is that each of the metrics listed below accurately measures the more abstract variable to some level of confidence. It is possible that one or more of the metrics does not really measure the background and experience variable. One potential hypothesis is that the metrics presented are not really related to the background and experience variable at all.

Secondly, one of the issues that makes this type of research difficult is that it is often hard to isolate the individual variables. In many cases a second or third variable provides a confounding factor that makes it difficult to clearly understand the variable being studied. In this section, cases where confounding factors could be present will be discussed. In cases where confounding factors appear to be present, new studies can be designed that attempt to isolate the variable of interest from the confounding factors.

Future study will need to be done to determine which of the variables and metrics depend on each other. Also, a further analysis of the data can be done to determine if one variable or metric masks possible results for another variable or metric.

For each of the variables below, data from the different studies that addressed the variable will be discussed. In many cases, there were not statistically significant results. Because the goal of this work is to generate a series of hypotheses rather than to test those hypotheses, statistical significance is not as important of an issue. The cases below where a hypothesis is supported by statistically significant results will be noted. For each variable, a summary of the data will be presented; the complete set of results can be found in a series of tables in Appendix B.

4.2.4.1 Application Domain Knowledge

The *Application Domain Knowledge* variable is concerned with the effect of an inspector's knowledge of the application domain on the inspection process. To collect this metric, subjects were asked to rate themselves either as experienced or not in the application domains in the study. This metric was collected in the CMSC735 University of Maryland (Fall 1999) study (see Table 7 for complete results).

The data show that for a requirements inspection, when looking at the tester perspective of PBR alone and the tester and user perspectives together, subjects with more application domain knowledge find more defects than those with less application domain knowledge. This result agrees with the literature that during a requirements inspection, the inspectors do not have another document to compare the requirements to in determining correctness as they would in a design or code inspection. Therefore, the more application domain knowledge an inspector has, the easier it should be for him or her to spot problems in the requirements document. One threat to the validity of this result is that the experts and the non-experts inspected different requirements documents. So, a confounding factor in this study is the requirements artifact inspected. Even so, there is enough indication to consider this variable further.

On the other hand, the data from the design inspections shows the opposite result. Subjects with less knowledge about the domain found more defects than those with more knowledge. This result also agrees with the literature, because inspectors with more application domain knowledge may make assumptions based on that knowledge and not report a defect where someone with less domain knowledge has the requirements document to use for comparison, so may report more defects. These results produce the following two hypotheses.

H.DK.1.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more domain knowledge ***will find more*** *defects than inspectors who have less domain knowledge.*

(Domain Knowledge) $\wedge$ (Requirements Inspection) $\Rightarrow$ (more defects found)

*H.DK.1.Des (Potential Hypothesis): For an OO Design inspection, inspectors who have more domain knowledge **will find fewer** defects than inspectors who have less domain knowledge.*

(Domain Knowledge) $\wedge$ (Design Inspection) $\Rightarrow$ $\neg$(more defects found)

4.2.4.2 Software Development Experience

Software Development Experience captures the relationship between experience in various aspects of software development and defect detection effectiveness. Because S*oftware Development Experience* is an abstract concept, this section will examine a set of metrics to begin to understand the important aspects of software development experience. The section is organized around the phase of the software development. For each lifecycle phase, a series of specific metrics will be discussed. For this discussion, it is assumed that more experience in any individual metric translates to more software development experience.

General Software Development Experience

This set of metrics deal with metrics that are non-lifecycle phase specific. Later sections will address metrics for each lifecycle phase in detail.

1. Software Development Experience

In some of the studies the metric, *Software Development Experience,* was collected as a metric, by asking the subjects whether they had industrial software development experience or not without indicating any specific type. Therefore each subject could have made different assumptions as to what constituted software development experience. Because this metric was not specific as to the types of software development experience that were important, there is a potential threat to validity of this result and it is thus is only reported as one piece of the software development experience variable. The remaining metrics in this section capture experience with more specific types of software development experience.

There were four studies that collected this metric, the CMSC 735 Fall 1999 study and the three studies conducted at USC (see Table 8 for compete results). The data from the CMSC 735 Fall 1999 study shows that when inspecting requirements, subjects with more experience found more defects than those with less experience. Additionally, the data from one requirements and one design study from USC also shows that subjects with industrial experience found more defects than those without industrial experience. On the other hand, the third USC study showed that subjects with less experience found more defects than those with more experience. Because the results from the other two studies were stronger, this third study seems to be an anomaly. Based on this data, the following hypotheses were generated.

*H.SD.1.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more software development experience **will find more** defects than inspectors who have less software development experience.*

(Software Development Experience) $\wedge$ (Requirements inspection) $\Rightarrow$ (more defects found)

H.SD.1.Des (Potential Hypothesis): For a design inspection, inspectors who have more software development experience ***will find more*** *defects than inspectors who have less software development experience.*

(Software Development Experience) $\wedge$ (Design inspection) $\Rightarrow$ (more defects found)

2. Experience as a Developer

This metric is a more specific instance of Software Development Experience. On the form used to collect this metric, subjects were asked for the number of years experience they had, but a specific definition of "experience" was not given, so presumably each subject came up with his or her own definition of experience. Therefore, while this metric is more specific than the previous one, there is still more uncertainty than some of the metrics discussed later.

In the NASA 1994 study (see Table 9 for complete results), the results show that for some documents (NASA B, ATM) inspectors with more experience as a developer found more defects and for others (NASA, PG) inspectors with less experience as a developer found more defects. These split results point towards a potential confounding factor, the artifact being reviewed. On the other hand, when the data from the documents is grouped together, the results show that subjects with more experience find more defects in most cases although the document effect still seems to be present and the difference between the two groups is small. Therefore, these results are not very strong, but lean towards showing that subjects with more software development experience often find more defects.

In the NASA 1995 study the results are more consistent overall and tend to show that subjects with more experience as a developer find more defects (see Table 10 for compete results). The cases where this result was not true were all for the subjects using PBR on the NASA documents. The results from this study show a much stronger indication of the importance of this variable than the results from the NASA 1994 study. Based on this data, the following hypothesis was generated.

> *H.SD.2.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience, as a developer* ***will find more*** *defects than inspectors who have less experience as a developer.*
>
> **(Experience as a Developer) $\wedge$ (Requirements inspection) $\Rightarrow$ (more defects found)**

Experience in Requirements Phase of Software Lifecycle

The metrics in this section cover experience in different aspects of working in the requirements phase of the software lifecycle. They include experience working with and writing requirements documents as well as experience working with and writing use cases.

1. General Requirements Experience

This metric covers overall requirements experience and was collected only in the CMSC 735 Fall 1997 study. In this study, all three PBR techniques were used. Because requirements experience was most closely related to the perspective of the user (who created use cases), the data was analyzed for the user perspective separately from the other two perspectives as well as in conjunction with them (complete results can be found in Table 11). The results show that subjects with more experience found more defects than those with less experience, for both the PBR techniques and the ad hoc method. These results make sense because the more experienced an inspector is in working with requirements, the better chance they will have of seeing the potential defects. Based on these results, the following hypothesis was generated.

> *H.SD.RE.1.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience working with requirements* ***will find more*** *defects than inspectors who have less experience working with requirements.*
>
> **(Experience working with requirements) $\wedge$ (Requirements inspection) $\Rightarrow$ (more defects found)**

2. Experience Using Requirements

From the NASA 1994 study (see Table 12 for complete results) the results do not provide a very strong indication one way or the other. When the data is analyzed for the individual requirements documents, the results are split. In some cases the subjects who have more experience using requirements find more defects, and in other cases the subjects who have less experience find more defects. But, when the documents are grouped together, there becomes a stronger indication that subjects with more experience using requirements find more defects.

From the NASA 1995 study (see Table 13 for complete results) the results are stronger. In all cases except for one of the NASA documents, the subjects with more experience found more defects than the subjects with less experience. Because almost all of the results point in the same direction there is a strong indication that this metric is important.

Because the results from the 1994 and 1995 studies differed, there was probably a confounding factor. The version of the artifacts used in 1994 was more complex than the version used in 1995. So, even though the artifact may be a confounding factor that is masking what is really going on, the data seems to point towards this variable as being important. Based on this data, the following hypothesis was generated.

H.SD.RE.2.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience using requirements ***will find more*** *defects than inspectors who have less experience using requirements.*

(Experience using requirements) $\wedge$ (Requirements inspection) $\Rightarrow$ (more defects found)

3. Experience Writing Requirements

There were four classroom studies where this metric was collected (see Table 14 for complete results). The results from these studies are not consistent. In the study run with undergraduates (CMSC 435 Fall 1998), the subjects with more experience found more defects than subjects with less experience with some of the results being statistically significant. On the other hand, in the three studies run with graduate students (CMSC 735 Fall 1999, and the two USC studies), the subjects with less experience found more defects than the subjects with more experience. In these three studies, all of the subjects used the PBR inspection technique. Because these results are contradictory, it is possible that the PBR technique was a confounding factor affecting the results and a conclusion cannot be drawn one way or the other as to the importance and relevance of this metric. Further study will be required.

There were also two industrial studies where this metric was collected (see Tables 15 and 16 for complete results). In the NASA 1994, subjects with more experience writing requirements found more defects than subjects with less experience in most cases. In this case some of the results were statistically significant. In all of the cases where less experienced subjects found more defects they were using the PBR technique. This result could again be caused by the PBR technique interfering with the experienced subjects' normal method of working and causing them to find fewer defects. On the other hand, the results from the NASA 1995 study are opposite. In most cases, the subjects with less experience writing requirements found more defects than the subjects with more experience.

Again there appears to be some confounding factors such as the inspection technique used. The only result that appears strong enough from which to draw any conclusions is the result for subjects using an ad hoc procedure.

H.SD.RE.3.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience writing requirements ***will find more*** *defects that inspectors who have less experience writing requirements, <u>especially when using an ad hoc technique</u>.*

(Experience writing requirements) $\wedge$ (Requirements inspection) $\wedge$ (Ad hoc technique) $\Rightarrow$ (more defects found)

4. General Use Case Experience

Because use cases are closely related to the requirements document and the requirements phase of the lifecycle, this metric potentially has an impact on a requirements inspection. Here subjects were asked if they had experience with use cases or not. No specifics about the type of experience were given, so each subject may have interpreted the question differently. Because of this uncertainty, there is a potential threat to the validity of this metric.

This metric was collected in the CMSC 735 Fall 1997 study (see Table 17 for complete results). The data from this study showed that in the cases where the subjects were using the PBR technique, those that had industrial experience with Use Cases found more defects than those without industrial experience. Based on this result, the following hypothesis was generated.

> *H.SD.RE.4.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience with use cases* ***will find more*** *defects than inspectors who have less experience working with use cases, <u>when using the PBR technique</u>.*
>
> **(Experience with UC) $\wedge$ (Requirements inspection) $\wedge$ (PBR technique) $\Rightarrow$ (more defects found)**

5. Experience Writing Use Cases

In collecting this metric subjects were asked if they had industrial experience writing use cases. Experience writing use cases was relevant for two reasons. First, the skill of writing use cases is closely related to the skill of working with requirements. Second, for the studies where PBR was used, one of the skills necessary to execute PBR is knowledge of writing use cases.

This metric was collected in three studies (see Table 18 for complete results). The results from the studies are split. From the CMSC 435 Fall 1998 study, when the subjects used a checklist, then those with more experience writing use cases found more defects than those with less experience. On the other hand, for the subjects using the PBR technique that involves use cases, the subjects with less experience found more defects. The results from the two USC studies where subjects used PBR are similar to each other. In the 2000 study the subjects from both groups found about the same number of defects and in the 2001 study the subjects with less experience found more defects. One potential confounding factor in this result is that the process and notation for creating the use cases in the PBR technique was non-standard. Therefore, those subjects who were more experienced with use cases might have gotten confused with the new notation and process and therefore not have been as able to focus on detecting defects. Based on these results, the following hypotheses were generated.

> *H.SD.RE.5.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience writing use cases* ***will find more*** *defects*

than inspectors who have less experience writing use cases, when using an Ad Hoc technique.

(Experience with UC) ∧ (Requirements inspection) ∧ (ad hoc technique) ⇒ (more defects found)

H.SD.RE.6.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience with a different use case technique and notation that the one used in the inspection ***will find fewer*** *defects than inspectors who have no experience in writing use cases, when using an inspection technique that involves creating Use Cases.*

(Experience with different UC technique or notation) ∧ (Requirements inspection) ∧ (inspection technique involving UC) ⇒ (more defects found)

Experience in the Design Phase of the Software Lifecycle

The metrics discussed in this section cover experience in different aspects in the design phase of the development lifecycle. The metrics range from general experience in software design, to experience in specific design paradigms (Structured and OO), and experience in a specific design notation (UML).

1. General Software Design Experience

This metric was collected in both studies on requirements inspections and studies on design inspections. The subjects were asked about the experience they had in software design, but no specific type of design methodology or paradigm was indicated. Because subjects might have design experience in a method or technique that is completely unrelated to the current study there is a threat to the validity of this result. The discussion below addresses the implications for requirements inspections and for design inspections (see Table 19 for complete results).

In the CMSC 735 Fall 1997 study dealing with a requirements inspection, the data shows that subjects with more experience find more defects than those with less experience. This result holds true both for subjects using an ad hoc inspection technique and for subjects using PBR. On the other hand, in the studies conducted at USC, the subjects classified themselves as “beginner”, “intermediate” or “advanced”. In this case the subjects who were “intermediate” found the most defects. Even though the two results are not directly comparable, in the USC studies the “beginner” group found the least amount of defects, lending some support to the idea that some level of experience is important.

The results of the CMSC 735 Fall 1999 study dealing with a design inspection were opposite. In all cases where the subjects used the OORTs, those with less experience found more defects than those with more experience. In the studies conducted at USC, subjects who were more experienced found more defects. A potential confounding factor for this metric is the design paradigm that the subject has experience in. It is possible that the first set of subjects did not have OO design experience, while the second group might have. A second confounding factor is the knowledge of the actual design being inspected. In the CMSC 735 studies, the subjects were not involved in the creation of the design whereas in the USC studies the subjects created the design that they were inspecting. Based on these results, the following hypotheses were generated.

> *H.SD.DE.1.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more software design experience* ***will find more*** *defects than inspectors who have less experience software design experience.*
>
> **(Software Design Experience) $\wedge$ (Requirements inspection) $\Rightarrow$ (more defects found)**

> *H.SD.DE.2.Des (Potential Hypothesis): For a design inspection, inspectors who have more experience in the design methodology or paradigm of the artifacts being inspected* ***will find more*** *defects than inspectors who have less experience in the design paradigm or methodology of the artifacts being inspected.*
>
> **(Experience in Design methodology) $\wedge$ (Design inspection) $\Rightarrow$ (more defects found)**

> *H.SD.DE.3.Des (Potential Hypothesis): For a design inspection, inspectors who have experience in an alternate design methodology* ***will find fewer*** *defects than inspectors who have no experience in design.*
>
> **$\neg$(Experience in Design methodology) $\wedge$ (Design inspection) $\Rightarrow$ $\neg$(more defects found)**

Experience in Design Based on Requirements

This metric investigated the subjects' experience in creating a design from a requirements document. Again, there was no specific design technique or methodology indicated in the data collection question. Because of this uncertainty, there is a potential threat to the validity of this result.

There were two studies that collected this metric, one for a requirements inspection, CMSC 435 Fall 1998, and one for a design inspection, CMSC 735 Fall 1999 (see Table 20 for complete results). For the requirements inspection, the inspectors with less experience found more defects than those with more experience. But the PBR technique that takes advantage of design knowledge was not used in this study.

In the study conducted on a design inspection there was one case where subjects with more experience in creating designs from a requirements document found more defects than those with less experience. As mentioned above, a potential confounding factor is that the design experience that the subjects had was not in the paradigm or methodology of the artifacts they were inspecting. Based on these results, the following hypotheses were generated.

> *H.SD.DE.4.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience in creating designs from requirements* ***will not find more*** *defects than inspectors who have less experience creating designs from requirements, when using a checklist or a non-design based PBR technique.*
>
> **(Experience creating Designs from requirements) ∧ (Requirements inspection) ∧ (checklist ∨ non-design PBR) ¬⇒ (more defects found)**

> *H.SD.DE.4.Des (Potential Hypothesis): For a design inspection, inspectors who have more experience in creating designs from requirements* ***will not find more*** *defects than inspectors who have less experience in creating designs from requirements, unless that experience is in a similar paradigm or technique as was used to create the artifacts being inspected.*
>
> **(Experience in creating designs from requirements) ∧ (Design inspection) ∧ ¬(Experience in similar paradigm as used to create artifact inspected) ¬⇒ (more defects found)**

2. Structured Design Experience

For this metric, the subjects were asked if they had experience creating structured designs. This type of experience is especially relevant to the requirements inspection when the inspector is using a technique such as PBR that heavily depends on design knowledge.

The results from the CMSC 735 Fall 1997 study, in which this metric was collected (see Table 21 for complete results), show that subjects with more experience found more defects than those with less experience in all cases except for when they used an ad hoc method. Additionally, when using the PBR technique based on design there is a bigger difference for experienced subjects than for the subjects using the other PBR techniques. Based on these results, the following hypothesis was generated.

> *H.SD.DE.5.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience in creating structured designs* ***will find more*** *defects than inspectors who have less experience in creating structured designs, especially when the inspectors are using an inspection technique that relies on design knowledge.*
>
> **(Experience creating design from requirements) ∧ (Requirements inspection) ∧ (inspection technique that relies on design knowledge) ⇒ (more defects found)**

3. OO Design Experience

This type of experience is useful when inspecting OO design artifacts. Also, inspectors working on requirements documents might find the knowledge of OO design useful in finding defects, especially if they are using an inspection technique focused around creating a design. Therefore, this metric was collected for a series of requirements inspection studies and design inspection studies (see Table 22 for complete results).

The results from the CMSC 735 Fall 1997 requirements inspection studies show that subjects with less experience found more defects than those with more experience. Also, in the two requirements studies run at USC the subjects classified themselves as "beginner", "intermediate", or "accomplished" in OO Design. Those studies show that in one case the subjects who were "accomplished" found the most defects and in the other case the "intermediate" group found the most defects with the "accomplished" group second. Because the requirements inspection technique that relied on design used a structured design approach, it is very possible that OO design experience did not help the subjects as much as structured design experience did (see previous metric), supporting the idea that for design experience to be useful it must be in the same paradigm or technique as the one currently being used.

The results from the studies conducted on OO design inspections also were mixed. In the CMSC Fall 1999 study, the subjects with less experience found more defects than the subjects with more experience. On the other hand, in the study run at USC, more experienced subjects found significantly more defects than less experienced subjects. A confounding factor in these two studies is that in the CMSC Fall 1999 study, subjects worked in pairs with one subject tasked to observe the other one performing the inspection, but in practice the inspections often deteriorated into two people working together to do the inspection. The subjects were classified based on the experience of the subject tasked to do the inspection, but because the subjects often worked together, this classification may not have been accurate. Based on these results, the following hypothesis was generated.

> *H.SD.DE.6.Des (Potential Hypothesis): For an OO design inspection, inspectors who have more OO design experience* ***will find more*** *defects than inspectors who have less OO design experience.*
>
> **(OO Design experience) $\wedge$ (Design inspection) $\Rightarrow$ (more defects found)**

4. UML Experience

Because the inspectors were inspecting OO designs, knowledge of the design notation should be helpful in performing the inspections. Subjects in the studies were asked if they had experience working with UML.

The data from the studies where this metric was collected (see Table 23 for complete results) show opposite results. The CMSC 735 Fall 1999 study shows that subjects with less experience found more defects than those with more experience. On the other hand, the study from USC shows that the subjects with more experience found more defects than those without industrial experience. A confounding factor here was the same as in the previous metric. Because the subjects were paired up in the CMSC 735 Fall 1999 study, and often worked together, the rating of UML experience may not be accurate. Therefore, the following hypothesis was generated based on the CMSC 735 Fall 1999 results.

> *H.SD.DE.7.Des (Potential Hypothesis): For a design inspection, inspectors who have more experience working with the design notation used in the artifacts being inspected* ***will find more*** *defects than inspectors who have less experience working with the design notation used in the artifacts being inspected.*
>
> **(Experience working with the design notation) ∧ (Design inspection) ⇒ (more defects found)**

Experience in the Testing Phase of the Software Lifecycle

The metrics in this section deal with different aspects of software testing experience. They range from general experience as a tester, to experience with specific testing paradigms, down to specific testing techniques.

1. General Testing Experience

When collecting this metric, specific types of testing experience were not asked for, therefore it is not clear if the experience that a subject had is relevant to the current inspection or not. But, general experience as a tester should make an inspector more aware of what to look for during a requirements inspection. This metric was collected in the NASA studies as well as in the studies at USC.

The results from the NASA 1994 study (see Table 24 for complete results) show that subjects with more experience as a tester found more defects than subjects with less experience when using an ad hoc technique. In fact, many of the results for the ad hoc technique were statistically significant. The opposite is true for subjects using the PBR technique; less experienced subjects found more defects. A confounding factor here is that the PBR procedures could interfere with the way an expert does their job and causing them to find fewer defects while at the same time helping a non-expert to find more defects than they normally would.

The results from the NASA 1995 study (see Table 25 for complete results) are slightly different but similar to the NASA 1994 results. In most, but not all, cases when they were using an ad hoc technique, the more experienced subjects found more defects than the less experienced subjects. But, unlike the previous study, when they were using the PBR technique, inspectors with more experience also found more defects than those with less experience. In fact, when the documents are grouped together, more experienced subjects find more defects than less experienced subjects in all cases.

Results from the USC study also support this metric as being important (see Table 26 for complete data). In both studies in which it was collected the subjects with more experience found more defects than those with less experience.

It is not clear why the results from the NASA 1994 study contradict those from the NASA 1995 and USC studies. It is possible that there was a confounding factor in the NASA 1994 study. As mentioned in the description of the studies in Section 4.2.2, the requirements documents that were inspected were simplified between the NASA 1994 and the NASA 1995 studies. So, in the NASA 1995 study, the requirements documents were more understandable than in the NASA 1994 study. Furthermore, in the USC studies, the inspectors had themselves written the requirements document that they were inspecting. Therefore those requirements documents can be assumed to be understandable to the inspectors. Based on the results of these studies, the following hypothesis was generated.

> *H.SD.TE.1.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience as a tester* ***will find more*** *defects than inspectors who have less experience as a tester, <u>especially when using an Ad Hoc technique, but also for PBR</u>.*
>
> **(Experience as tester) $\wedge$ (Requirements inspection) $\wedge$ (Ad hoc $\vee$ PBR) $\Rightarrow$ (more defects found)**

2. Functional Testing Experience

This metric should be a more closely related aspect of software development experience because functional testing is associated with the requirements document, and the subjects here were inspecting a requirements document.

The results from the CMSC 735 Fall 1997 study (see Table 27 for complete results) show that when they are using PBR, subjects with more functional testing experience find more defects than those with less experience. While none of the results were statistically significant, the one that was closest was for the subjects who were using the Tester perspective of PBR. This result makes sense, because those subjects using the Tester perspective would be using their knowledge of functional testing more than the subjects using the other perspectives of PBR. Based on these results, the following hypothesis was generated.

> *H.SD.TE.2.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more functional testing experience* ***will find more*** *defects than inspectors who have less functional testing experience, <u>when using PBR, especially the tester perspective</u>.*
>
> **(Functional testing experience) $\wedge$ (Requirements inspection) $\wedge$ (PBR, tester) $\Rightarrow$ (more defects found)**

3. Experience in Testing Based on Requirements

This metric is applicable here for the same reasons as the functional testing experience metric. Testing based on requirements is typically going to be done using a functional testing method.

The results from this metric, which was collected in the CMSC 435 Fall 1998 study, (see Table 28 for complete results) show that when they use a checklist method, inspectors with more experience testing find more defects than those with less, but when they use the User perspective of PBR the two groups find the same number of defects. It is not surprising that testing experience did not help inspectors who were using a PBR technique that did not require testing knowledge. Based on these results, the following hypotheses were generated.

H.SD.TE.3.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience testing based on requirements ***will find more*** *defects than inspectors who have less experience testing based on requirements, when using a non-procedural checklist based method.*

(Experience testing based on requirements) ∧ (Requirements inspection) ∧ (non-procedural checklist) ⇒ (more defects found)

H.SD.TE.4.Req (Potential Hypothesis) For a requirements inspection, inspectors who have more experience testing based on requirements will ***not find any more or fewer*** *defects than inspectors who have less experience testing based on requirements, when using the PBR User perspective.*

(Experience testing based on requirements) ∧ (Requirements inspection) ∧ (PBR, User) ¬⇒ (more defects found)

4. Experience in Equivalence Partition Testing

This specific testing technique was used by the subjects who used the tester perspective of PBR. The PBR theory is that if a subject is more experienced with a specific skill necessary to do the PBR technique, they will be more effective in finding defects.

Based on the results from the CMSC 735 Fall 1997 study, this metric is probably not very useful in predicting inspection performance (see Table 29 for complete results). For the subjects who used the tester perspective of PBR, those with more experience did not find as many defects as those with less experience. On the other hand, for the subjects using an ad hoc procedure and those using the other two PBR perspectives, subjects with more experience found more defects than those with less experience. Based on this result, the following hypothesis was generated.

H.SD.TE.5.Req (Potential Hypothesis): For a requirements inspection, the inspectors who have more experience in Equivalence Partition Testing ***will not find any more or fewer*** *defects than inspectors who have less experience in Equivalence Partition Testing.*

> **(Equivalence Partition Testing experience) ∧ (Requirements inspection) ¬⇒ (more defects found)**

Experience in the PBR Perspective

Because this metric is not specific to any of the lifecycle phases discussed above and does not span more than one lifecycle phase, it has been left in its own category. When using PBR, the subjects assume a particular perspective from which to inspect the requirements document. More perspective experience should allow inspectors to focus less on understanding the perspective and more on finding defects.

Two measures of perspective experience were collected in the CMSC 735 Fall 1999 study; a primary and a secondary (see Tables 30 and 31 for complete results). The results for each of these metrics shows that in all cases the inspectors who had more experience in their perspective found more defects than those who had less experience in their perspective. The primary metric was "testing experience" for those using the tester perspective and "experience writing use cases" for those using the user perspective. The secondary metric was "testing based on requirements" for those using the tester perspective and "experience reviewing use cases" for those using the user perspective. Based on these results, the following hypothesis was generated.

> *H.SD.PE.1.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience in their assigned PBR perspective* ***will find more defects*** *than inspectors who have less experience in their assigned PBR perspective, when using PBR.*
>
> **(Experience in PBR perspective) ∧ (Requirements inspection) ∧ (PBR) ⇒ (more defects found)**

When the *Software Development Experience* variable is examined overall, it appears that there is enough evidence to warrant further consideration as an important variable. When looking at the metrics that deal with general software development experience, there is a pretty strong indication from the data that they are important to continue studying. The metrics dealing with requirements experience show some strong results, but also have some contradictions among the studies. Requirements experience seems to be important, but more study will be required in order to more fully understand what is going on. The design experience metrics have enough support to continue studying this type of experience in more detail, but the results are not strong enough either way to draw a conclusion at this time. Finally, the testing experience seems to be important, but from the data so far we cannot yet determine the exact context where each specific type of testing experience will be useful. Overall, there is enough support from the data to consider *Software Development Experience* as an important variable and to continue studying it.

4.2.4.3 Inspection Process Experience

Like the software development experience variable, *Inspection Process Experience* is a large, abstract concept, difficult to measure directly. In order to better understand the impact of the process experience variable it must be broken down into a number of specific metrics.

Because the process studied here is software inspections, high process experience means high experience in performing inspections, not necessarily any specific method for performing inspections. In approximating process experience through this collection of specific metrics, an assumption is made that an accurate mapping of the individual metrics to levels of experience can be made. This assumption introduces a threat to the validity of the results. As each of these specific metrics is discussed below, the mapping of the values of the metric to either high or low process experience will be presented.

Comfort Reviewing Requirements

The first metric, *Comfort Reviewing Requirements,* assumes that the level of comfort with performing a process is related to the experience with that process. This metric is more concrete than *Process Experience* but it is still subjective. Each subject self-reports his or her comfort level, so there is no way to guarantee that each subject rates himself or herself based on the same criteria. For this metric, we will assume that a high comfort level will correspond to high experience and low comfort level to low experience.

From the 1994 NASA study (see Table 32 for compete data), the subjects who were more comfortable reviewing requirements found more defects in most cases than those who were less comfortable. In most of the cases where the subjects who were less comfortable reviewing requirements found more defects, they were using the PBR technique.

From the 1995 NASA study (see Table 33 for complete results), again the subjects who were more comfortable reviewing requirements usually found more defects than those that were less comfortable. As in the 1994 NASA study above, the cases where the subjects who were less comfortable reviewing requirements found more defects than those that were more comfortable were all cases where the subjects used the PBR technique.

In both of these studies, the cases where high *comfort reviewing requirements* did not translate to a higher defect detection rate were mostly subjects that used the PBR technique. The confounding factor here is that the PBR techniques were created to aid less experienced subjects in doing the inspection. Therefore for more experienced subjects the techniques may have interfered with their normal work practices causing them to perform poorer than normal while at the same time increasing the performance of the non-expert subjects. Based on these results the following hypotheses were generated.

H.PE.1.Req (Potential Hypothesis): For a requirements inspection, inspectors who are more comfortable with reviewing requirements ***will find more*** *defects than inspectors who are less comfortable reviewing requirements, <u>when using an ad hoc procedure</u>.*

(Comfort with reviewing requirements) $\wedge$ (Requirements inspection) $\wedge$ (ad hoc procedure) $\Rightarrow$ (more defects found)

H.PE.2.Req (Potential Hypothesis): For a requirements inspection, inspectors who are more comfortable reviewing requirements ***will not find any more or fewer*** *defects than inspectors who are less comfortable reviewing requirements, <u>when using PBR.</u>*

(Comfort reviewing requirements) ∧ (Requirements inspection) ∧ (PBR) ¬⇒ (more defects found)

H.PE.3.Req (Potential Hypothesis): For a requirements inspection, an inspector's comfort level with reviewing requirements may not be a real indication of his or her Process Experience; rather it may reflect some other type of knowledge or experience.

(Comfort with reviewing requirements) ¬⇒ (Process experience)

H.PE.4.Req (Potential Hypothesis): For a requirements inspection, PBR neutralizes the effect of experience.

Experience with reviewing requirements

This metric measured whether subjects had industrial experience reviewing requirements or not, making it less subjective than the previous metric because the subjects are not required to make an interpretation of what this means. So, it is assumed that subjects with industrial experience reviewing requirements have high process experience and that subjects with no industrial experience reviewing requirements have low process experience.

This metric was collected in a series of four different studies, CMSC 435 Fall 1998, CMSC 735 Fall 1999 and two USC studies (see Table 34 for complete results). The results show that subjects with more experience find more defects in most cases than the subjects with less experience. In most of the cases where the less experienced subjects found more defects they were inspecting the Parking Garage document, which was of a more familiar domain, or were inspecting requirements that they had created and previously inspected at USC. There seems to be a confounding factor of domain knowledge because the results indicate that when the subjects are not familiar with the domain of the requirements document (Loan Arranger and the Fall 2000 USC study (first semester of a two semester course), then those subjects who have more experience find more defects than those with less. Based on this result, the following hypotheses were generated.

*H.PE.5.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience reviewing requirements **will find more** defects than inspectors who have less experience reviewing requirements, when inspecting a requirements document which is either unfamiliar or from an unfamiliar domain.*

(Experience reviewing requirements) ∧ (Requirements inspection) ∧ (¬(document familiar) ∨ ¬(domain unfamiliar)) ⇒ (more defects found)

*H.PE.6.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience reviewing requirements **will not find any more or fewer** defects than an* *inspector who has less experience reviewing requirements, <u>when inspecting a requirements document from a familiar domain</u>.*

(Experience reviewing requirements) $\wedge$ (Requirements inspection) $\wedge$ (domain familiar) $\neg\Rightarrow$ (more defects found)

Experience with reading Object Oriented Designs

This metric deals with an objective measure of experience, this time in the realm of Object Oriented designs. Subjects were asked if they had industrial experience reading Object Oriented design documents or not. It was assumed that subjects with industrial experience had high process experience and subjects without industrial experience have low process experience.

The two studies that collected this metric (see Table 35 for complete results) have opposite results. In the 735 Fall 1999 study the subjects with more experience reading OO designs found fewer defects using the OORTs than subjects with less experience. On the other hand, in the study conducted at USC, the subjects with more experience reading OO designs found more defects than those with less experience. One potential confounding factor here is that the subjects from the 735 Fall 1999 study were overall less experienced than those from the USC study. Therefore the high and low experience subjects from the 735 Fall 1999 study were more similar to each other than the high and low experience subjects from the USC study and the lack of a positive correlation is not as surprising. Based on the results of the USC study, the following hypothesis was generated.

*H.PE.7.Des (Potential Hypothesis): For a design inspection, inspectors who have more experience reviewing object-oriented designs **will find more** defects than inspectors who have less experience reviewing object-oriented designs.*

(Experience reviewing OO Designs) $\wedge$ (Design inspection) $\Rightarrow$ (more defects found)

Software Inspection Experience

The metric *Software Inspection Experience* should be the most closely related to the process experience variable because the specific process being studied is a software inspection. But, because the specific type of inspection the subject had experience with was not collected, and there are many types of inspections, simply having experience in inspections may or may not translate to a higher process experience.

This metric was collected in three different studies, CMSC 735 Fall 1999 and two USC studies (see Table 36 for compete results). Inspectors who had less experience found more defects in most cases than those who had more experience. Two possible explanations for this situation exist. First, because the inspectors used a prescribed technique, either PBR or OORTs, in each of these studies, it is possible that the detailed techniques got in the way the high experienced subjects' normal technique and kept them from their optimal performance. Second, the inspection experience that the subjects had may have been in a different type or style of inspection process and did not translate to the inspection style used in the study. Based on these results, the following hypotheses were generated.

H.PE.8.Req (Potential Hypothesis): For a requirements inspection, for inspectors who are highly experienced in performing inspections, detailed techniques get in their way and prevent the optimal performance during the inspection.

(Experience performing inspections) ∧ (Requirements inspection) ∧ (Detailed techniques) ¬⇒ (more defects found)

H.PE.8.Des (Potential Hypothesis): For a design inspection, for inspectors who are highly experienced in performing inspections, detailed techniques get in their way and prevent the optimal performance during the inspection.

(Experience performing inspections) ∧ (Design inspection) ∧ (Detailed techniques) ¬⇒ (more defects found)

H.PE.9.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more software inspection experience ***will find*** *more defects than inspectors who have less software inspection experience, when that experience is with the same style of inspection currently being used.*

(Software Inspection experience) ∧ (Requirements inspection) ∧ (same style of inspection) ⇒ (more defects found)

H.PE.9.Des (Potential Hypothesis): For a design inspection, for Software Inspection experience to help an inspector, that experience must be with the same style of inspection currently being used.

(Software inspection experience) ∧ (Design inspection) ∧ (same style of inspection) ⇒ (more defects found)

Considering all of the metrics presented above, some general conclusions can be drawn. First, it becomes clear that *Process Experience* is difficult to measure accurately. Second, there are potentially other aspects of the *Process Experience* that have not been captured here. Important process experience metrics that could be missing deal with information such as the learning curve of a new process, and the amount and type of skill development necessary before a process is really useful. But, based on the metrics presented above, there is an indication that *Process Experience* is an important variable to consider further.

While there are not enough statistically significant results to draw definitive conclusions, there is enough support to warrant further study of this variable. Based on the results discussed above, it appears that there are some situations where process experience helps inspectors, and others where it does not help or even hurts. Based on these results, and using Grounded Theory, the hypothesis that *Process Experience* is important, was refined to include all of the additional hypotheses presented in this section. Because all the data discussed above came about as a byproduct of studying some other phenomena, future studies can be designed based on these new hypotheses to examine this variable in more detail. Additional data from these future studies will help understand *Process Experience* better and allow for continued refining of the hypotheses to determine its importance.

4.2.4.4 Training

Because the way that someone learns a new process or technique can have an impact on their performance of that process or technique, *Training* is an important issue to consider. Based on qualitative feedback from a number of different studies, it is clear that training is an important issue. Subjects often ask for more training and more examples of the process they are going to use. In order to examine the training variable in detail, a study would have needed to be conducted such that the subjects were split into two groups. Prior to performing the inspection process, each group would have received a different type of training. One potential way to study training would be to give both groups the same "training lecture", but then allow one group some extra time to practice the new technique and ask the researcher questions to make sure that he or she understands what they are to do. The results of these two groups could be compared to begin to understand what aspects of training are important and helpful for the inspectors.

H.T.1 (Potential Hypothesis): Inspectors who are properly trained ***will find more*** *defects than inspectors who are not properly trained.*

(Proper Training) ⇒ (more defects found)

H.T.2 (Potential Hypothesis): Inspectors who have laboratory time where they can practice the inspection technique ask questions about it ***will find more*** *defects than inspectors who do not have laboratory time, <u>when a new technique is used</u>.*

(Laboratory time during training) ∧ (new technique) ⇒ (more defects found)

4.2.4.5 Motivation

More so than some of the other variables mentioned in this section, *Motivation* is very difficult to quantify into a measurement. The main method of collecting metrics for this variable was through qualitative data collected after a study. Judging by the responses of the subjects, the researcher can begin to understand if the students were properly motivated or not. An example of this situation comes from a study that was initially run at the University of Maryland and then subsequently replicated at the Norwegian University of Science and Technology and at the University of Southern California. More details on these two replications can be found in [Shull03].

There were some major differences between these replications that shed some light on this *motivation* variable. In the original study, discussed in Section 4.2.1.4, which was conducted in a graduate level Software Engineering Class at the University of Maryland, one of the goals of the class was to learn about software processes and empirical methods used to evaluate those processes. As a practical example, students in the class used two inspection techniques (PBR and OORTs) on a set of artifacts that were given to them by the instructors. Their goal was to use the techniques to find defects, but, more importantly to evaluate the techniques based upon their experiences. The researchers worked with the instructor of the class to ensure that the study would fit well with the goals of the class and that the students would understand why they were asked to perform these inspections. The qualitative feedback from the subjects in the class showed that they were overall successful in using the techniques and providing good feedback to the researchers in their reports. Students did not complain that the assignment was unrelated to the class or that the time required was too onerous.

In a similar study run at the University of Southern California, the PBR and OORT techniques were tailored for the local setting. In this graduate level software engineering course, the subjects were working in teams to build a real project for a real customer. Students were required to meet with their customer to solicit requirements and then follow the Spiral lifecycle model over two semesters to complete the analysis, design, implementation, testing and delivery of the system. Over the two semesters the students were required to perform a series of inspections. After performing their first inspection in an ad hoc manner, the students were trained in the PBR and OORTs tailored for their environment. They were shown how the techniques fit with what they were already doing and the benefits they would gain by using the techniques. After performing their first inspection without the techniques, the students were able to see the potential benefits of having a prescribed technique to aid them in defect detection. The qualitative feedback from the subjects in this class showed that it was an overall positive experience. In fact, students were asked what the highlight of the class was, and many of them mentioned the reading techniques that they had learned and used.

On the other hand, in the replication at the Norwegian University of Science and Technology, the results were not so positive. The qualitative feedback from the subjects in this study indicated that they felt as though they were doing "slave labor", and the instructor was afraid that the class was going to "revolt" during the semester. There are a few reasons for this generally negative reaction. These reasons all have to deal with *Motivation* or lack thereof. First, the class in which the study was run was not class in which software development was done, nor was it a class where students were learning about empirical evaluation of software processes. The researcher in this case never presented the students with an adequate explanation of how the study fit with the rest of the goals of the class. The comments from the subjects after the study indicated this fact. Additionally, the students were given an incorrect and unrealistically short time estimate for completing the assignment. When the assignment took three or four times longer than the students were told, they got frustrated. As opposed to the very positive feedback received from the students at USC, the students in this class indicated that the inspection study was one of the worst things about their class.

The previous discussion of three studies does not define a quantitative metric for measuring *Motivation*, nor does it provide any statistical result to show its importance. But, the qualitative results from the two replications show that properly motivated subjects, especially in a classroom environment, have a much better experience when performing a process. These results are enough of an indication to show that *Motivation* should be studied further. This further study will require the determination of how to quantify motivation into a metric, and then performing a quantitative evaluation to complement the qualitative results discussed here.

H.M.1 (Potential Hypothesis): Properly motivated inspectors will find more defects than inspectors who are not properly motivated.

(Proper motivation) ⇒ (more defects found)

4.2.5 New variables that appeared from the data

While analyzing the data from the studies described in Section 4.2.2, not only were hypotheses formed about the variables identified in Section 4.1, but also some new variables were found. Based on the results of these studies, these new variables seemed to have an important effect on the outcome of the study. Therefore, it was decided that two new variables, *Working Language Experience (WL)* and *Level of Process Specificity (PS)*, should be added to the list originally defined in Section 4.1.4. Associated with those two variables is a set of hypotheses similar to those proposed in the previous section. The same numbering scheme that was used for the previous hypotheses will be used here, with hypotheses about working language experience being denoted with "WL" and hypotheses about the level of process specificity being denoted by "PS".

4.2.5.1 Working Language Experience

In some of the studies discussed in Section 4.2.2, another class of metrics was collected that were unrelated to any of the previous variables. In the CMSC 735 Fall 1999 and the USC studies, we collected background information on the subjects' experience with the working language as well as their competency level in two aspects of English, reading comprehension and listening/speaking skills. After analyzing this data, which is discussed in this section, it became apparent that *Working Language Experience* was an important variable to consider. Therefore a new set of hypotheses was defined, adding to the original list from Section 4.1.4.

> *H.WL – Inspectors who are more proficient and comfortable in various aspects of the working language* ***will find more*** *defects than those that are less proficient or comfortable.*

Native Language

In the first metric, each subject was classified as either a native English speaker or a non-native English speaker. Because the experimental instructions, forms, and artifacts were all in English, this distinction was an important one.

The results from the studies where this metric was collected (see Table 37 for complete results) show that in most cases the subjects who spoke English as their native language found more defects than those who did not speak English as their native langue. The cases from the CMSC 735 Fall 1999 study where the subjects with native English speakers did not find more defects were both dealing with the parking garage domain. It is possible that the Loan Arranger domain (the one where native English speakers found more defects) was more difficult to understand and therefore showed greater difference in familiarity with the English language. In the studies from USC, the two requirements studies showed that non-native speakers found more defects, while in the design study, native English speakers found more defects. A potential confounding factor here is application domain knowledge. It can be argued that because the subjects in these studies elicited the requirements and created the requirements documents, they would be familiar with the requirements and the application domain. Therefore, these results are similar to those subjects from CMCS 735 Fall 1999 who were inspecting the PGCS document. Based on these results, the following hypotheses were generated.

H.WL.1.Req (Potential Hypothesis): For a requirements inspection, inspectors who are native speakers of the working language ***will find more*** *defects than inspectors who are not native speakers, <u>especially in the case where the application domain is not familiar</u>.*

(Native seakers of working language) $\wedge$ (Requirements inspection) $\wedge$ $\neg$(familiar domain) $\Rightarrow$ (more defects found)

H.WL.1.Des (Potential Hypothesis): For a design inspection, inspectors who are native speakers of the working language ***will find more*** *defects than inspectors who are not native speakers, <u>especially in the case where the application domain is not familiar</u>.*

(Native speaker of working language) $\wedge$ (Design inspection) $\wedge$ $\neg$(familiar domain) $\Rightarrow$ (more defects found)

Reading Comprehension

The second metric only examines those subjects who were non-native speakers. This metric compares the subjects who reported a lower comfort in English reading comprehension to the subjects who reported a higher comfort in English reading comprehension. The value of this metric was only based on a self-characterization by the subjects.

The results of this metric (see Table 38 for complete results) show that in most cases the non-native English speakers that were more comfortable with English reading comprehension found more defects than those that were less comfortable. Unlike the previous metric, in this case the times where the subjects who were less comfortable found more defects were when they were inspecting the Loan Arranger document. The USC data does not show significant difference between the two groups. The two requirements studies show opposite results, while the design study shows that the subjects with less comfort reviewing requirements found more defects. Based on this data, the following hypotheses were generated.

H.WL.2.Req (Potential Hypothesis): For a requirements inspection, inspectors who have a higher comfort level in reading comprehension of the working language ***will find more*** *defects than inspectors who have a lower comfort level in reading comprehension of the working language<u>, for inspectors who are not native speakers of the working language</u>.*

(Comfort in reading comprehension) $\wedge$ $\neg$(Native speaker of working language) $\wedge$ (Requirements inspection) $\Rightarrow$ (more defects found)

H.WL.2.Des (Potential Hypothesis): For a design inspection, inspectors who have a higher comfort level in reading comprehension of the working language ***will find fewer*** *defects than inspectors who have a lower comfort level in reading comprehension of the working language, <u>for inspectors who are not native speakers of the working language</u>.*

(Comfort in reading comprehension) ∧ ¬(Native speaker of working language) ∧ (Design inspection) ⇒ (more defects found)

Listening and Speaking Skills

Like the previous metric, this metric was only measured for the subjects who were not native English speakers. The reason that *Listening and Speaking Skills* can be an important metric is that an instructor who was speaking English conducted the training. Therefore, those subjects who were less comfortable in listening to English may not have understood the procedures as well as those who were more comfortable. This metric was also a self-reported level of comfort with listening to and speaking English.

The results of this metric (see Table 39 for complete results) show that in most cases, those subjects who were more comfortable listening and speaking English found more defects than those who were less comfortable. The cases where the less comfortable subjects found more defects were during a design inspection. Because the designs being inspected were artifacts consisting of diagrams, it is possible that the explanations from the instructors were not as important as they were for the requirements inspection where subjects were dealing with textual documents. Based on theses results the following hypotheses were generated.

H.WL.3.Req (Potential Hypothesis): For a requirements inspection, inspectors who have a higher comfort level in listening to and speaking the working language will ***find more*** *defects than inspectors who have a lower comfort level in listening and speaking the working language, <u>for inspectors who are not native speakers of the working language</u>.*

(Comfort listening and speaking) ∧ ¬(Native speaker of working language) ∧ (Requirements inspection) ⇒ (more defects found)

H.WL.3.Des (Potential Hypothesis): For a design inspection, inspectors who have a higher comfort level in listening to and speaking the working language will **find fewer** defects than inspectors who have a lower comfort level in listening and speaking the working language, <u>for inspectors who are not native speakers of the working language</u>.

(Comfort listening and speaking) ∧ ¬(Native speaker of working language) ∧ (Design inspection) ⇒ ¬(more defects found)

The three metrics discussed for the *Working Language Experience* variable give an indication that this variable is important and has some effect on the inspection process. The results are not strong enough in either direction to draw concrete conclusions, but there are enough positive results to support further study of this variable.

4.2.5.2 Level of Process Specificity

Another variable that was discovered during the review of the previous studies was that of *Level of Process Specificity*. In his dissertation, Shull compared the results of the CMSC Fall 1997 study to the results of the NASA study. The main difference between those two studies was the level of specificity in the PBR process. In the NASA study, the PBR techniques were very high-level and provided the subjects little detailed guidance on performing the inspection. On the other hand, the PBR techniques used in the CMSC Fall 1997 study were more detailed, giving the subjects a step-by-step procedure to follow while doing the inspection. In comparing the two studies, Shull was trying to understand how the additional details in the PBR technique affected the number of defects that the subjects found. In this analysis, the subjects were classified as having high experience, medium experience, or low experience in the modeling technique in their assigned perspective. The results showed that for the subjects with medium experience, the detailed technique helped them find more defects. On the other hand, for the high experience and low experience subjects, the detailed technique did not help the subjects improve, either over the less detailed PBR or over their own ad hoc procedures [Shull98].

This result identifies that there is a link between an aspect of software development experience and the process that should be used for the inspection. While the previous results are specifically in terms of an inspector's experience with the model used in their PBR technique, I argue that various other aspects of software development experience can have an impact on the type of process that an inspector needs to use. In this case, we will focus on the level of specificity in the process and what we can understand about its interaction with an inspector's previous experience.

> *H.PS The level of specificity necessary in an inspection process is related to the level of experience the inspector has.*

4.2.6 Expanded list of variables and metrics

In using a Grounded Theory approach, I began by defining a series of variables based on readings from the literature. This section used historical data to make the first grounded theory iteration to refine each of these variables. In this iteration I have defined a series of metrics to quantify each variable as well as a series of hypotheses that need to be tested concerning each variable. In addition, two new variables were discovered. Figure 7 is an updated version of Figure 5 that includes the two new variables and how they relate to the inspection process and inspection artifact.

As a summary, the updated list of variables and their definitions appear below. For each variable there is a list of the studies in which it was collected. In doing the analysis, all of the subjects were grouped into "high experience" and "low experience" for the various metrics listed below. Appendix C contains the questions used to collect each metric and how the grouping of "high experience" and "low experience" was done.

1) ***Domain Knowledge –*** This variable deals with the inspectors' knowledge, experience, and familiarity with the problem domain of the software system being inspected.
    - i. CMSC 735 Fall 1999
2) ***Software Development Experience*** – This variable deals with the level and number of years experience the inspectors have in different aspects of software development, ranging from general experience, to experiences in specific lifecycle phase activities.
    - a. General Software Development Experience

        i. Experience as a Developer (NASA 1994, 1995)
        ii. Software Development Experience (CMSC 735 Fall 1999, USC)
    b. Experience in the Requirements Phase of the Software Lifecycle
        i. General Requirements Experience (CMSC 735 Fall 1997)
        ii. Experience Using Requirements (NASA 1994, 1995)
        iii. Experience Writing Requirements (NASA 1994, 1995, CMSC 435 Fall 1998, CMSC 735 Fall 1999, USC)
        iv. General Use Case Experience (CMSC 735 Fall 1997)
        v. Experience Writing User Cases (CMSC 435 Fall 1998, USC)
    c. Experience in the Design Phase of the Software Lifecycle
        i. General Software Design Experience (CMSC 735 Fall 1997, CMSC 735 Fall 1999, USC)
        ii. Experience in Design Based on Requirements (CMSC 435 Fall 1998, CMSC 735 Fall 1999)
        iii. Structured Design Experience (CMSC 735 Fall 1997)
        iv. OO Design Experience (CMSC 735 Fall 1997, CMSC 735 Fall 1999, USC)
        v. UML Experience (CMSC 735 Fall 1999, USC)
    d. Experience in the Testing Phase of the Software Lifecycle
        i. General Testing Experience (NASA 1994, 1995, USC)
        ii. Functional Testing Experience (CMSC 735 Fall 1997)
        iii. Experience in Testing Based on Requirements (CMSC 435 Fall 1998)
        iv. Experience in Equivalence Partition Testing (CMSC 735 Fall 1997, USC)
        v. Experience in the PBR Perspective (CMSC 735 Fall 1999)

3) ***Inspection Process Experience*** – This variable examines the experience that the inspectors have in performing the process of inspections.
    a. Comfort Reviewing Requirements (NASA 1994, 1995)
    b. Experience with Reviewing Requirements (CMSC 435 Fall 1998, CMSC 735 Fall 1999, USC)
    c. Experience with Reading Object Oriented Designs (CMSC 735 Fall 1999, USC)
    d. Software Inspection Experience (CMSC 735 Fall 1999, USC)

4) ***Training*** – This variable deals with the amount and type of training that the inspectors receive prior to performing an inspection.
    a. Presence of "laboratory" time to try out new procedure before using
        i. Not yet specifically measured in any studies

5) ***Motivation*** – This variable deals with the desire of the inspectors to learn a new inspection process and to perform that process well.
    a. Goals and Relevance of study understood (No studies)
    b. Accurate time estimates (No studies)

6) ***Working Language Experience –*** This variable deals with the various types of relevant experience that inspectors have with the language of the artifacts and inspection procedures.
    a. Native Language (CMSC 735 Fall 1999, USC)
    b. Reading Comprehension (CMSC 735 Fall 1999, USC)
    c. Listening and Speaking Skills (CMSC 735 Fall 1999, USC)

7) ***Level of Process Specificity* –** This variable deals with the amount of detail that is required in a technique depending on the experience of the inspector who is using that technique.
    a. Relationship of perspective experience and process detail (CMSC 735 Fall 1997)

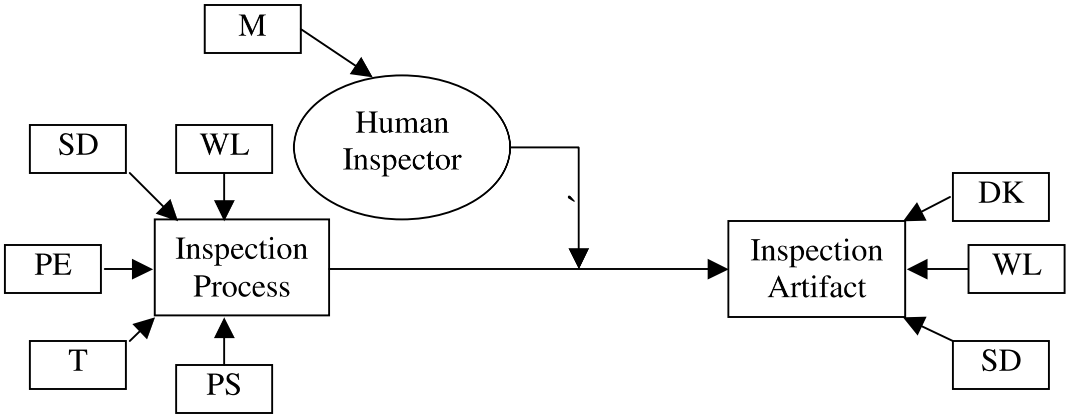


Figure 7 – Experience Variables

5. Refine hypotheses based on data from new studies

Starting with the refined list of variables, metrics and hypotheses, established by using data from previous studies, the grounded theory approach allows for continued refining of the list of variables, metrics and hypotheses. Two methods were used for this continued refining. This chapter presents the first method, which is to use results from a new set of studies, not used to establish the initial hypotheses in Section 4.2. If the data from the new studies supports the hypotheses presented in Section 4.2, then they become stronger. If the data from the new studies contradicts the hypotheses from Section 4.2, then the variables, metrics and hypotheses must be refined based on the new results. The second method, presented in Chapter 6, is that a series of new studies can be designed to test specific hypotheses. For each of these methods, as the results are discussed, any necessary changes to the hypotheses will be made.

As the data was analyzed, and refinements were being made to the hypotheses, a number of constraints were identified, both in the original hypotheses as well as in the refined hypotheses. The nature of those constraints can be characterized as follows. First, there are constraints on the technique type. These constraints somehow limit the applicability of the hypotheses to a specific subset of techniques. The second constraint type were those that applied to characteristics of the inspector. These constraints limit the pool of inspectors based on some characteristic. Finally, the third constraint type are those that deal with the artifacts being inspected. These constraints limit the applicability of the hypothesis to specific classes of artifacts. The new studies allow us to either add one of these constraints or remove one of these constraints from a hypothesis. Constraints can be removed because some of the hypotheses in Section 4.2.3 were initially defined with constraints.

To summarize, the potential changes to the hypotheses can be one of the following:

1) *Add a constraint* - This type of change restricts the hypothesis based on:
    a. The technique type
    b. The characteristics of the inspector
    c. The artifact being inspected
2) *Remove a constraint* - This type of change allows the hypothesis to apply to:
    a. More technique or sub-technique types
    b. More inspectors
    c. More artifacts

When one of these changes is made, the hypothesis number will be updated to reflect the change. If a constraint is added on the technique type, the previous hypothesis number will be appended with a “.ACT”. If the constraint is added about the inspector, the number will be appended with a ”.ACI”. If the constraint is added about the artifact, the number will be appended with a “.ACA”. If the constraint is removed on a technique, the hypothesis number will be appended with a “.RCT”. If the constraint is removed on the inspector, the hypothesis number will be appended with a “.RCI”. If the constraint is removed on the artifact, the hypothesis number will be appended with a “.RCA”. This numbering scheme allows the history of the evolution of the hypothesis to be captured.

5.1 Brazilian replications of the NASA experiments to verify previous results

Dr. Jose Maldonado, in Brazil, replicated the original NASA 1994 study at the University of Sao Paulo, Brazil. This replication was done in conjunction with researchers at the University of Maryland, including myself. Dr. Maldonado agreed to share his data and results. Dr. Maldonado conducted four small replications (R1, R2, R3, and R4). The first two (R1 and R2) were run with relatively inexperienced university students while the second two (R3 and R4) were run with more experienced graduate students and industrial developers. The results of all four replications were used to continue to refine the list of variables, metrics, and hypotheses from Section 4.2

A few comments should be made concerning these replications before the data is discussed:

1) The data from these replications were not compared with the data from the original NASA study at this time to draw conclusions. Because of the differences in the setting, a straight comparison of the two data sets would not be reasonable.
2) The subjects in these replications are all Brazilians who speak Portuguese as their first language rather than English. The experimental materials including the requirements document to be inspected were all written in English. This language difference presents a potential threat to the validity of these results. Because of this fact, if the data from these replications contradicts results we have seen from the previous section, hypotheses may be modified, but they will not be removed. If these types of cases present themselves, they will be discussed at that time.
3) Because the subjects were all undergraduates in the first two replications, there was not a very large group that has industrial experience; therefore we do not have results for all of the variables and metrics.
4) The checklist that was used in these replications was slightly different from the one used in the original study. The checklist used here was organized around the specific defect types and was more procedural.
5) Although all three PBR perspectives were used in this study, because there were not enough subjects using each perspective to analyze them separately, analysis was only done on PBR as a whole and not on the individual perspectives.
6) Subjects from R1 and R2 used PBR techniques that were similar to those used in the NASA 1994/1995 studies. Those techniques did not include a lot of detail to direct the inspectors. Subjects from R3 and R4 used PBR techniques similar to those used in the CMSC Fall 1997 and CMSC Fall 1999 studies. Those techniques provided the inspectors with very detailed instructions on how to perform the inspection.

Many of the metrics discussed in Section 4.2 were collected during these four studies. The goal of analyzing this data set was not to test any specific hypotheses. Rather this data allowed the grounded theory approach to continue by either providing support for the existing hypotheses, or by refining those hypotheses to account for this data. Below, each one of the metrics that was collected will be discussed along with its impact on the previously established hypotheses. Complete results for all the variables can be found in Appendices D, E, F, and G.

5.1.1 Domain Knowledge

This metric was collected for only R2 (see Table 50), R3 (see Table 63) and R4 (see Table 77). The spread of the subjects between high and low domain knowledge was not even in these studies. In R2 very few subjects had high knowledge of the PGCS but most had high knowledge of the ATM. In R3, very few subjects had high knowledge about either domain. In R4, the subjects were spread out more evenly. In all three studies, the subjects using checklist who had more application domain knowledge found more defects. For subjects using PBR the results were opposite. In all but one case, the subjects who had less application domain knowledge found more defects than those with more application domain knowledge. These results make sense because inspectors who used a non-procedural technique like a checklist will be forced to rely more on the knowledge they already have than those inspectors who used a procedural technique that helps to guide them in the types of defects to look for.

As a result of this data, I was able to *add a constraint about the technique* to H.DK.1.Req, from Section 4.2. The original hypothesis was:

> *H.DK.1.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more domain knowledge* ***will find more*** *defects than inspectors who have less domain knowledge.*
>
> **(Domain Knowledge) $\wedge$ (Requirements inspection) $\Rightarrow$ (more defects found)**

The new hypothesis, H.DK.1.Req.ACT, with the added constraint (underlined in non-italic bold) is:

> *H.DK.1.Req.ACT (Potential Hypothesis): For a requirements inspection, inspectors who have more domain knowledge* ***will find more*** *defects than inspectors who have less domain knowledge,* <u>**especially when using a non-procedural or ad hoc technique**</u>**.**
>
> **(Domain Knowledge) $\wedge$ (Requirements inspection) $\wedge$ (non-procedural technique $\vee$ ad hoc technique) $\Rightarrow$ (more defects found)**

5.1.2 Software Development Experience

General Software Development Experience

Because these studies only involved the inspection of a requirements document, only the metrics relevant to a requirements inspection were collected. The metric "experience as a developer" was collected in all four studies. In R1 (see Table 41) and R2 (see Table 46) it was collected as number of years as a developer, and in R3 (see Table 53) and R4 (see Table 67) it was collected as "industrial experience" vs. "non-industrial experience". The data from the four studies overall shows a strong indication that subjects with more experience as a developer find more defects than those with less experience. The results in R2 are especially strong, with two of the aggregated results being statistically significant. Because this data supports the earlier hypothesis, no change is necessary, but now the hypothesis has more support.

H.SD.2.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience, as a developer ***will find more*** *defects than inspectors who have less experience as a developer.*

(Experience as a Developer) $\wedge$ (Requirements inspection) $\Rightarrow$ (more defects found)

Experience in Requirements Phase of Software Lifecycle

1) **Experience Writing Requirements**

This metric was collected all four studies, but in R1 and R2 there was only one subject who had high experience. Therefore, the results will only be reported for R3 (see Table 54) and R4 (see Table 67). The results from these two studies partially support and reinforce the previous hypothesis. The previous hypothesis stated that Experience Writing Requirements was especially important for inspectors who were using an ad hoc or non-procedural technique. In these two studies, the results are not as clear. For subjects using a checklist, there were more cases where less experienced subjects found more defects than those with more experience. On the other hand, for subjects using PBR, there were more cases where more experienced subjects found more defects than less experienced subjects.

As a result of this data, the previous hypothesis was broadened by *removing a constraint on the technique to* include not only inspectors using an ad-hoc or non-procedural methods, but to include inspectors using all techniques. The original hypothesis, H.SD.RE.3.Req, with the constraint to be removed underlined in non-italic bold, was:

H.SD.RE.3.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience writing requirements ***will find more*** *defects than inspectors who have less experience writing requirements,* **<u>especially when using a non-procedural or an ad hoc technique</u>**.

(Experience writing Requirements) $\wedge$ (Requirements Inspection) $\wedge$ (non-procedural $\vee$ ad hoc technique) $\Rightarrow$ (more defects found)

The new hypothesis, H.SD.RE.3.Req.RCT, with the removed constraint is:

H.SD.RE.3.Req.RCT (Potential Hypothesis): For a requirements inspection, inspectors who have more experience writing requirements will find more defects than inspectors who have less experience writing requirements.

(Experience writing Requirements) $\wedge$ (Requirements Inspection) $\Rightarrow$ (more defects found)

2) **Experience Writing Use Cases**

This metric was only collected for R3 (see Table 55) and R4 (see Table 68). The results from these two studies show that subjects with less experience writing use cases found more defects than those with more experience, in most cases. Hypothesis H.SD.RE.5 stated that subjects with more experience who were using an ad-hoc or non-procedural technique would find more defects than those with less experience. The results here do not support the hypothesis, but the checklist that was used in these studies was more procedural and better defined than the ad hoc procedure used in the previous studies. As a result of this data, I was able to *add a constraint on the technique* to the original hypothesis. The original version of H.SD.RE.5.Req was:

H.SD.RE.5.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience writing use cases ***will find more*** *defects than inspectors who have less experience writing use cases, when using an Ad Hoc technique.*

(Experience writing Use Cases) ∧ (Requirements inspection) ∧ (ad hoc technique) ⇒ (more defects found)

The new hypothesis, H.SD.RE.5.REQ.ACT, with the added constraint (underlined in non-italic bold) is:

H.SDE.RE.5.Req.ACT (Potential Hypothesis): For a requirements inspection, inspectors who have more experience writing use cases ***will find more*** *defects than inspectors who have less experience writing use cases, when using an Ad Hoc technique* **which is not a well-defined, procedural checklist**.

(Experience writing Use Cases) ∧ (Requirements inspection) ∧ (ad hoc technique) ∧ ¬(well-defined, procedural checklist) ⇒ (more defects found)

Experience in the Design Phase of the Software Lifecycle

1) General Software Design Experience

This metric was only collected in R3 (see Table 56) and R4 (see Table 69). The results from these two studies show that, in most cases, inspectors with more software design experience find more defects than inspectors with less software design experience. In the cases where the inspectors with less experience found more defects the subjects who were all using a checklist based procedure. H.SD.DE.1.Req stated that inspectors with more software design experience will find more defects than those with less experience. This checklist was different from the ad hoc techniques used to develop the original hypotheses. Because of this result, I was able to *add a constraint on the technique* to the original hypothesis. The original hypothesis, H.SD.DE.1.Req, was:

H.SD.DE.1.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more software design experience ***will find more*** *defects than inspectors who have less experience software design experience.*

(Software Design Experience) ∧ (Requirements inspection) ⇒ (more defects found)

The new hypothesis, H.SD.DE.1.Req.ACT, with the added constraint (underlined in non-italic bold) is:

H.SD.DE.1.Req.ACT (Potential Hypothesis): For a requirements inspection, inspectors who have more software design experience ***will find more*** *defects than inspectors who have less experience software design experience,* **except when using a well-defined, procedural checklist**.

(Software Design experience) ∧ (Requirements inspection) ∧ ¬(well-defined, procedural checklist) ⇒ (more defects found)

2) **Experience in Design based on Requirements**

This metric was also only collected in R3 (see Table 57) and R4 (see Table 70). The results from these two studies show that in most cases inspectors with more experience creating designs based on requirements found more defects than inspectors with less experience. For R4, this result was true in all cases. For R3, this result was true except for inspectors using a checklist on the PGCS and PBR inspectors on the ATM. The original hypothesis, H.SD.DE.4.a, stated that having design experience with creating designs based on requirements would not help inspectors using a checklist or non-design based PBR technique. Unlike the data used to develop that hypothesis, which was from relatively inexperienced undergraduate students, these studies were done with more experienced graduate students and industry professionals. The results here showed that a relationship between experience in design based on requirements and the number of defects found during a requirements inspection. Because these results were from more experienced subjects, the original hypothesis was refined by *removing a constraint on the technique.* The original hypothesis, H.SD.DE.4.Req, with the constraint to be removed underlined in non-italic bold was:

H.SD.DE.4.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience in creating designs from requirements ***will not find more*** *defects than inspectors who have less experience creating designs from requirements,* **when using a checklist or a non-design based PBR technique**.

(Experience creating designs from requirements) ∧ (Requirements inspection) ∧ (checklist ∨ non-design based PBR) ¬⇒ (more defects found)

The new hypothesis, H.SD.DE.4.Req.RCT, with the removed constraint is:

H.SD.DE.4.Req.RCT (Potential Hypothesis): For a requirements inspection, inspectors who have more experience in creating designs from requirements ***will not find more*** *defects than inspectors who have less experience creating designs from requirements.*

(Experience creating designs from requirements) ∧ (Requirements inspection) ¬=> (more defects found)

Experience in the Testing Phase of the Software Lifecycle

1) **Experience as a Tester**

This metric was collected in all four studies, but in R1 and R2 there was only one highly experienced subject, so the results will only be reported for R3 (see Table 58) and R4 (see Table 71). The results from these two studies show that in most cases inspectors with more experience as a tester found more defects than those with less experience. These results support hypotheses H.SD.TE.1. For R3, the cases where this result was not true was for subjects using the checklist on PGCS alone and both documents analyzed together. For R4, the only case where the result was not true was for subjects using PBR on PGCS. Therefore I was able to *remove a constraint on the technique* from the original hypothesis, H.SD.TE.1.Req. The original hypothesis was:

H.SD.TE.1.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience as a tester ***will find more*** *defects than inspectors who have less experience as a tester, especially when using an Ad Hoc technique, but also for PBR.*

(Experience as a tester) ∧ (Requirements inspection) ∧ (Ad hoc ∨ PBR technique) ⇒ (more defects found)

The refined hypothesis, H.SD.TE.1.Req.RCT, with the removed constraint (underlined in non-italic bold) is:

H.SD.TE.1.Req.RCT (Potential Hypothesis): For a requirements inspection, inspectors who have more experience as a tester ***will find more*** *defects than inspectors who have less experience as a tester,* **when using an Ad Hoc, procedurally-based checklist, or PBR technique**.

(Experience as tester) ∧ (Requirements inspection) ∧ (Ad hoc ∨ procedurally-based checklist ∨ PBR) ⇒ (more defects found)

2) **Experience in Testing based on Requirements**

This metric was collected in R3 (see Table 59) and R4 (see Table 72), but in R3 only one subject had industrial experience, so the results will be reported for R4 only. These results show that in most cases inspectors with more experience in testing based on requirements found more defects than those with less experience in testing based on requirements. The two cases where this result was not true were for inspectors using PBR on PG and inspectors using checklist on ATM. But when the data from both documents is analyzed together, subjects with industrial experience find more defects in all cases. The hypothesis H.SD.TE.3.Req was refined by *removing a constraint on the technique* to include the results from this study, while the hypothesis H.SD.TE.4.Req was supported and did not need refining. The original hypotheses were:

H.SD.TE.3.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience testing based on requirements ***will find more*** *defects than inspectors who have less experience testing based on requirements, when using a non-procedural checklist based method.*

(Experience testing based on requirements) ∧ (Requirements inspection) ∧ (non-procedural checklist) ⇒ (more defects found)

H.SD.TE.4.Req (Potential Hypothesis) For a requirements inspection, inspectors who have more experience testing based on requirements will ***not find any more or fewer*** *defects than inspectors who have less experience testing based on requirements, when using the PBR User perspective.*

(Experience testing based on requirements) ∧ (Requirements inspection) ∧ (PBR, User) ¬⇒ (more defects found)

The refined hypothesis, H.SD.TE.3.Req.RCT, with the removed constraint (underlined in non-italic bold) is:

H.SD.TE.3.Req.RCT (Potential Hypothesis): For a requirements inspection, inspectors who have more experience testing based on requirements ***will find more*** *defects than inspectors who have less experience testing based on requirements, when using a* **non-procedural or procedural** *checklist based method.*

(Experience testing based on requirements) ∧ (Requirements inspection) ∧ (non-procedural ∨ procedural checklist) ⇒ (more defects found)

Overall, the testing experience seems to be an important type of experience for inspectors to have. Testing experience typically requires some type of analysis skills in order to search for and detect problems in software. Consequently, it makes sense that this type of experience would be helpful for an inspector who is also analyzing an artifact looking for defects.

Experience in the PBR perspective

This metric was collected in R3 (see Table 60) and R4 (see Table 73), but there was only 1 subject in R4 who did not have industrial experience in their perspective, so the results will be reported for R3 only. In all cases, the inspectors who had more experience in their perspective found more defects than those who had less experience. This result supported hypothesis H.SD.PE.1.Req, so no change is necessary.

H.SD.PE.1.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience in their assigned PBR perspective ***will find more defects*** *than inspectors who have less experience in their assigned PBR perspective, when using PBR.*

(Experience in perspective) ∧ (Requirements inspection) ∧ (PBR) ⇒ (more defects found)

5.1.3 Process Experience

Comfort Reviewing Requirements

This metric was collected in R1 (see Table 40) and R2 (see Table 45). The results from those studies show that in most cases, inspectors with a high level of comfort reviewing requirements found more defects than those with a low level of comfort. The cases where this result was not true were the inspectors who were all using PBR. This result allowed hypothesis H.PE.1.Req to be refined by *removing a constraint on the technique* to broaden the hypothesis to include inspectors using a procedural checklist. The original hypothesis, H.PE.1.Req was:

> *H.PE.1.Req (Potential Hypothesis): For a requirements inspection, inspectors who are more comfortable with reviewing requirements* ***will find more*** *defects than inspectors who are less comfortable reviewing requirements, when using an ad hoc procedure.*
>
> **(Comfort reviewing requirements) ∧ (Requirements inspection) ∧ (ad hoc procedure) ⇒ (more defects found)**

The refined hypothesis, H.PE.1.Req.RCT, with the removed constraint (underlined in non-italic bold) is:

> *H.PE.1.Req.RCT (Potential Hypothesis): For a requirements inspection, inspectors who are more comfortable with reviewing requirements* ***will find more*** *defects than inspectors who are less comfortable reviewing requirements, when using an ad hoc procedure* **or a procedurally-based checklist**.
>
> **(Comfort reviewing requirements) ∧ (Requirements inspection) ∧ (ad hoc ∨ procedurally based checklist) ⇒ (more defects found)**

Experience with reviewing Requirements

This metric was colleted in R3 (see Table 51) and R4 (see Table 64). Overall the results seem to indicate that having more experience reviewing requirements did not help the inspectors find more defects. Hypothesis H.PE.5.Req stated that for inspectors who were not familiar with the domain, this type of experience would help. Based on the application domain knowledge data from R3, it appears that both PGCS and ATM were relatively unfamiliar domains for the inspectors, so this result does not support the earlier hypothesis. The main difference in these studies was the native language of the subjects. Therefore I can *add a constraint on the inspectors* to hypothesis, H.PE.5.Req. The original hypothesis, H.PE.5.Req was:

> *H.PE.5.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more experience reviewing requirements* ***will find more*** *defects than inspectors who have less experience reviewing requirements, when inspecting a requirements document which is either unfamiliar or from an unfamiliar domain.*
>
> **(Experience reviewing requirements) ∧ (Requirements inspection) ∧ (document unfamiliar ∨ domain unfamiliar) ⇒ (more defects found)**

The refined hypothesis, H.PE.5.Req.ACI, with the added constraint (underlined in non-italic bold) is:

> *H.PE.5.Req.ACI (Potential Hypothesis): For a requirements inspection, inspectors* **who are familiar with working in the language of the artifact being reviewed** *who have more experience reviewing requirements **will find more** defects than inspectors* **who are familiar with working in the language of the artifact being reviewed** *who have less experience reviewing requirements, when inspecting a requirements document which is either unfamiliar or from an unfamiliar domain.*
>
> **(Experience reviewing requirements) ∧ (Requirements inspection) ∧ (inspectors familiar with working language) ∧ (document unfamiliar ∨ domain unfamiliar) ⇒ (more defects found)**

Software Inspection Experience

This metric was collected in R3 (see Table 52) and R4 (see Table 65). The results from these two studies show that in most cases inspectors with more experience performing software inspections do not find more defects than those with less experience. These results support hypothesis H.PE.8.Req that detailed techniques interfere with the normal process for experienced inspectors and cause them to find fewer defects. The results also support hypothesis H.PE.9.Req. The type of inspection experience the subjects had was not collected in this study, so we still cannot be sure if the experience they had was in the same type of inspection conducted in this study. Therefore, H.PE.8.Req and H.PE.9.Req below remained unchanged:

> *H.PE.8.Req (Potential Hypothesis): For a requirements inspection, for inspectors who are highly experienced in performing inspections, detailed techniques get in their way and prevent the optimal performance during the inspection.*
>
> **(Experience in performing inspections) ∧ (Requirements inspection) ∧ (detailed techniques) ¬⇒ (more defects found)**
>
> *H.PE.9.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more software inspection experience **will find** more defects than inspectors who have less software inspection experience, when that experience is with the same style of inspection currently being used.*
>
> **(Inspection experience) ∧ (Requirements inspection) ∧ (same style of inspection) ⇒ (more defects found)**

5.1.4 Working Language Experience

Native Language

Because the subjects in these replications were all Brazilians, none of them spoke English as their native language. Therefore, there were no results for this metric.

Reading Comprehension

The results from this metric, collected in all four studies, show that in most cases subjects who have higher English reading comprehension skills find more defects than those with lower English reading comprehension skills. This result is especially true in R3 (see Table 61) and R4 (see Table 74) with more advanced inspectors. Because the inspectors in R3 and R4 were graduate students and industrial professions versus the undergraduates in R1 (see Table 43) and R2 (see Table 48), it is quite possible that they have had more experience working in English than the other inspectors. Because the metric was self-reported, there is no way for the researchers to verify the level of reading comprehension reported by the subjects. These results allowed hypothesis H.WL.2.Req to be modified by *removing a constraint on the inspector*. The original hypothesis, H.WL.2.Req was:

> *H.WL.2.Req (Potential Hypothesis): For a requirements inspection, inspectors who have a higher comfort level in reading comprehension of the working language* ***will find more*** *defects than inspectors who have a lower comfort level in reading comprehension of the working language, for inspectors who are not native speakers of the working language.*
>
> **(Comfort level in reading comprehension) ∧ ¬(Native speaker of working language) ∧ (Requirements inspection) ⇒ (more defects found)**

The new hypothesis, H.WL.2.Req.RCI, with the increased domain (underlined in non-italic bold) is:

> *H.WL.2.Req.RCI (Potential Hypothesis): For a requirements inspection, inspectors who have a higher comfort level in reading comprehension of the working language* **or those who have worked in that language** ***will find more*** *defects than inspectors who have a lower comfort level in reading comprehension of the working language* **or those who have not worked in that language***, for inspectors who are not native speakers of the working language.*
>
> **(Comfort level in reading comprehension ∨ Experience working in the language) ∧ ¬(Native speaker of working language) ∧ (Requirements inspection) ⇒ (more defects found)**

Listening and Speaking Skills

This metric was collected in all four studies, but for R3 there was only one subject with high English listening and speaking skills, so the results will only be reported for R1 (see Table 44), R2 (see Table 49), and R4 (see Table 75). The results from these studies show that in most cases for the ATM document and overall subjects who have higher English listening and speaking skills find more defects than those with lower skills. It is not clear why the results for the PGCS requirements document were opposite of the other results. Therefore, I left the earlier hypothesis unchanged, but if future results support the PGCS result, then the hypothesis will need to be modified.

5.1.5 Level of Process Specificity

As mentioned in the introduction to this section, the level of process specificity in the PBR technique varied across the four studies discussed here. In R1 and R2, which were using low experienced undergraduate subjects, the level of process specificity was low, i.e. the inspectors were not given a step-by-step procedure to create an abstraction model relevant to their perspective. On the other hand, in R3 and R4, which were using more experienced graduate students and industry professionals, the level of process specificity was high, i.e. the inspectors were given a step-by-step procedure to create a specific abstraction model in order to help them find defects.

The data from Table 1 shows that in most cases, the inspectors did not realize the expected gain when using PBR versus a checklist. In R1, for the PGCS the two sets of inspectors found about the same number of defects, while in the ATM, the PBR inspectors did see a small gain over the checklist inspectors. In R2, the PBR inspectors did slightly better than the checklist inspectors on both requirements documents, but the increase was not as large as expected. In R3, there was slight increase for PBR inspectors on the PGCS, but the checklist inspectors found twice as many defects on the ATM. Finally, in R4, the most experienced subjects, the checklist inspectors found more defects tan the PBR inspectors on both requirements documents.

Table 1 – Percent of Defects Found (Brazilian studies)

	PGCS		**ATM**	
	PBR	Checklist	PBR	Checklist
R1	15.3%	15.6%	11.7%	7.5%
R2	15.3%	13.5%	12.3%	10.2%
R3	17.7%	14.5%	6.6%	12.9%
R4	8.7%	17.7%	10.3%	15.6%

In a discussing these results, we determined, based on discussions Dr. Maldonado had with the subjects, that *Level of Process Specificity* influenced the inspectors' performance with PBR. The conclusion was reached that the amount and type of detail included in a technique must match up with the inspectors' experience in their assigned perspective. More experienced inspectors probably do not need as much detail because they are able to create their own abstraction model based on their experiences. On the other hand, the less experienced inspectors, most likely will need a more detailed, step-by-step procedure that helps them to create a simple abstraction model to aid in defect detection. As a result of this study two new hypotheses were formed:

> *H.PS.1.Req (Potential Hypothesis) For a requirements inspection, inspectors who are experienced in the PBR perspective they will assume* ***will be more effective*** *using a PBR technique with a low level of detail than when using a more detailed PBR procedure.*
>
> **(Experience in perspective) ∧ (Requirements inspection) ∧ (Low detail technique) ⇒ (more defects found)**

H.PS.2.Req (Potential Hypothesis) For a requirements inspection, inspectors who are not experienced in the PBR perspective they will assume ***will be more effective*** *when using a PBR technique with a step-by-step procedure to help them create an abstraction model than when using a PBR technique with less detail.*

¬(Experience in perspective) ∧ (Requirements inspection) ∧ (step-by-step technique) ⇒ (more defects found)

5.2 Advice for Practitioners

Up to this point, I have provided results of interest for the research community. To conclude this chapter, I will provide some concrete suggestions to the practitioner community. The first piece of advice is that the background knowledge of testing, seems to be one of the most important types of knowledge for an inspector to have. Because testing requires a similar type of analysis ability that performing an inspection does, this conclusion makes sense. Therefore, practitioners should try to populate their inspection teams with inspectors who have some testing experience. The second issue that is important for practitioners is that for many of the experience variables, the presence of an inspection technique neutralizes the difference in performance among inspectors. Detailed techniques are especially useful for inexperienced inspectors to help make up for their lack of experience. On the other hand, those detailed techniques do not seem to help and in some cases even hurt more experienced inspectors. The recommendation is that the level of detail provided in an inspection technique be properly tailored based on the level of experience of the inspector who will use that technique. Finally, knowledge of the application domain is important during a requirements inspection where no document from a previous lifecycle exists, but it is not as important during a design inspection, where there is a requirements document for inspectors to use as a reference.

6. Studies designed to further study hypotheses in detail

Studies must be designed to specifically test those hypotheses that are of special interest. This chapter presents two studies that fall into this category. The study in Section 6.1 was designed to further study the Process Experience variable. Section 6.2 presents a study that was designed to study the Process Specificity variable.

6.1 Study designed to test ***Process Experience*** variable (CMSC 735 Fall 2001)

We have learned through the previous studies that having experience with performing inspections helps an inspector to find more defects. But, there were still some open questions about this experience:

1) How much experience does the inspector need before an effect is seen?
2) What are the most effective methods of gaining that necessary experience?

By answering these questions, guidance can be provided on simplifying the training and methods for building experience for an organization. These questions led to the design of the first experiment. Hypotheses H.PE.9.Req, dealing with the software inspection experience metric, was used as a basis for this study.

> *H.PE.9.Req (Potential Hypothesis): For a requirements inspection, inspectors who have more software inspection experience* ***will find*** *more defects than inspectors who have less software inspection experience, when that experience is with the same style of inspection currently being used.*
>
> **(Software Inspection experience) $\wedge$ (Requirements inspection) $\wedge$ (same style of inspection) $\Rightarrow$ (more defects found)**

This study explored this hypothesis in the context of a specific inspection technique, (PBR). The PBR procedure asks each inspector to take a specific role or *perspective* while inspecting the requirements, such as tester, user or designer. Each inspector then looks at the requirements with that perspective in mind with the goal of verifying that the requirements are correct from their assumed perspective.

Previous studies of PBR had shown that the technique was feasible and effective at detecting defects, although the effectiveness varied from inspector to inspector. It made sense to begin expanding the understanding of the variables impacting the effectiveness of PBR. A useful tool for investigating these variables is experimentation. Two approaches are available when designing an experiment. The experiment can either be a brand new experiment, designed from scratch, or it can be a replication of a previous experiment. Experimental replications are a good method for understanding variables because, in a replication, the design of the original study can be slightly modified to shift the focus on to a different variable than the original study. This experiment is a replication of the requirements inspection part of the experiment run in the CMSC735 Fall 1999 class, which is described in Section 4.2.1.4.

One of the issues this study was designed to address was understanding how process experience could be gained. One potential method for an inspector to gain process experience is through observing a PBR inspection prior to performing his or her own inspection. A secondary goal of this study was to understand the steps of the PBR procedure better, in order to determine if steps should be added, changed, reordered or removed. In order to give the subjects a chance to observe a PBR inspection and to study PBR at the level of individual steps, this study used an observational approach. An observational approach is an experimental method suitable for understanding how a process is applied. In an observational study, a subject performs a process while an experimenter observes. Often the subject is instructed to think-aloud so the experimenter can better understand their thought processes. This type of study allowed half of the subjects to serve as the experimenters and observe another subject prior to performing their own inspection. In addition, this study provided a level of detail about individual process steps and their usefulness, such as steps that were confusing or out of order, that is difficult to collect using traditional post-experiment questionnaires [Singer96].

Based on the above goals, the specific hypotheses tested in this study were three new hypotheses, that are related to H.PE.9.Req, created to address the specific Process Experience goals described above, H.PE.10, H.PE.11 and H.PE.12.

H.PE.10 (Potential Hypothesis): In a requirements inspection, inspectors with experience using PBR ***will find more*** *defects than those who do not have experience using PBR.*

(Experience using PBR) ∧ (Requirements inspection) ⇒ (more defects found)

H.PE.11 (Potential Hypothesis): Process experience can be gained by observing an inspection

(Observation) ⇒ (Process Experience)

H.PE.12 (Potential Hypothesis): In a requirements inspection, inspectors who observe a PBR inspection ***will find more*** *defects than those who do not.*

(Observation of PBR) ∧ (Requirements inspection) ∧ (PBR) ⇒ (more defects found)

6.1.1 Experimenters

Researchers at the University of Maryland, who had created the techniques and were therefore experts in their use, performed both the original experiment and the replication.

6.1.2 Subjects

The subjects were graduate students at the University of Maryland enrolled in a graduate level Software Engineering class in the fall semester of 2001. The subjects were paired up with one subject acting as the *executor* (responsible for applying the procedure) and the other as the *observer* (responsible for recording observations about the application).

In this study, there were 26 subjects grouped into 13 pairs. As will be described in the Procedure section each pair performed two inspections, switching roles in between. This setup allowed all 26 subjects to perform a requirements inspection. For these subjects, 35% had industry experience writing requirements, and another 39% had classroom experience; 19% had industrial experience writing Use Cases and another 50% had classroom experience; 38% had industrial experience reviewing requirements and another 46% had classroom experience.

6.1.3 Materials

The PBR reading techniques were applied to the requirements documents from two systems: one for the Loan Arranger (LA) system described in Section 4.2.1.2, and one for an automated parking garage control system (PGCS) described in Section 4.2.1.1. The LA requirements had 8 pages, 26 functional requirements and 4 non-functional requirements. The PGCS requirements had 17 pages, 21 functional requirements, and 9 performance requirements.

6.1.4 Procedure

A background questionnaire was distributed to the subjects to collect relevant information about their software development experiences. A copy of this questionnaire can be found in Appendix H. Based on the results of this questionnaire, the subjects were assigned to the proper group to satisfy the experimental assumptions.

This experiment consisted of only an individual review. There was no team meeting to collect or detect more defects. The subjects performed only one inspection of a requirements document.

Before the study, subjects received training in the reading techniques to be applied and the observational methods. Training in observational methods was done by presenting the roles of executor and observer and defining their specific responsibilities; a short example was performed in class. Because each team had to write a summary report, the subjects were asked to come up with their own questions during the observation for eliciting information about the overall effectiveness of the techniques and the way in which the process was applied (e.g. if the procedure was too detailed or missing key information). As part of the training, after the classroom lecture, each team spent 30-45 minutes with a researcher working through the PBR procedure and the observational techniques on a sample set of requirements. The goal of this activity was for the researcher to verify that the subjects understood their tasks and were performing them properly.

In order to test the effects of learning, the experiment described here added a new step to the original experiment. Instead of performing only one requirements inspection, the pair of subjects performed two different requirements inspections. In the first inspection, roughly half of the teams inspected the requirements for the LA and the other half the requirements for the PGCS. After this inspection was complete, the team members switched roles, i.e. the process observer in the first inspection became the process executor in the second inspection. The teams also switched requirements documents, from LA to PGCS or vice-versa. All subjects used only the User perspective of PBR. The experimental design is presented in Figure 8.

Treatments	**Group 1:** 4 Low Experience Teams 3 High Experience Teams	**Group 2:** 3 Low Experience Teams 3 High Experience Teams
Review #1	LA	PGCS
	Switch Roles	*Switch Roles*
Review #2	PGCS	LA

Figure 8 – Design of Study 1

After performing the two inspections, each team wrote a report discussing their experiences and evaluating the PBR procedure. They were told to discuss the methods they used to understand the PBR procedure. The report also addressed whether or not PBR was useful for the task it was designed to accomplish. And finally, the report included any suggested improvements to PBR. All of these conclusions had to be backed up by data from the observations recorded during the inspection.

Finally, after the subjects had completed the inspections and submitted their evaluation reports, a class discussion was held. The goal of this discussion was to show the subjects the experimental design as well as the high-level goals and hypotheses. A preliminary analysis of the data they had provided was also given. Subjects were given an opportunity to discuss the results with the researchers and clarify confusing items in the data providing an initial check on the validity of the results.

6.1.5 Data Collection

Both quantitative and qualitative data was collected. The quantitative data included the time required to perform the inspection using PBR, and the number and type of defects detected. The qualitative data was collected using the observational techniques and included in the report described earlier. The report included both direct observations taken during the inspection as well as retrospective, or post-hoc, information. This data included:

- A subjective evaluation of the effectiveness of the technique.
- Any specific problems with steps in the technique

The retrospective data included:

- Usefulness of the different perspectives
- Practicality of the techniques and whether the subjects would use them again
- High-level or global problems with the techniques.

In addition to this data, the teams were asked to discuss, in the report, any effects on the second inspector as a result of observing the first inspection. One more source of data was the class discussion after the experiment had been run. A copy of the assignment description given to the students can be found in Appendix H.

6.1.6 Results/Lessons Learned About Process Experience

Based on the data collected in this study there were both qualitative and quantitative results. The two results tell somewhat conflicting stories. First these results are presented and then a discussion of why the results might differ follows. This section concludes with ideas for future study.

The *quantitative data* is split on whether observing a PBR inspection prior to performing one helps an inspector find more defects. Because there were two groups of subjects, high experience and low experience, the data was analyzed in two ways. First the data was analyzed collectively (high experience and low experience together), and then each of the two experience groups was analyzed separately.

Table 2 – Experimental Data (Study 1)

Experience	Artifact	First Inspection	Second Inspection	p-value	
Low	PGCS	23.5% (3)	10.3% (4)	.23	X
	LA	12.5% (4)	31.5% (3)	.02	√
	PGCS & LA	17.2% (7)	19.4% (7)	.738	
High	PGCS	9.8% (3)	11.8% (3)	.643	√
	LA	18.5% (3)	11.1% (3)	.295	X
	PGCS & LA	14.2% (6)	11.4% (6)	.484	
High and Low	PG	16.7% (6)	10.9% (7)	.293	X
	LA	15.1% (7)	21.3% (6)	.32	√
	PGCS & LA	15.8% (13)	15.7% (13)	.979	X

Table 2 above shows the quantitative results for this study. The results are grouped by experience level (low experience, high experience, or all together) and artifact inspected (PGCS, LA, or all together). For each row, the average number of defects found by the subjects who fell into that category for the first inspection and the average number of defects found by subjects that fell into that category for the second inspection are shown. The numbers in parenthesis indicate the number of subjects that each average represents. The p-value is from a t-test run to determine if there was a difference between those subjects in the first inspection and those in the second inspection. An alpha level of .1 was used. In the last column a '√' indicates that subjects in the second inspection found more defects on average than those in the first inspection, while an 'X' indicates the opposite.

The quantitative results are mixed. For low experience subjects, the subjects who had first observed a PBR inspection, found more defects both in the LA document alone and in the two documents overall (PGCS & LA). Only the results for the LA document were statistically significant. On the other hand, for the PGCS document, the subjects in the second inspection did considerably worse than those in the first inspection.

For the high experienced subjects, the results are the exact opposite from the low experienced subjects. On the PGCS document, inspectors who observed an inspection prior to performing one found more defects than those who had not, but on the LA and on both documents together, the subjects who had not observed an inspection before performing one found more defects.

Finally, when all the subjects are taken together, on both documents, they exhibit basically the same behavior regardless of whether they were in the first inspection or the second inspection. But, when looking only at the PGCS document, the second inspectors did worse. For the LA document, the second inspectors did better.

The quantitative data did not provide support for the hypotheses that having observed a PBR inspection improves the performance of an inspector. The qualitative data provides a more complete picture with regard to subject perception.

The qualitative results come from the report written by the subjects where they were asked to discuss the impact of observing an inspection prior to doing their own inspection. (Note: It should not be assumed that the remaining subjects in each of the bullet points below disagreed with the point being made, in fact many did not address the issue at all.)

- Subjects from teams 1, 10, 11, and 13 (4 out of 13 or 23%) stated that observing the inspection process helped the second inspector better understand the process.
- Subjects from teams 2, 3, 5, 6, 7, 8, and 13 (7 out of 13 or 54%) stated that observing the inspection process helped the second inspector to better understand and perform the individual steps of the process
- Subjects from teams 1,2, 4, 5, and 11 (5 out of 13 or 38%) subjects stated that observing the inspection process helped the second inspector be more confident or efficient.

Overall, none of the subjects indicated that observing the inspection first hurt the effectiveness of the second inspector. The two teams that did not make positive comments, simply did not comment on the issue at all:

- Subjects from teams 9 and 12 (2 out of 13 or 15%) did not comment on the effect of observing the inspection first
- None of the subjects stated that observing the first inspection either hurt their performance or had no effect on their performance.

Based on this feedback as well as the class discussion with the subjects after the experiment was complete, there was an indication that subjectively they found the opportunity to observe PBR before using it helpful, even though the quantitative data does not show this result consistently.

There are two potential explanations for these conflicting results. First because different subjects performed the first inspection and the second inspection the analysis had to be done between-subject rather than within-subject to judge improvement; therefore, there is the possibility that the set of subjects who performed the first inspection were simply better inspectors than those who performed the second inspection. The second potential explanation is that the expected benefit gained from *Process Experience* only occurs if the same inspector performs both inspections rather than just observing the first one. This conclusion would contradict H.PE.11 and H.PE.12. Based on these results, I proposed a new hypothesis, H.PE.13, which was related to the hypotheses tested in this study.

H.PE.13 (Potential Hypothesis): Inspectors who observe an inspection using the same technique or technology ***will not find more*** *defects than inspectors who do not observe an inspection.*

(Observation of an inspection) $\neg\Rightarrow$ (more defects found)

A new experiment can now be designed and run to test this hypothesis. The experiment would be designed such that each subject performs more than one inspection on multiple requirements documents using the same inspection technique such as PBR. The defect rate from the second inspection can then be compared with the defect rate for the first inspection within-subject to determine if there is any effect. For more details about this study see [Carver03].

6.2 Experiment to test the *Level of Process Specificity* variable (CMSC735 Fall 2002)

Based on the data from the previous studies, it appeared that in many cases, the presence of an inspection technique seemed to neutralize the effects of experience. Additionally, it was hypothesized that this result arose because of the details in the technique getting in the way of the innate process that an expert would normally use. So, in order to better understand if the amount of detail in a technique was an important issue, a study was run. This study was designed to address the following questions:

1) Does the level of process specificity matter in an inspection?
2) Does an inspector's software development experience have an effect on the amount of detail they need in an inspection process

By answering these questions, advice could be provided to an organization about how to choose (or not choose) techniques based on the backgrounds of their inspectors.

In this study the variable *Level of Process Specificity* was examined. Based on the questions above, the main idea was to begin understanding the impact that Software Development Experience has on the execution of a specific process for inspection of a requirements document, Perspective Based Reading (PBR). The PBR processes provide inspectors with a step-by-step process for constructing an abstraction of the requirements into a model applicable to the assigned perspective. Then it proposes questions for the inspector to answer to help uncover defects. The amount of detail in the PBR techniques can be tailored depending on numerous factors. The general hypothesis for this study was:

> *H.PS The level of specificity necessary in an inspection process is related to the level of experience the inspector has.*

Effectiveness was defined as number of defects found. In exploring H.PS, many of the aspects of software development knowledge, defined earlier, could have been explored. In this study I chose to study the relationship between process detail and an inspector's experience in the specific PBR perspective (designer, tester, user) assigned to the inspector. This choice led to the testing of the two hypotheses uncovered during the review of the data of the studies from Brazil, discussed in Chapter 5.

> *H.PS.1.Req (Potential Hypothesis) For a requirements inspection, inspectors who are experienced in the PBR perspective they will use* ***will be more effective*** *using a PBR technique with a low level of detail than when using a more detailed PBR procedure.*
>
> **(Experience in perspective) ∧ (Requirements inspection) ∧ (Low detail technique) ⇒ (more defects found)**

> *H.PS.2.Req (Potential Hypothesis) For a requirements inspection, inspectors who are not experienced in the PBR perspective they will use* ***will be more effective*** *when using a PBR technique with a step-by-step procedure to help them create an abstraction model than when using a PBR technique with less detail.*

$\neg$(Experience in perspective) $\wedge$ (Requirements inspection) $\wedge$ (step-by-step technique) $\Rightarrow$ (more defects found)

To address these two hypotheses, two versions of the PBR technique were needed, one containing a high level of detail and one containing a low level of detail. In previous studies where a step-by-step technique was used subjects encountered problems because the abstraction model was too low level and detailed for a requirements inspection. Because of these problems, the step-by-step PBR procedure for creating an abstraction model needed to be tailored to include a model that was more abstract and higher level. For instance, in previous studies, the abstraction model for the tester perspective was a set of test cases created by using a specific testing technique, equivalence partition testing (which is typically used at the unit testing stage). The version of PBR in this study used a test case model that was more suitable for creating test cases based on a requirements document. Therefore, a new hypothesis was created, H.PS.3 for this study:

H.PS.3.Req (Potential hypothesis): For a requirements inspection, too much detail, for novices or experienced developers, will hurt the performance of the inspector.

(Requirements inspection) $\wedge$ (technique with too much detail) $\Rightarrow$ $\neg$(more defects found)

Based on this hypothesis, depending on their level of experience, inspectors should get as little detail as possible while still being able to do their job.

6.2.1 Subjects

The subjects were students in a graduate level software engineering class in the Fall 2002 semester at the University of Maryland. Approximately 1/3 of the students were experienced testers, and the other students were not experienced.

6.2.2 Materials

For the inspection in this study the PGCS requirements document, described in Section 4.2.2.1, was seeded with defects. Subjects used one of two versions of PBR to perform the inspection.

Version 1 of the PBR technique was a high-level version that did not contain a step-by-step procedure to create an abstraction model of the requirements. Rather, the technique instructed the inspector to create an abstraction model, a set of test cases, using any method they were familiar with, and then it provided a series of questions for the inspector to answer to help uncover defects.

Version 2 of the PBR technique included a step-by-step procedure to build the abstraction model, a set of test cases, based on the Category Partition Testing technique [Ostrand88]. This technique was simple enough that it was hoped that the inspectors would be able to easily understand the model and focus more attention on answering the questions and uncovering defects. The two versions of PBR can be found in Appendix I.

6.2.3 Procedure

The first step of this study was for each subject to complete a background and experience questionnaire, a copy of which can be found in Appendix I. This questionnaire collected information about experience in a number of areas of software development, including information about experience in the PBR perspectives. Due to the small number of subjects in this study, it was only feasible to use one of the PBR perspectives. So, the responses from questionnaire were used to select a perspective that had a good combination of experienced and inexperienced subjects. Based on the results of the questionnaire, the tester perspective was chosen.

The second step was to split the subjects into a high experience and a low experience group. The subjects were split such that those with more than one year of industrial experience were classified as high experience, and the rest as low experience. This decision was made prior to looking at the responses from the questionnaires so that the division of subjects in to the high experience group and the low experience group was not biased by the data.

Because there were two versions of the PBR techniques, the original design of this experiment was to split both groups in half so that there would be both high and low experienced subjects using each version of the PBR technique. But, there were not enough high experienced subjects, so they were all placed in one group.

The low experience subjects were split between the two versions of the techniques as follows; a few of the low experience subjects were concurrently enrolled in a testing course, in which they were exposed to, but did not use, the category partition testing technique used in the PBR technique, so those subjects were assigned to the version of PBR that contained the category partition testing details. The remaining subjects were randomly assigned to one of the two versions of the PBR technique. The complete design can be seen in Table 3.

Table 3 – Experimental Design (Study 2)

	Group 1	~~**Group 2 (not enough subjects to use)**~~	**Group 3**	**Group 4**
Perspective Experience	High	~~High~~	Low	Low
Process Detail	No Model	~~Model included~~	Model included	No Model
Model	Own	~~Abstract~~	Category Partition Testing	Own

Based on the technique assigned to each subject, they received the appropriate training. The majority of the training was the same for all subjects regardless of which technique there were assigned to use. The training covered 2 class periods as shown in Table 4.

The pretest at the beginning of the training session was used to evaluate the subjects' level of knowledge and experience about requirements defects and about creating test cases. In the pretest the subjects were asked to 1) identify and describe defects based on an excerpt from a requirements document, which they were given, and 2) create test cases from the same requirements excerpt. Table 4 shows the specific topics covered during the training sessions. At the completion of the training session (prior to the use of PBR) each subject was given a post-test, similar to the pretest, to evaluate what they had learned from the training. The two requirements excerpts used for the pretest and posttest (ABC Video excerpt and ATM excerpt) were balanced such that half of the class had ABC Video for the pretest and the other half ATM, with each subject receiving the second document as their posttest.

After the posttest, the subjects were given the appropriate version of PBR and the requirements document to be inspected. Working alone, the subjects performed the inspection of the requirements document, recording any defects detected. For each defect, the subjects also recorded the time they found the defect and the type or class (from a predetermined scheme) of the defect.

Table 4 – Training Sessions (Study 2)

Class Period 1

Activity	Time	Topic	Handout	Slides
Pretest	9:30-9:45	--	ABC / ATM	2
Training	9:45-10:00	Inspection Background	--	3-13
	10:00-10:30	Software Defects	Gas Station	14-21
	10:30-10:45	Reading Techniques	--	22-27

Class Period 2

Activity		Time	Topic	Handout	Slides
Training		9:30-10:00	PBR	--	28-42
Assignment		10:00-10:05	Explanation	--	43
Split Class		10:05-10:10	--	--	44
Group 3	Training	10:10-10:35	Category Partition Testing	PBR Technique	45-60
	Posttest	10:35-10:45	--	ABC / ATM	61
	Assignment	10:45	Pass out Assignment	Assignment / PG / Defect Form	--
Groups 1 & 4	Posttest	10:10-10:20	--	ABC / ATM	--
	Assignment	10:20	Pass out Assignment	Assignment / PBR Technique / PG / Defect Form	--

After completion of the inspection, the subjects were given a questionnaire where they were asked about their experiences with PBR. The questionnaire covered issues dealing with the level of detail in the technique they were given, as well as their background and how those two items impacted their performance. There were two versions of the questionnaire, depending on which version of the technique the subject used. A copy of the post-study questionnaires can be found in Appendix I.

6.2.4 Data Collection

Both quantitative and qualitative data was collected in this study. The quantitative data collected included the time required to perform the inspection using PBR, and the number and type of defects detected. The qualitative data was collected via a post-experiment questionnaire as well as through subject feedback, both in their comments on the defects lists as well as during the class discussion.

Table 5 – Defect Rates (Study 2)

	Group 1	Group 3	Group 4
Perspective Experience	High	Low	Low
Process Detail	No Model	Model included	No Model
Model	Own	Category Partition Testing	Own
Average Defect Rate	22.1%	20.6%	26.5%

6.2.5 Data Analysis

Table 5 shows the results based on the initial scoring of the subject's defect report forms. Hypotheses H.PS.1 and H.PS.2 implied that Group 1 and 3 would find more defects on average than Group 4, because Group 1 had more experience in the perspective and therefore should have been familiar with a testing technique of their own that they could use for creating the test cases, and Group 3 was given a technique that showed them how to create test cases, while Group 4, who were not experienced testers, were told to create test cases on their own, with no help from a technique.

The hypotheses were not supported. However, given the small sample size, I examined the data for statistical outliers since they could have greatly influenced the results. One subject from Group 1 and one subject from Group 4 were found to be statistical outliers. Table 6 shows the revised results after eliminating the outliers.

Also, because versions of the tester perspective of PBR had been used in previous studies on this requirements document, the data from this study was compared with historical data to understand if the different versions of the techniques had any effect. For the high experienced subjects the results were:

- Subjects from this study, who used a PBR technique with no specified model for creating test cases found 24.1% of the defects
- Subjects from CMSC735 1997 who used a PBR technique that included a very specific method (Equivalence Partition Testing) for creating test cases, found 27.1% of the defects

- o Subjects from R3 in Brazil who used a PBR technique that included a very specific method (Equivalence Partition Testing) for creating test cases, found 17.7% of the defects
- o Subjects from R4 in Brazil who used a PBR technique that included a very specific method (Equivalence Partition Testing) for creating test cases, found 8.7% of the defects

There was statistical significance that the subjects in this study found more defects than the subjects from Brazilian studies R3 and R4.

Table 6 – Defect rates (without statistical outliers)

	Group 1	Group 3	Group 4
Perspective Experience	High	Low	Low
Process Detail	No Model	Model included	No Model
Model	Own	Category Partition Testing	Own
Average Defect Rate	22.1%	20.6%	26.5%
Average Defect Rate (after removing outliers)	24.1%	20.6%	21.8%

For the low experience subjects the results were as follows:

- o Subjects from this study who used a PBR technique with no specified model for creating test cases found 21.8% of the defects
- o Subjects from this study who used a PBR technique with a simple model (Category Partition Testing) for creating test cases found 20.6% of the defects.
- o Subjects from CMSC735 1997 who used a PBR technique that included a very specific method (Equivalence Partition Testing) for creating test cases found 27.1% of the defects.

None of these results were statistically significant.

6.2.6 Results

The data discussed above, both the comparisons between techniques within this study and comparisons to historical data, do not support hypotheses H.PS.1 and H.PS.2 that the level of detail in an inspection technique must be tailored to the amount of software development experience of the inspector who will be using that technique. There are some potential threats to the validity of this result. First, the data from this study was analyzed against data from previous studies that were run in different settings with different subjects. While every effort was made to compare data only from comparable subjects, there is no way to ensure this. Secondly, during the class discussion of this experiment, some of the subjects raised the point that most of the defects that they detected in the requirements document would have been found without using any specific technique at all; therefore the expected benefits from the techniques could not be seen by using the documents used in this study. One of the major issues was the complexity of the document. The subject in the class believed that the artifact that they inspected was simple enough, that defects were easy to find without a specified technique. This result means that another dimension that needs to be considered is the interaction between artifact complexity and the inspection process.

Based on the results of this study, I refined H.PS.1.Req and H.PS.2.Req by *adding a constraint on the inspected artifact*. The original versions of H.PS.1.Req and H.PS.2.Req were:

> *H.PS.1.Req (Potential Hypothesis) For a requirements inspection, inspectors who are experienced in the PBR perspective they will assume* ***will be more effective*** *using a PBR technique with a low level of detail than when using a more detailed PBR procedure.*
>
> **(Experience in perspective) ∧ (Requirements inspection) ∧ (Low detail technique) ⇒ (more defects found)**
>
> *H.PS.2.Req (Potential Hypothesis) For a requirements inspection, inspectors who are not experienced in the PBR perspective they will assume* ***will be more effective*** *when using a PBR technique with a step-by-step procedure to help them create an abstraction model than when using a PBR technique with less detail.*
>
> **¬(Experience in perspective) ∧ (Requirements inspection) ∧ (step-by-step technique) ⇒ (more defects found)**

The refined hypotheses with the added constraint (underlined in non-italic bold) are:

> *H.PS.1.Req.ACA (Potential Hypothesis) For a requirements inspection, inspectors who are experienced in the PBR perspective they will assume* ***will be more effective*** *using a PBR technique with a low level of detail than when using a more detailed PBR procedure,* **when inspecting a requirements document of sufficient complexity**.
>
> **(Experience in perspective) ∧ (Requirements inspection) ∧ (low detail technique) ∧ (artifact of sufficient complexity) ⇒ (more defects found)**
>
> *H.PS.2.Req.ACA (Potential Hypothesis) For a requirements inspection, inspectors who are not experienced in the PBR perspective they will assume* ***will be more effective*** *when using a PBR technique with a step-by-step procedure to help them create an abstraction model than when using a PBR technique with less detail,* **when inspecting a requirements document of sufficient complexity**.
>
> **¬(Experience in perspective) ∧ (Requirements inspection) ∧ (step-by-step technique) ∧ (artifact of sufficient complexity) ⇒ (more defects found)**

To continue this investigation, another study should be run that addresses the shortcomings of this study. In the next study, subjects should use all three versions of the PBR technique so that comparisons among the versions of the techniques can be done within the same subject pool rather than having to compare to historical data drawn from another subject pool. Secondly, a better mechanism, such as some sort of test, for rating each subject as having high or low experience needs to be developed. The current method relies only on the subjects' self-reporting which may or may not be consistent and accurate.

Finally, the artifact being reviewed needs to be studied further to understand the scope of applicability of inspection techniques. One of the issues is that artifacts are difficult to develop and the ones used in the experiments were developed in an ad hoc fashion, seeded with defects by defect reduction experts. There is not yet a theory on how to select an appropriate set of artifacts for testing different aspects of the reading techniques, i.e., varying the artifacts according to application domain, defects that are seeded, and level of complexity of understanding. A paper by Miller [Miller01] has shown that the results of related experiments could not be combined using meta-analysis on diverse experiments. The software engineering community has not defined the relevant characteristics of benchmarks or the value ranges of those characteristics for describing requirements documents.

7. Results

This section contains the two major results of this work. A description of the refined grounded theory methodology that I used is in Section 7.1. The complete list of hypotheses generated by this work is in Section 7.2

7.1 Refinement of Grounded Theory Methodology

This section describes the implementation and tailoring of the grounded theory research approach that I performed. This methodology can be reused by other researchers who are asking the following question: Which variables affect the outcome of a process? The three major steps involved in the methodology are: 1) Define an initial list of variables and hypotheses; 2) Refine the list of variables and hypotheses by making them more concrete and adding constraints; and 3) Run new studies. More details about each of these steps follow.

In general the methodology can be used to refine the following formula:

(variables $\boldsymbol{x_i}$) affect the (outcome $\boldsymbol{y_i}$) of (process $\boldsymbol{p}$) in a (positive/negative) way

Where *X*, the set of x_i, are the variables we are interested in identifying, the y_i are the measurable outcomes of the process *p* that has been chosen for study. The set of variables *X*, may be bounded based on the researchers expertise. For example, in my work, I chose *p* as “software inspections”, the outcome *y* to be “number of defects detected”, and bounded the set of variables *X* to be variables in the context of the inspection process.

Define an initial list of variables and hypotheses

Inputs: The process *p*, an initial idea of y_i, any bounds on the x_i

Steps:

1) Review literature describing studies conducted to specifically address the relationship between the x_i and the y_i. Record any variables identified in the results of those studies.
2) Review literature describing studies conducted on the chosen process, even if the goal of the study was not to specifically relate the variables in the chosen context to the outcome of the chosen process.
    a. It is important to note that in cases where variables in the context were not studied specifically, researchers often implicitly identify candidate variables in the choices made when designing the study, or in explaining the results of the study.
    b. Record any variables that were taken into account when designing the study, or any variables that were identified as potential explanations for unexpected results of the study.
3) Review literature from other domains, e.g. Education or Psychology, looking for studies that deal with the relationship between a generalized version of the chosen process *p* or any process p_i and the specific outcomes y_i or a generalized outcome.
    a. Note cases where variables that are already defined in the above steps find support in this more general literature.
    b. Record any new variables related to the more general process that can be tested on the process chosen for study.
4) For each identified variable, generate a high-level hypothesis of the form:
    a. (Variable x_i) $\Rightarrow$ (outcome y_i) [positive effect]

b. (Variable x_i) $\Rightarrow \neg$ (outcome y_i) [negative effect]
c. (Variable x_i) $\neg\Rightarrow$ (outcome y_i) [no effect]

5) Assign a number each hypothesis to allow for easier tracking. An abbreviation for each of the identified variables should be created, and assigned to the new hypothesis. For example, a high-level hypothesis dealing with Domain Knowledge would be assigned the number '*H.DK*'. This numbering will be updated as the hypotheses evolve.

Output: The list of variables and high-level hypotheses related to those variables.

Refining the Variables and Hypotheses

Input: The list of variables from step 1, the high-level hypotheses from step 1, and a set of related experiments that include raw data

Steps:

1) Select data sets, from the related experiments in the input, to be used to refine the variables and the hypotheses. In order to use data from a study it must meet the following qualifications:
    a. Researcher must have access to raw data including data related to the identified variables and data related to the outcome of the use of the chosen process.
    b. There must not be any serious threats to the validity of the study that would call into question the validity of the results.
2) Map metrics collected in each study to the variables that have been defined and map the values of the metrics to the categories chosen to group the subjects for analysis. For example, in this work I mapped the values of each metric to either "high experience" or "low experience". This mapping should be defined in such a way that all studies can be mapped consistently. For instance, subjects with industrial experience are always mapped to the "high experience" category while subjects without industrial experience are always mapped to the "low experience" category.
    a. In the process of mapping each metric to a variable, there may be some metrics collected in a particular study that do not map to any of the identified variables. In this situation, the researchers who ran that study have identified a variable that was not on the initial list. This new variable should be added to the list of variables. For example, in this work, I added the variable of "Working Language Experience" while mapping the metrics from historical studies to the variables on my initial list.
3) Analyze the data, based on the above mapping to determine if each metric has an effect on the outcome of the process. Running a t-test or ANOVA can show the statistical level of the effect between the groups. The goal of this analysis is not to obtain a statistically significant difference between the groups; rather it is to look for trends in the data so that "grounded" hypotheses can be generated.
4) Generate "grounded" hypotheses based on the data analysis. To generate the hypotheses, data from all available studies, which were analyzed separately in step 3, should be grouped together to generate the hypotheses. Because data from studies may become available at different times, this step may be repeated with each new set of data that becomes available. Each new hypothesis is assigned a number based on the high-level hypothesis that it is related to. For example, hypothesis related to '*H.DK*', would be numbered '*H.DK.1*', '*H.DK.2*', and so on.

a. The first data set allows the researcher to create the hypotheses. For each metric collected, a hypothesis should be generated that accurately describes all of the data that is available at the time. For example:

$$(\mathbf{x_i}) \vee (\mathbf{x_k}) \Rightarrow (\mathbf{y_i})$$

b. In each subsequent data set, if the initial hypotheses generated in the previous step, do not accurately describe the new data, then those hypotheses must be modified to describe all of the data. The goal of this refinement is that, after adding constraints, each hypothesis accurately describes the data present. For example:

$$(\mathbf{x_i}) \wedge (\mathbf{x_j}) \vee (\mathbf{x_k}) \Rightarrow (\mathbf{y_i})$$

c. Constraints are defined as variables that are assigned specific values in order to constrain the applicability of the hypothesis.

    i. Add constraints. When the data shows a relationship between the metric and the outcome of the process in some, but not all cases, then the hypothesis must be refined to describe this situation. Keeping in mind the identified variables, the results of the data analysis are examined to characterize those cases where the relationship between the metric and the outcome of the process is not consistent with the rest of the data. If those cases can be characterized in terms of one of the identified variables, then a constraint should be placed on the hypothesis that limits the applicability of that hypothesis. In this work constraints were added that constrained the processes that the hypothesis applied to, the artifacts that the hypothesis applied to, or the pool of people that the hypothesis applied to. The constrained hypothesis will take the form:

    $$(\mathbf{x_i}) \wedge (\mathbf{x_j}) \wedge (\mathbf{x_l} = [\textit{\textbf{one or more specific values}}]) \Rightarrow (\mathbf{y_i})$$

    ii. Remove constraints. A constraint is removed from a hypothesis if there is a constraint present that upon analysis of new data no longer is required in order to accurately describe the data. The unconstrained hypothesis will take the form:

    $$(\mathbf{x_i}) \wedge (\mathbf{x_j}) \wedge (\mathbf{x_l} = [\textit{\textbf{all potential values}}]) \Rightarrow (\mathbf{y_i})$$

    iii. Number the refined hypothesis. As the hypotheses are refined, they should be given a new number to reflect the refinement. If a constraint is added then '*AC*' should be appended to the number of the unrefined hypothesis. If a constraint is removed, then '*RC*' should be appended to the number of the unrefined constraint. For example, if *H.DK.1* is refined by adding a constraint, the number of the new hypothesis would be *H.DK.1.AC*. On the other hand if *H.DK.1* is refined by removing a constraint, the new number would be *H.DK.1.RC*.

Output: Refined list of "grounded" hypotheses with specific metrics and constraints

Designing New Studies

Input: List of "grounded" hypotheses

Steps:

1) Choose one or more hypotheses to study further. It is assumed that the chosen hypotheses are of value and interest. The selection of hypotheses to study further should be based on the following criteria:

    a. The hypotheses are *stabilized.* For a hypothesis to be stabilized **one** of the following must be true:
        i. Data from multiple data sets support that hypothesis.
        ii. Each new data set requires a constraint to be added, such that the hypothesis is continually becoming more specific.
        iii. Each new data set requires a constraint to be removed, such that the hypothesis is continually becoming broader and more general.
    b. The hypotheses satisfy the external constraints placed on the study, such as consistency with course material or relationship to a real problem identified in an industrial setting.
    c. If more than one hypothesis is chosen, the hypothesis must be related, such that it is possible to design a study to test the multiple hypotheses without the assumptions of one hypothesis violating the assumptions of another hypothesis.

2) Design and run a study to test the chosen hypotheses.
3) Analyze the data from the study to determine if there is statistically significant support for the "grounded" hypotheses.
    a. A hypothesis is supported by a statistically significant result, then the researcher can be confident that they have accurately described the relationship between a context variable and the outcome of a process.
    b. If the results of the study do not provide statistically significant support for a hypothesis, then the data from that study is treated like the historical data and used to refine the hypotheses by adding or removing constraints so that the refined hypotheses accurately describe all of the data.

Output:

1) Set of hypotheses supported by statistically significant results.
2) Set of refined hypotheses based on the data from this study.

Promoting for external replication

When one of the following circumstances occurs, a hypothesis is ready for external replication:

1) Statistically significant support for the hypothesis has been obtained from a newly designed and run study. Once a hypothesis has been identified and refined through historical data, and then supported by statistically significant results from a new study, then studies on this hypothesis should be replicated by other researchers in other environments to continue understand all of the potential variables.
2) The hypothesis is refined each time a new data set is analyzed and there is no consistency as to whether constraints are added or removed, meaning that sometimes constraints are added and other times constraints are removed. In this case, researchers in different environments may find the hypothesis useful and worth studying, but no further study should be done in the current environment.

7.2 Complete list of hypotheses

The table that follows in the next few pages presents the complete list of hypotheses that have been generated and refined in this work. The table below contains the following columns:

Hypothesis number – the number corresponding to the number used in the text
Variable – the background or experience variable the hypothesis is related to
Sub-variable – breaks the larger variables down into smaller sets of variables
Metric – the specific metric that the hypothesis is relating to effectivness
R/D – the hypothesis applies to a R(equirements) or D(esign) inspection
↑,↓*, or* ↕ - the metric increases (↑) the effectiveness, decreases (↓) the effectiveness or has no effect (↕) on the effectiveness.
Constraint – any constraint that was included on the hypothesis. If the hypothesis was refined by removing a constraint, then the constraint appears in parenthesis.

Hypotheses that were refined by adding or removing constraints are shaded with details of the refinement appearing on the next line.

Hypothesis number	Variable	Sub-Variable	Metric	R/ D	↑,↓, or ↕	Constraint
H.SD.1.Req	Development Exp.	General S/W Development	Development Experience	R	↑	
H.SD.1.Des	Development Exp.	General S/W Development	Development Experience	D	↑	
H.SD.2.Req	Development Exp.	General S/W Development	Experience as a Developer	R	↑	
H.SD.RE.1.Req	Development Exp.	Requirements Experience	Exp. working with Req.	R	↑	
H.SD.RE.2.Req	Development Exp.	Requirements Experience	Experience using Requirements	R	↑	
H.SD.RE.3.Req	Development Exp.	Requirements Experience	Exp. writing Requirements	R	↑	When using an ad hoc technique
H.SD.RE.3.Req.RCT					↑	(When using a non-procedural or ad hoc technique)
H.SD.RE.4.Req	Development Exp.	Requirements Experience	General Use Case Experience	R	↑	When using PBR
H.SD.RE.5.Req	Development Exp.	Requirements Experience	Experience writing Use Cases	R	↑	When using an ad hoc technique
H.SD.RE.5.Req.ACT					↑	When using an ad hoc technique which is not a well defined, procedural checklist
H.SD.RE.6.Req	Development Exp.	Requirements Experience	Use Case Notation Experience	R	↑	When using an inspection technique that relies on creating Use Cases
H.SD.DE.1.Req	Development Exp.	Design Experience	General S/W Design Exp.	R	↑	
H.SD.DE.1.Req.ACT					↑	Except when using a well-defined, procedural checklist

H.SD.DE.2.Des	Development Exp.	Design Experience	Exp. in Design method	D	↑	
H.SD.DE.3.Des	Development Exp.	Design Experience	Exp. in alternate Design method	D	↓	
H.SD.DE.4.Req	Development Exp.	Design Experience	Exp. creating Designs from Req.	R	↕	When using a checklist or non-design based PBR
H.SD.DE.4.Req.RCT					↕	(when using a checklist or non-design based PBR)
H.SD.DE.4.Des	Development Exp.	Design Experience	Experience creating Designs from Req.	D	↕	
H.SD.DE.5.Req	Development Exp.	Design Experience	Exp. creating Structured Designs	R	↑	When using a technique that relies on design knowledge
H.SD.DE.6.Des	Development Exp.	Design Experience	Exp. creating OO Designs	D	↑	
H.SD.DE.7.Des	Development Exp.	Design Experience	Exp. in design notation	D	↑	
H.SD.TE.1.Req	Development Exp.	Testing Experience	General exp. as a Tester	R	↑	When using an ad hoc or PBR technique
H.SD.TE.1.Req.RCT					↑	When using an ad hoc, procedurally-based checklist, or PBR
H.SD.TE.2.Req	Development Exp.	Testing Experience	Functional Testing Exp.	R	↑	When using PBR tester perspective
H.SD.TE.3.Req	Development Exp.	Testing Experience	Exp. in testing based on Req.	R	↑	When using a non-procedural checklist
H.SD.TE.3.Req.RCT					↑	When using a non-procedural or procedural checklist

H.SD.TE.4.Req	Development Exp.	Testing Experience	Exp. in testing based on Req.	R	↕	When using the PBR user perspective
H.SD.TE.5.Req	Development Exp.	Testing Experience	Exp. in Equivalence Partition Testing	R	↕	
H.SD.PE.1.Req	Development Exp.	Perspective Experience	Perspective Experience	R	↑	When using PBR
H.DK.1.Req	Domain Knowledge	N/A	Domain Knowledge	R	↑	
H.DK.1.Req.ACT					↑	When using non-procedural or ad hoc technique
H.DK.1.Des	Domain Knowledge	N/A	Domain Knowledge	D	↓	
H.PE.1.Req	Process Experience	N/A	Comfort with reviewing req.	R	↑	When using an ad hoc procedure
H.PE.1.Req.RCT					↑	When using an ad hoc or procedurally-based checklist
H.PE.2.Req	Process Experience	N/A	Comfort with reviewing req.	R	↕	When using PBR
H.PE.3.Req	Process Experience	N/A	Comfort with reviewing req.	R	↕	
H.PE.4.Req	Process Experience	N/A			↕	
H.PE.5.Req	Process Experience	N/A	Exp. reviewing req.	R	↑	When inspecting a requirements document is unfamiliar or from unfamilar domain
H.PE.5.Req.ACI					↑	For inspectors who are familiar with the working language of the artifact
H.PE.6.Req	Process Experience	N/A	Exp. reviewing req.	R	↕	When inspecting a requirements document from a familiar domain

H.PE.7.Des	Process Experience	N/A	Exp. reviewing OO designs	D	↑	
H.PE.8.Req	Process Experience	N/A	Exp. in Inspections	R	↕	
H.PE.8.Des	Process Experience	N/A	Exp. in Inspections	D	↕	
H.PE.9.Req	Process Experience	N/A	Software Inspection Exp.	R	↑	
H.PE.9.Des	Process Experience	N/A	Software Inspection Exp.	D	↑	
H.PE.10.Req	Process Experience	N/A	Experience using PBR	R	↑	
H.PE.11.Req	Process Experience	N/A	Observation		↑	
H.PE.12.Req	Process Experience	N/A	Observation of PBR	R	↑	
H.PS.1.Req.ACA					↑	When inspecting requirements document of sufficient complexity
H.PS.2.Req	Process Specificity	N/A	Lack of exp. in PBR perspective	R	↑	
H.PS.2.Req.ACA					↑	When inspecting requirements document of sufficient complexity
H.PS.3.Req	Process Specificity	N/A	Exp. in PBR perspective	R	↕	Technique with too much detail
H.T.1	Training	N/A	Proper Training		↑	
H.T.2	Training	N/A	Laboratory Time		↑	When a new technique is used
H.M.1	Motivation	N/A			↑	
H.WL.1.Req	Working Language	N/A	Native Speaker	R	↑	When the application domain is not familiar

H.WL.1.Des	Natural Language	N/A	Native Speaker	D	↑	When the application domain is not familiar
H.WL.2.Req	Natural Language	N/A	Reading Comprehension	R	↑	For inspectors who are not native speakers of the language
H.WL.2.Req.RCI					↑	For inspectors who have worked in the language
H.WL.2.Des	Natural Language	N/A	Reading Comprehension	D	↑	For inspectors who are not native speakers of the language
H.WL.3.Req	Natural Language	N/A	Listening and Speaking	R	↑	For inspectors who are not native speakers of the language
H.WL.3.Des	Natural Language	N/A	Listening and Speaking	D	↓	For inspectors who are not native speakers of the language

Figure 9 – Complete List of Hypotheses

8. Conclusion

In this dissertation I have discussed the process of developing and refining a set of hypotheses about the impact an inspector's background knowledge and previous experiences can have on the number of defects they find during an inspection. I have used a grounded theory approach to develop and refine these hypotheses. The first iteration was to examine literature both from the software engineering community as well as from the educational and psychology community. After developing an initial list of variables thought to have relevance to software Inspections, based on the literature, each of those variables was refined into a set of metrics that could be collected and examined in specific inspection related studies. A set of studies that had been conducted previously were reanalyzed to form a series of hypotheses about each of the variables and metrics that had been developed. The next iteration of grounded theory was done by using data from a study conducted in Brazil. The hypotheses were either reinforced or refined based on the data from this study. Finally, two studies were designed and run to examine specific variables and hypotheses. The results of these two studies allowed for further refinement of the hypotheses.

The next section summarizes my contributions in terms of the methodology, contributions for researchers and contributions for practitioners. Section 2 summarizes the contributions and limitations of this work. Finally, Section 3 discusses future work, including ideas for more analysis on the existing data, as well as planning for future studies to continue the examination of the variables and hypotheses.

8.1 Contributions

This section discusses my contributions in terms of grounded theory and its application in this context. Then I will discuss my contributions for the researcher and to the practitioner.

8.1.1 Grounded Theory

I showed that a grounded theory approach can be used to develop and refine the hypotheses in this work. Because the field of software engineering is relatively new and immature, very few "truths" are known. This fact made the use of grounded theory particularly useful. The literature base gave me a solid foundation to build on. But, because of all of the unknown issues in the software engineering field, the flexibility to use various types of data to test and refine hypotheses allowed me to finish with a better, more supported, set of hypotheses. By continuing to test and refine these hypotheses in different settings and with different subjects, additional well-supported theories about software inspections will emerge.

To use grounded theory in software engineering, the basic process as used in the sociology field had to be slightly modified. Beginning with background research to establish an initial set of hypotheses is a fairly standard approach. In this case, I not only consulted the software engineering literature, but also sought out relevant literature from other fields such as sociology, psychology and education. This extra literature search allowed for a broader base on which to build.

In terms of refining the hypotheses, I was able to use various types of data from various sources for this step. One addition that I made was a classification scheme for the changes to the hypotheses. I have been able to classify my changes as either adding or removing constraints about the process, inspector, or artifact. By adding this classification scheme, I can examine the history of changes that were made to a hypothesis and decide if the hypothesis seems to be converging on a “truth” or not. As each new study is run, or data set is analyzed, I can look to see if the existing hypotheses are supported as they are, or if constraints are added and removed. If each new study adds or removes a constraint, then we are not yet converging on the “truth”. But, if multiple studies or data sets all support the same hypothesis, or move the hypothesis in the same direction (i.e. constraints are always added, or constraints are always removed), then we are converging towards the “truth”.

To support this discussion of changes, I have also added a predicate calculus notation for the hypotheses. This notation has allowed me to encode both the variable being measured, as well as any constraints that have appeared throughout the process. In the next section, I will examine this collection of predicate calculus statements to abstract up the results.

Another addition that I made was the clustering of the hypotheses. By initially defining a series of high-level, abstract variables, I was able to group each new hypothesis under the appropriate category. This grouping makes it easier to understand how the hypotheses fit together and the location of the potential interactions.

Weaknesses of Grounded Theory

While the grounded theory methodology was found to be useful, there were some weaknesses. One major weakness is that because conclusions are fit to the data present, the conclusions that are drawn are heavily dependant on that data. This dependence raises a few potential issues. The first issue is that the conclusions drawn can be incorrect if the data used to draw those conclusions is not a representative set of data. Secondly, if there is not enough theoretical input to the process or if the overriding assumptions the researcher makes about the environment prove to be incorrect, then the conclusions that are drawn may accurately account for the data that is present, but may still incorrectly describe the phenomena being studied. In order to mitigate this problem, the researcher should include as much theory and literature survey as possible when forming their initial assumptions.

Another weakness of grounded theory is that the researcher does not always have a clear idea of when to stop the process. Because hypotheses and theories evolve based on new data that is analyzed, the potential exists that convergence on a final answer may never happen. In practice the stopping point often becomes the point where there is no more data to analyze. In order to mitigate this weakness, a researcher should define the stopping criteria, *a priori,* relevant to the domain being studied. For example, in continuing this work, we could choose our stopping criteria to be the case where a hypotheses (including any constraints) is supported, as is with no modifications, by one or more statistically significant results.

8.1.2 For the Researcher

The contribution here is a series of variables and associated hypotheses were initially discovered in the literature and by analysis of historical data, and then refined based on data from later studies. The complete list of variables and hypotheses appears at the end of Chapter 4. In this section, I will discuss how the hypotheses relate to each other and how to abstract them up to get more concrete results.

8.1.2.1 Refinement of Grounded Theory Methodology

The implementation and tailoring of the grounded theory research approach that I used in this work can be followed in order to do similar research into the relationship between a process and its context. This section provides details about my methodology so that it can be reused. The three steps involved in the methodology were: 1) Defining an initial list of variables; 2) Refining the list of variables by making them more concrete and adding constraints; and 3) Running new studies. More details about each of these steps follow.

Defining an initial list of variables

To obtain an understanding of previous work and provide a solid basis upon which to build the research program, the research should begin in the literature. The first set of literature that should be reviewed is that literature describing studies that specifically address the context issues of the process being studied. Secondly literature should be reviewed that describes studies conducted on the process being studied to determine what insights other researchers have gained on the issue currently being studied. It is important to note that even if the context variables were not studied specifically, researchers often comment on context variable while either describing the design of their studies or while explaining their results. Finally, if the process being studied can logically be related to a more general, non-software process, then literature from related domains such as education and psychology should be reviewed to gather insights about processes in general. Based on this review of the literature, an initial list of, potentially abstract, variables about the context of the process can be created.

Refining the Variables

The goal of this step is to make the general, abstract variables more concrete by defining specific metrics. When working with historical data, the metrics that have already been collected can be mapped to a related variable from the initial list. If there are specific metrics defined in a historical data set that do not map directly to an abstract variable on the list, then a new variable should be added to the list. Once this mapping has been done, analysis of the metrics can begin.

For each metric, the effectiveness of the subjects is grouped based on the value of that metric and analyzed to determine if the value of the metric has any effect on the effectiveness of the subjects. The goal of this process is not to obtain statistically significant results, rather it is to look for trends in the data that allow "grounded" hypotheses to be generated.

In the process of doing this analysis, hypotheses are generated so that they accurately describe the data. In the first set of historical data, the hypotheses are created. In each subsequent data set, those hypotheses are either supported without change, modified to take into account the new data, or new hypotheses are created. The refinement is done by grouping the data in same way as for the initial set of data. When the new data is analyzed, if the existing hypothesis does not accurately describe the data, then that hypothesis must be refined.

This refinement is done by adding or removing constraints. When the data shows a relationship between the metric and the effectiveness in some, but not all cases, then the hypothesis must be refined to describe this situation. Constraints may be added that constrain the processes that the hypothesis applies to, the artifacts that the hypothesis applies to, or the pool of people that the hypothesis applies to. The goal of this refinement is that, after adding constraints, each hypothesis accurately describes the data present. Similarly, a constraint is removed from a hypothesis if there is a constraint present that upon analysis of new data no longer is required in order to accurately describe the data.

Designing New Studies

The final step in the process is to design new studies to specifically test one or more of the "grounded" hypotheses. Once a hypothesis has stabilized, then that hypothesis has enough support to warrant a full study. For a hypothesis to stabilize, it either must be supported by data from multiple historical studies, or it must have either a constraint added each time that it is refined, such that it is becoming more specific, or it must have a constraint removed each time it is refined, such that it is becoming more general.

The goal of a newly designed study is to gain a statistically significant result to support the "grounded" hypotheses. When a new study is run, and the hypotheses are supported by a statistically significant result, then the researcher can be more confident that they have accurately described the relationship between a context variable and effectiveness of the process user. If the results of the study do not provide statistically significant support for hypotheses, then the data from that study is treated like the historical data and used to refine the hypotheses by adding or removing constraints so that the refined hypotheses accurately describe all of the data.

Deciding when to stop

The refinement and evaluation of a hypothesis should continue until statistically significant support for the hypothesis has been obtained. Once a hypothesis has been identified and refined through historical data, and then supported by statistically significant results from a new study, then this hypothesis can be accepted as an accurate description of the relationship between a context variable and the effectiveness of a process. On the other hand, some hypotheses are refined each time a new data set is analyzed. If a situation occurs such that there is no consistency on the adding or removing of constraints to a hypothesis, meaning that sometimes constraints are added and other times constraints are removed, then we can discard the hypothesis with the assumption that it was not accurately describing a real relationship. But, if a hypothesis is always being refined to be more constrained or always being refined to be less constrained, then the process of refining and testing the hypothesis should continue until statistical significance is reached.

8.1.2.2 New Variables Discovered

The first thing that I learned about the variables was the discovery of new variables from the data that did not appear in the literature search. Looking at the historical data two new variables appeared, working language experience and process specificity. The grounded theory methodology allowed me to add these new variables to the initial list defined from the literature.

8.1.2.3 Details of hypotheses that were either changed or supported by the data

This section groups the hypotheses by the types of refinements that were made to them. These refinements include: 1) no refinement; 2) the addition of a constraint; and 3) the removal of a constraint.

Hypotheses supported

The following hypotheses were all supported identified based on the historical data and then subsequently supported by the data from the replications in Brazil.

H.SD.2.Req

(Experience as Developer) ∧ (Requirements inspection) ⇒ **(more defects found)**

H.SD.TE.4.Req

(Experience testing from requirements) ∧ (Requirements inspection) ∧ (PBR) ¬⇒ **(more defects found)**

H.SD.PE.1

(Experience in perspective) ∧ (Requirements inspection) ∧ (PBR) ⇒ **(more defects found)**

H.PE.8

(Experience in performing inspections) ∧ (Requirements inspection) ∧ (detailed techniques) ¬⇒ **(more defects found)**

H.PE.9

(Inspection experience) ∧ (Requirements inspection) ∧ (same style of inspection) ⇒ **(more defects found)**

Constraints Added

The next set of hypotheses all had a constraint added to the initial version of the hypothesis. Because data from a subsequent study showed a result that was more narrowly applicable than the initial hypothesis, we cannot say that the initial unconstrained hypothesis supports the more constrained version. Therefore, each of the following hypotheses has support from one study.

In the first group of hypotheses, the initial hypothesis has support from the historical data, while the constrained hypothesis has support from the replications in Brazil.

H.DK.1.Req

(Domain Knowledge) ∧ (Requirements inspection) ⇒ **(more defects found)**

Added constraint on technique

(Domain Knowledge) ∧ (Requirements inspection) ∧ (non-procedural technique ∨ ad hoc technique) ⇒ **(more defects found)**

H.SD.RE.5.Req

(Experience writing Use Cases) $\wedge$ (Requirements inspection) $\wedge$ (ad hoc technique) $\Rightarrow$ **(more defects found)**

Added constraint on technique

(Experience writing Use Cases) $\wedge$ (Requirements inspection) $\wedge$ (ad hoc technique) $\wedge$ $\neg$(well-defined, procedural checklist) $\Rightarrow$ **(more defects found)**

H.SD.DE.1.Req

(Software Design Experience) $\wedge$ (Requirements inspection) $\Rightarrow$ **(more defects found)**

Added constraint on technique

(Software Design experience) $\wedge$ (Requirements inspection) $\wedge$ $\neg$(well-defined, procedural checklist) $\Rightarrow$ **(more defects found)**

H.PE.5.Req

(Experience reviewing requirements) $\wedge$ (Requirements inspection) $\wedge$ (document unfamiliar $\vee$ domain unfamiliar) $\Rightarrow$ **(more defects found)**

Added constraint on inspector pool

(Experience reviewing requirements) $\wedge$ (Requirements inspection) $\wedge$ (inspectors familiar with working language) $\wedge$ (document unfamiliar $\vee$ domain unfamiliar) $\Rightarrow$ **(more defects found)**

In the second group of hypotheses, the initial hypothesis has support from the replications in Brazil, while the constrained hypothesis has support from the study described in Section 6.2.

H.PS.1

(Experience in perspective) $\wedge$ (Requirements inspection) $\wedge$ (Low detail technique) $\Rightarrow$ **(more defects found)**

Added constraint on artifact

(Experience in perspective) $\wedge$ (Requirements inspection) $\wedge$ (low detail technique) $\wedge$ (artifact of sufficient complexity) $\Rightarrow$ **(more defects found)**

H.PS.2

¬(Experience in perspective) ∧ (Requirements inspection) ∧ (step-by-step technique) ⇒ (more defects found)

Added constraint on artifact

¬(Experience in perspective) ∧ (Requirements inspection) ∧ (step-by-step technique) ∧ (artifact of sufficient complexity) ⇒ (more defects found)

Constraints Removed

In the following group of hypotheses data from the replications in Brazil allowed me to remove a constraint from the hypotheses. Because the refined hypotheses are more general than the hypothesis generated from the historical data, the data from Brazil as well as the historical data both support the refined version of the hypotheses.

H.SD.RE.3.Req

(Experience writing Requirements) ∧ (Requirements Inspection) ∧ (non-procedural ∨ ad hoc technique) ⇒ (more defects found)

Removed constraint on technique

(Experience writing Requirements) ∧ (Requirements Inspection) ⇒ (more defects found)

H.SD.DE.4.Req

(Experience creating designs from requirements) ∧ (Requirements inspection) ∧ (checklist ∨ non-design based PBR) ¬⇒ (more defects found)

Removed constraint on technique

(Experience creating designs from requirements) ∧ (Requirements inspection) ¬⇒ (more defects found)

H.SD.TE.1.Req

(Experience as a tester) ∧ (Requirements inspection) ∧ (Ad hoc ∨ PBR technique) ⇒ (more defects found)

Removed constraint on technique

(Experience as tester) ∧ (Requirements inspection) ∧ (Ad hoc ∨ procedurally-based checklist ∨ PBR) ⇒ (more defects found)

H.SD.TE.3.Req

(Experience testing based on requirements) $\wedge$ (Requirements inspection) $\wedge$ (non-procedural checklist) $\Rightarrow$ **(more defects found)**

Removed constraint on technique

(Experience testing based on requirements) $\wedge$ (Requirements inspection) $\wedge$ (non-procedural $\vee$ procedural checklist) $\Rightarrow$ **(more defects found)**

H.PE.1.Req

(Comfort reviewing requirements) $\wedge$ (Requirements inspection) $\wedge$ (ad hoc procedure) $\Rightarrow$ **(more defects found)**

Removed constraint on technique $\Rightarrow$ **(more defects found)**

(Comfort reviewing requirements) $\wedge$ (Requirements inspection) $\wedge$ (ad hoc $\vee$ procedurally based checklist)

H.WL.2.Req

(Comfort level in reading comprehension) $\wedge$ $\neg$(Native speaker of working language) $\wedge$ (Requirements inspection) $\Rightarrow$ **(more defects found)**

Removed constraint on inspector pool

(Comfort level in reading comprehension) $\wedge$ (Experience working in the language) $\wedge$ $\neg$(Native speaker of working language) $\wedge$ (Requirements inspection) $\Rightarrow$ **(more defects found)**

Based on the results above the hypotheses were evolved as follows:

- Supported (i.e. no change) – 5 hypotheses
- Added Constraint on Technique – 3 hypotheses
- Added Constraint on Inspectors – 1 hypothesis
- Added Constraint on Artifact – 2 hypotheses
- Removed Constraint on Technique – 5 hypotheses
- Removed Constraint on Inspector – 1 hypothesis
- Removed Constraint on Artifact – 0 hypotheses

These results show that using this methodology I was able to arrive at some hypotheses that were supported with no changes. In addition, the methodology allowed me to add and removed constraints based on the data. The above results show the number of hypotheses that had a constraint added is similar to the number that had a constraint removed. These results show that the hypotheses can be evolved to explain the data that is present.

8.1.3 For the Practitioner

This work also contributes a series of conclusions that are useful for the practitioner community. First of all, the results have shown that testing experience, one of the specific types of experience studied, is important for inspectors to possess. This result makes sense because testing is a process that requires some analysis skills. Inspections, while not requiring the same analysis skills as testing, also require analysis of an artifact to look for defects.

A second set of results apply to techniques that are used in an inspection. Data from the studies has shown that the presence of an inspection technique often neutralizes the effect of different kinds of experience. It is not clear whether the techniques raise the performance of less experienced inspectors, or lower the performance of more experience inspectors, or some combination of both. Another interesting issue that arose concerning inspection techniques is that the amount of detail present in the technique must match up with the experience level of the inspector who is using that technique

Finally, the results have provided some insight into the training process. Based on the study discussed in Chapter 6, it does not appear that observation is an effective way to train new inspectors. Observation does help an inspector's familiarity with the process, but it does not provide enough experience to increase an inspector's performance. It seems that an inspector must actually perform an inspection in order to gain useful experience.

8.2 Summary

The series of hypotheses generated by this work provide a good starting point for the continued investigation of the relationship between and inspectors' background knowledge and past experiences and his or her performance during an inspection.

For the researcher, this work provides a framework to use when investigating the relationship between experience and process. The hypotheses that have been generated are based on both theory and observed data and therefore should be useful in continuing to understand software inspections. These hypotheses allow a researcher to choose one or more that are of interest in their particular setting and run a study to better understand them. Those results can then be combined with the previous knowledge to continually build a growing knowledge base on software inspections.

For example, the output of the study described in Section 6.1 suggested that in order for inspectors to gain process experience, they had to actually perform the process. A researcher can take this hypothesis and design an experiment to test it. In such an experiment, each subject would perform two (or more) inspections using the same technique. The results of the second inspection could then be compared to the results of the first inspection to determine if there was any improvement. This design has the benefit of having the same set of subjects in both treatment groups, so the statistical analysis would be stronger. By selecting two artifacts of similar difficulty, length, and defect profile, and providing document A to half of the subjects first and document B to the other half first, the effects of the document can be minimized in the analysis.

For the practitioner, this work provides a series of characteristics or traits that may be important when planning an inspection process. While all of the hypotheses still require future testing to ensure their validity, they do provide some evidence of the characteristics that may or may not be important when choosing inspectors. This guidance can help a practitioner when he or she is building an inspection team by either pointing out the best candidates for the team, or by identifying the areas in which training may be necessary for members of the team who are deficient in those areas.

The limitations of this work include the following. Much of the results were based on data drawn from studies with students rather than professionals. Additionally, the data in many of the studies was collected for another purpose and reanalyzed for this work. Other limitations were mentioned throughout the text concerning issues of confounding variables. Another limitation of this work is that the procedure for deciding when to add or remove constraints contained some subjectivity. Finally, much of the defect data collected was based on defects that were seeded in artifacts by researchers, rather then being naturally occurring defects. The nature of the defects limits the applicability of these results to the extent that the seeded defects are representative of naturally occurring defects.

8.3 Future Work

As a starting point Studies 1 & 2 from the previous chapter could be redesigned in order to address some of the flaws in the original studies. Study 1 could be redesigned to test the new hypotheses:

(Experience performing an inspection) ⇒ (more defects found)

and

¬(Experience performing an inspection) ⇒ ¬(more defects found)

Each subject would be allowed to perform two inspections, rather than only observing the first inspection. This study would require two requirements documents of relatively equal complexity. The ordering of the documents should be randomized in order to eliminate the document effect as much as possible. After performing both inspections, data could be analyzed in the following manner. First, performance on the second inspection could be compared with performance on the first inspection for each individual inspector to determine how many inspectors improved and how many did not. Secondly, the average of the first inspection could be compared with the average of the second inspection to understand if there was an overall improvement from the first inspection to the second inspection.

On the other hand, in order to make Study 2 more effective, it should be run in an industrial rather than classroom setting. We can hypothesize that classroom experience is not enough experience to qualify an inspector as experienced in their particular perspective. This study would be designed such that subjects from industry would be given either a technique containing a high amount of detail or a technique containing a low amount of detail. The performance of the subjects can then be compared to determine if the level of detail has any effect on the inspectors' effectiveness. The hypothesis tested would be:

(Experience in perspective) ∧ (Req. Inspection) ∧ (Low detail technique) ⇒ (more defects found)

and

(Experience in perspective) ∧ (Req. Inspection) ∧ (High detail technique) ⇒ ¬(more defects found)

As stated previously, this work forms a framework for the ongoing investigation of the variables and hypotheses identified. Studies can be designed and run to test each individual hypothesis. As studies are run, the results can then be compared against the current hypothesis, and continuing to use a grounded theory approach, these hypotheses can be refined if necessary.

Because there were multiple results suggesting the importance of testing experience, this variable would be a good place to continue running studies. We could design a study to test the hypothesis:

(Experience as a tester) $\Rightarrow$ (more defects found)

In this study, instead of asking the subjects to self-report their experience, a pre-test should be created to evaluate their testing knowledge. The subjects could then be placed into the high experience group or low experience group based on this test. The subjects then would all receive the same training and the same requirements artifact. Data analysis could be done between the two groups to determine if those subjects who scored higher on the pre-test found more defects than the subjects who scored lower on the pre-test.

The data presented in this work was analyzed such that all defects were treated equally. In addition to designing and running the new studies described above, more analysis can be done on the existing data to look at classes of defects, to begin to understand if there is a relationship between any of the identified background knowledge or previous experience of an inspector and his or her tendency to find that particular class of defect.

This study of individual defect types could take on many forms. One interesting classification of defects would be done based on severity. Because severe defects either are more expensive to fix or cause more catastrophic consequences if not fixed, it is more important to find these defects. After classifying each defect based on its severity, the data can be reanalyzed to determine if any of the above variables have an impact on the number of severe defects found.

In addition, we could begin to look at the interaction between specific variables and defect classes. For instance, the data from this work indicated that application domain knowledge was important for an inspector to have during a requirements inspection. Because application domain knowledge would tend to help an inspector understand the types of information that had been left out of a requirements document, we could reanalyze the existing data to examine the more specific hypothesis:

(Application Domain Knowledge) $\wedge$ (Req. Inspection) $\Rightarrow$
(more Omission Defects found)

There are still some questions related to the interaction between an inspector's background and the types of defects that he or she is most likely to find. For example, the native language of an inspector showed us as an interesting variable. Some questions worth studying are:

1) Do non-native speakers find different types of defects than native speakers find?
2) If so, how do the defects differ?

Another example would concern the application domain knowledge issue. Because application domain experts found more defects during requirements inspections and application domain novices found more defects during design inspections, questions worth studying would be:

1) What types of requirements defects are missed by application domain novices?
2) What types of design defects are missed by application domain experts?

Similar questions could be posed and investigated for the other variables.

Appendix A – Detailed descriptions of studies

Throughout Chapter 4, the results from a series of experiments are used as evidence for the proposed hypotheses. A brief description of each of these studies was provided in Chapter 4. This appendix will provide a more complete discussion of each one of those studies. Each study will be described in the following manner:

Background – Provides information about why the particular study was run, and any other relevant background information.

Researchers – Describes the researchers who carried out the study.

Subjects – Provides a brief description of the subjects who participated in the study, including such information as number of subjects and whether the subjects were professionals or students, graduates or undergraduates.

Materials – Provides a description of the materials used during the study, including the technology being studies as well as the software artifacts used in the study.

Procedure – Describes the procedures followed by the subjects in carrying out the experiment. This description includes information about the various treatments in the study as well as how subjects were assigned to their particular treatment.

Data Collection – Describes the method(s) with which the data was collected. This section also describes the types of data that were collected.

NASA Studies (1994/1995)

Background

This was a series of two studies conducted separately but using the same materials and design, and drawing subjects from the same subject pool (although the two sets of subjects were distinct.) The goal of these studies was to compare the effectiveness of a perspective-based requirements inspection method to that of the standard method used by NASA software practitioners.

Researchers

Researchers from the University of Maryland conducted this study.

Subjects

Professional software developers at NASA Goddard Space Flight center, Greenbelt, MD.

Materials

A version of Perspective Based Reading (PBR) was studied. In this version of PBR, because the subjects were experience software developers, the amount of detail in the techniques was low. The techniques instructed the subjects which perspective to view the requirements from and a series of questions to consider. PBR was compared against the standard technique for requirements review used in the NASA environment.

PBR and the standard technique were performed on four requirements documents. Two of the requirements documents were generic (non-NASA specific). The first was for a Parking Garage Control System (PG). This system controls the entrances and exits for a parking garage and keeps track of available spaces through the use of daily and monthly tickets (passes). The second generic system was for an Automated Teller Machine (ATM). This system is responsible for the control of an ATM machine. The other two were for domain specific (NASA documents). All four of the documents were seeded with a set of defects by the experimenters.

Procedure

The subjects were given a background survey to collect information about their background and related experience. The subjects were then assigned to one of the generic and one of the NASA specific requirements documents. The assignment was done randomly so that each generic and each domain specific document had approximately half of the subjects inspecting it. After completion of this inspection, the subjects were trained in one of the three perspectives of PBR (Designer, Tester, or User). The subjects then inspected whichever generic and whichever domain specific requirements document they had not inspected in the first inspection.

Data Collection

Quantitative data was collected in the form of a background survey to collect information about the subjects' background and experiences. Defect report forms also were collected which provided information about the defects found in the requirements documents. Qualitative data was collected through post-study surveys and questionnaires.

CMSC735/MSWE609 University of Maryland (Fall 1997)

Background

The goal of this study was to compare the effectiveness of a checklist-based requirements inspection to a perspective-based requirements inspection. Because the techniques were relatively new and had not undergone many studies, the requirements inspection was done in isolation, i.e. not in the context of a project being constructed. As this was an early study, the researchers decided it was best to test the techniques this way rather than risk failure of a real project with untested methods.

Researchers

Researchers at the University of Maryland conducted this study.

Subjects

The subjects were students in a graduate level software engineering class at the University of Maryland. They were drawn from two populations. Some of the students were from the regular Masters/PhD in Computer Science program and some were from the Masters of Software Engineering program.

Materials

A requirements checklist and a more detailed version of the PBR techniques were evaluated. The checklist simply provided the inspectors with a series of things to look for when reviewing the requirements document. The PBR procedures were more detailed than those used in the NASA study described above. This detail was necessary because the subjects were less experienced as software developers. Therefore, the subjects needed more concrete instructions on how to think from their particular perspective.

The checklist and PBR techniques were used on two generic documents. The ATM and PG requirements documents described in the NASA experiment above were again used here. These documents were also seeded with defects by the experimenters.

Procedure

A background questionnaire was given to the subjects to collect the relevant background information and related experience. Subjects were instructed in the use of the checklist technique. Then all subjects were given the ATM requirements document to inspect using the checklist. The subjects reviewed the document and recorded their defects on the defect report forms. After completion of this inspection, the subjects were then trained in the PBR techniques described above. The subjects were then given the PG requirements document to inspect using PBR. The subjects again recorded their defects on the defect report form.

Data Collection

Quantitative data was collected via the background survey concerning the background and experiences of the subjects. The defect report forms were also used for data collection about the defects detected in the requirements documents. Qualitative data was collected via post-study surveys.

CMSC435 University of Maryland (Fall 1998)

Background

This study was conducted with senior level undergraduate students enrolled in a software engineering project course at the University of Maryland. The goal of this study was to again compare the checklist-based requirements review with the PBR requirements review, this time in the context of a project. Subjects received training in either the PBR technique or the Checklist-based technique. The subjects then performed a requirements inspection on the requirements document for the project they were developing. Quantitative data was collected via defect report forms. Qualitative data was collected via post-study questionnaires.

Researchers

Researchers at the University of Maryland conducted this study.

Subjects

The subjects were undergraduate and graduate students enrolled in a senior level undergraduate software engineering course at the University of Maryland. As a part of the course, the subjects were constructed a semester long software project including the requirements, design, code, testing and other related activities.

Materials

A checklist to aid in the inspection of a requirements document, and the more detailed PBR techniques described in CMSC735/MSWE609 study above were evaluated.

Subjects used these two techniques on a requirements document that would later be used to create a design and implementation of a system. The system being developed was the Loan Arranger (LA) system. This system is in the financial domain, and is responsible for organizing and bundling loans for sale to financial institutions. The system has complexities due to non-functional requirements. This requirements document was seeded with defects by the experimenters.

Procedure

Subjects were first given a background questionnaire and survey to collect information about their previous experience with relevant software engineering topics, as well as previous classroom and work background. The subjects were then trained in either the checklist inspection method or the PBR inspection method (each subject was assigned to only one group). After the training, the subjects received a requirements document to inspect. Each subject used whichever technique he or she had been trained in, and recorded all of the defects found while reviewing the requirements document.

Data Collection

Quantitative data was collected via the background questionnaires as well as the defect collection forms. Qualitative data was collected via post-study surveys.

CMSC735 University of Maryland (Fall 1999)

Background

The goal of this study was to understand both PBR and OORTs at a more detailed level than we had in the past. To this end, we employed some new methods for obtaining the level of detail we were interested in. Rather than simply performing and inspection and reporting defects and then answering a post-experiment questionnaire, subjects were paired together, and observed each other during the inspection process. The process executor was instructed to "think-aloud" while performing the inspection, and the observer was instructed to take notes about the process. The idea here was to begin to understand where the process executor ran into problems when using the techniques. In addition, two different problem domains were used during this study. One half of the subjects inspected requirements and design from the same domain, and the other half of the teams inspected requirements and designs from different domains. This allowed us to begin to investigate whether knowledge of the requirements influenced the design inspection process.

Researchers

Researchers at the University of Maryland conducted the study.

Subjects

Students from a graduate-level software engineering class at The University of Maryland.

Materials

The detailed PBR techniques first describe in the CMSC735/MSWE 609 (Fall 1997) study were used here. In addition a new set of object oriented design reading techniques (OORTs) were studied. These techniques provide guidance to a reviewer on how to compare the different UML design artifacts both with each other and with the artifacts from the requirements phase of the lifecycle.

The techniques were applied to artifacts from two domains. The Parking Garage (PG) system and the Loan Arranger (LA) system both described earlier. Both of these requirements documents were seeded with defects by the experimenters.

Procedure

Subjects were grouped into pairs for the purpose of observation. There were two treatments to this study. In first treatment, the subjects were trained in the PBR procedures and one subject used PBR on the assigned set of requirements while his partner observed and took notes about the execution of the technique. The subject who was executing PBR was told to think-aloud so that his partner would be able to take notes. After this inspection was complete, the subjects were then trained in the OORTs. The partners switched roles from the first treatment. The second inspection was then done using the OORTs on the assigned design. Assignment of requirements and designs were done such that roughly half of the teams inspected each requirements document and each design document. Within each of those groups, half of the subjects inspected the requirements and design from the same system and the other half inspected the requirements from one system and the design from the other system.

Data Collection

Quantitative data was collected via defect report forms and background questionnaires completed by each subject. Qualitative data was obtained from reports written by each team based on their observations during the inspections.

University of Southern California (Fall 2000/Spring 2001)

Background

The goal of this study was to determine if the PBR and OORT techniques could be tailored for use in the spiral development model and the MBASE document standard. This was the first time the reading techniques had been used in this type of setting. The differences between the previous settings and this one are described in the Materials section below.

Researchers

Researchers at the University of Maryland in conjunction with researchers from the University of Southern California ran this study.

Subjects

Subjects were graduate students enrolled in a graduate level software engineering course. This course covered two semesters in which the students acquired a real client and created a project from start to finish including elicitation and negotiation of requirements, creation of design, coding, testing, and release planning. Not all students participated in both semesters; many only completed the first semester.

Materials

A tailored version of the PBR and OORT techniques was used here. The techniques had to be tailored based on three major differences in the USC environment. First, the original techniques were designed for the waterfall lifecycle model. The subjects at USC were using the spiral lifecycle model; so some tailoring had to be done to the techniques to account for the level of detail required in each spiral. Secondly, the original techniques were designed with a simple UML-based documentation standard in mind. The subjects at USC used a very detailed documentation guideline called MBASE. Based on the way MBASE organized the artifacts and information, some tailoring had to be done to the techniques. Finally, the original techniques were designed to be used in an inspection where the focus is placed on individual defect detection and the team meeting is merely a formality or even non-existent. The subjects at USC were required to use a Fagan-style inspection where the focus is placed on the team meeting. The techniques had to be tailored to allow them to be used for the individual preparation phase prior to the team meeting.

The artifacts inspected were those created by each team in the process of constructing the software. So, each team had different artifacts, and the defects present were not seeded by the experimenters, rather they were naturally occurring.

Procedure

After completion of the initial requirements documentation, the subjects were then trained in the PBR procedures. Each subject was assigned one of the specific PBR techniques based on which role they were taking during the inspection team meeting. The subjects use the PBR techniques to individually review the document prior to the inspection meeting. During this individual review, each subject was instructed to look for potential defects and record them on a form, which they brought to the team meeting. During the team meeting, when the team decided that something was a defect, the subjects made a notation on their individual forms if that was one of the potential defects they had uncovered.

After the requirements were inspected and corrected, each team then created a design for their system. The subjects were then trained in the OORT techniques. Because each team's design was different and contained different information depending on the domain of their project, the team lead was allowed to choose which of the OORT techniques applied to his or her project. After making this selection, each subject was assigned techniques from the subset applicable to his or her project. Subjects did the preparation and team meeting in the same was as for the requirements inspection stage.

Data Collection

Quantitative data was collected via the individual preparation forms that contained the potential defects as well as the indication of which of those potential defects became real defects during the team meeting. Qualitative data was collected through post-experiment questionnaires and interviews with the subjects.

Appendix B –Data from studies

This appendix contains the analyzed data from each of the studies described in Appendix A. The data is grouped and analyzed based on the Background and Experience Variables defined in Chapter 4. Because each variable is defined by a series of metrics, there is a table of data for each metric. The tables are all in the following format:

Example Table

			Metric			
Experiment	**Artifact**	**Technique**	**Group 1**	**Group 2**	**p-value**	
CMSC 435 Fall 1998	LA	Checklist	10.7% (18)	10.3% (2)	.96	X
	PG	PBR (U)	11% (22)	8.6% (2)	.489	X

Where:

- *Experiment* – The study or experiment from Appendix A in which the data was collected.
- *Artifact* – The artifact that was inspected by the subjects in the treatment group.
  - PG – Parking Garage; (Req) = Requirements; (Des) = Design
  - ATM – Automated Teller Machine
  - LA – Loan Arranger; (Req) = Requirements; (Des) = Design
- *Technique* – The technique used by the subjects in the treatment group.
  - Ad hoc – No specific technique was defined; subjects used any methods they knew.
  - Checklist – A non-procedural list of items to look for in a software artifact during an inspection.
  - NASA Technique – The typical technique used by NASA professionals at Goddard Space Flight Center.
  - PBR – Perspective Based Reading; the letters 'T', 'D', and 'U' indicate which of the PBR perspectives (Tester, Designer, or User) are included in that group of subjects. (NOTE: If only a subset of 'T', 'U', and 'D' appear, then only those perspectives that appear in the table were used in that study.)
  - OORTs – Object Oriented Reading Techniques.
- *Group 1/Group 2* – These two columns represent the average percent of the known defects found by the subjects in each of the two treatments represented by the columns. The number in parenthesis is the number of subjects that were in that group.
- *p-value* – The p-value obtained from running a t-test between the two treatment groups
- *√/X* – A '√' indicates that the more experienced group found more defects than the less experienced group, while an 'X' indicates that the less experience group found more defects than the more experience group.

In each table below, shaded rows indicate that the data in that row is an aggregation of data appearing in other rows.

Domain Knowledge

Table 7 - Domain Knowledge (CMSC735 Fall 1999)

			Domain Knowledge			
Experiment	**Artifact**	**Technique**	**Non-expert**	**Expert**	**p-value**	
735 Fall 1999	LA (Req)	PBR (U&T)	17.4% (8)	--	--	--
	PG (Req)	PBR (U&T)	--	22.9% (6)	--	--
	LA & PG (Req)	PBR (U)	15.7% (6)	14.1% (2)	.841	X
		PBR (T)	22.2% (2)	27.3% (4)	.552	√
		PBR (U&T)	17.4% (8)	22.9% (6)	.325	√
	LA (Des)	OORTs	25.1% (7)	--	--	--
	PG (Des)	OORTs	18.6% (1)	11.6% (6)	.371	X
	LA & PG (Des)	OORTs	24.3% (8)	11.6% (6)	.002	X

Software Development Experience

Table 8 – Software Development Experience (CMSC735 Fall 1999, USC)

			Software Development Experience			
Experiment	**Artifact**	**Technique**	**None/Class**	**Industry**	**p-value**	
CMSC 735 Fall 1999	LA (Req)	PBR (U & T)	17.4% (8)	--	--	--
	PG (Req)	PBR (U & T)	14.1% (2)	27.3% (4)	.164	√
	LA & PG (Req)	PBR (U)	15.3% (8)	--	--	--
		PBR (T)	22.2% (2)	27.3% (4)	.552	√
		PBR (U & T)	16.7% (10)	27.3% (4)	.071	√
USC Fall 2000	--	PBR	23.3% (11)	17.5% (20)	.594	X
USC Spring 2001	--	PBR	20.6% (15)	26% (16)	.649	√
USC Spring 2001	--	OORT	24.9% (12)	26.5% (19)	.822	√

Table 9 –Experience as a Developer (NASA 94)

			Experience as a Developer			
Experiment	**Artifact**	**Technique**	**<= 3 Years**	**>= 5 Years**	**p-value**	
NASA 94	NASA A	NASA Technique	13% (3)	7.4% (3)	.251	X
		PBR	16.7% (2)	18.1% (4)	.879	√
		PBR & NASA	14.4% (5)	13.5% (7)	.851	X
	NASA B	NASA Technique	13.3% (2)	18.3% (4)	.723	√
		PBR	11.1% (3)	6.7% (3)	.643	X
		PBR & NASA	12% (5)	13.5% (7)	.862	√
	PG (Req)	NASA Technique	20.3% (2)	17.2% (4)	.659	X
		PBR	22.9% (3)	22.9% (3)	1	X
		PBR & NASA	21.9% (5)	19.6% (7)	.701	X
	ATM (Req)	NASA Technique	23.8% (3)	25% (3)	.911	√
		PBR	26.8% (2)	37.5% (4)	.342	√
		PBR & NASA	25% (5)	32.1% (7)	.327	√
	NASA (A & B)	NASA Technique	13.1% (5)	13.7% (7)	.935	√
		PBR	13.3% (5)	13.2% (7)	.98	X
		PBR & NASA	13.2% (10)	13.4% (14)	.965	√
	PG & ATM (Req)	NASA Technique	22.4% (5)	20.5% (7)	.749	X
		PBR	24.5% (5)	31.2% (7)	.374	√
		PBR & NASA	23.4% (10)	25.9% (14)	.611	√
	All	NASA Technique	17.8% (10)	17.1% (14)	.882	X
		PBR	18.9% (10)	22.2% (14)	.565	√
		PBR & NASA	18.3% (20)	19.7% (28)	.713	√

Table 10 – Experience as a Developer (NASA 95)

			Experience as a Developer			
Experiment	**Artifact**	**Technique**	**<= 3 Years**	**>= 5 Years**	**p-value**	
NASA 95	NASA A	NASA Technique	34.4% (6)	53.3% (2)	.375	√
		PBR	66.7% (1)	53.3% (5)	.639	√
		PBR & NASA	39% (7)	53.3% (7)	.28	√
	NASA B	NASA Technique	20% (1)	58.7% (5)	.239	√
		PBR	49.3% (5)	43.3% (2)	.847	X
		PBR & NASA	44.4% (6)	54.3% (7)	.567	√
	PG (Req)	NASA Technique	17.9% (1)	36.4% (5)	.418	√
		PBR	31% (6)	44.6% (2)	.161	√
		PBR & NASA	29.1% (7)	38.8% (7)	.216	√
	ATM (Req)	NASA Technique	22.2% (6)	25.9% (2)	.702	√
		PBR	33.3% (1)	35.6% (5)	.886	√
		PBR & NASA	23.8% (7)	32.8% (7)	.184	√
	NASA (A & B)	NASA Technique	32.4% (7)	57.1% (7)	.067	√
		PBR	52.2% (6)	50.5% (7)	.914	X
		PBR & NASA	41.5% (13)	53.8% (14)	.229	√
	PG & ATM (Req)	NASA Technique	21.6% (7)	33.4% (7)	.14	√
		PBR	31.3% (7)	38.2% (7)	.271	√
		PBR & NASA	26.4% (14)	35.8% (14)	.064	√
	All	NASA Technique	27% (14)	45.3% (14)	.027	√
		PBR	41% (13)	44.3% (14)	.698	√
		PBR & NASA	33.7% (27)	44.8% (14)	.063	√

Table 11 – General Requirements Experience (CMSC735 Fall 1997)

			Requirements Experience (General)			
Experiment	**Artifact**	**Technique**	**None/Class**	**Industry**	**p-value**	
735 Fall 1997	ATM (Req)	Ad hoc	31.2% (29)	35.2% (36)	.307	√
	PG (Req)	PBR (U)	28% (13)	34.1% (9)	.327	√
		PBR (D, T)	28.3% (16)	29.6% (28)	.734	√
		PBR (U, D, T)	28.2% (29)	30.7% (37)	.418	√
	ATM & PG (Req)	Both	29.7% (58)	32.9% (73)	.196	√

Table 12 –Experience Using Requirements (NASA 94)

			Experience Using Requirements			
Experiment	**Artifact**	**Technique**	**< 3 Years**	**>= 5 Years**	**p-value**	
NASA 94	NASA A	NASA Technique	13% (3)	7.4% (3)	.251	X
		PBR	11.1% (2)	20.8% (4)	.244	√
		PBR & NASA	12.2% (5)	15.1% (7)	.569	√
	NASA B	NASA Technique	20% (2)	15% (4)	.723	X
		PBR	11.1% (3)	6.7% (3)	.643	X
		PBR & NASA	14.7% (5)	11.4% (7)	.672	X
	PG (Req)	NASA Technique	17.2% (2)	18.8% (4)	.827	√
		PBR	22.9% (3)	22.9% (3)	1	--
		PBR & NASA	20.6% (5)	20.5% (7)	.988	X
	ATM (Req)	NASA Technique	23.8% (3)	25% (3)	.911	√
		PBR	30.4% (2)	35.7% (4)	.651	√
		PBR & NASA	26.4% (5)	31.1% (7)	.526	√
	NASA (A & B)	NASA Technique	15.8% (5)	11.7% (7)	.541	X
		PBR	11.1% (5)	14.8% (7)	.563	√
		PBR & NASA	13.4% (10)	13.3% (14)	.965	X
	PG & ATM (Req)	NASA Technique	21.2% (5)	21.4% (7)	.964	√
		PBR	25.9% (5)	30.2% (7)	.575	√
		PBR & NASA	23.5% (10)	25.8% (14)	.634	√
	All	NASA Technique	18.5% (10)	16.6% (14)	.676	X
		PBR	18.5% (10)	22.5% (14)	.487	√
		PBR & NASA	18.5% (20)	19.5% (28)	.769	√

Table 13 –Experience Using Requirements (NASA 95)

			Experience Using Requirements			
Experiment	**Artifact**	**Technique**	**<= 3 Years**	**>= 5 Years**	**p-value**	
NASA 95	NASA A	NASA Technique	44% (5)	31.1% (3)	.505	X
		PBR	55.6% (3)	55.6% (3)	1	--
		PBR & NASA	48.3% (7)	43.3% (6)	.715	X
	NASA B	NASA Technique	40% (3)	64.4% (3)	.333	√
		PBR	46.7% (5)	50% (2)	.915	√
		PBR & NASA	44.2% (8)	58.7% (5)	.406	√
	PG (Req)	NASA Technique	27.4% (3)	39.3% (3)	.492	√
		PBR	34.3% (5)	34.5% (3)	.98	√
		PBR & NASA	31.7% (8)	36.9% (6)	.521	√
	ATM (Req)	NASA Technique	22.2% (5)	34.7% (3)	.776	√
		PBR	33.3% (3)	37% (3)	.746	√
		PBR & NASA	26.4% (8)	30.9% (6)	.525	√
	NASA (A & B)	NASA Technique	42.5% (8)	47.8% (6)	.718	√
		PBR	50% (8)	53.3% (5)	.841	√
		PBR & NASA	46.3% (16)	50.3% (11)	.7	√
	PG & ATM (Req)	NASA Technique	24.2% (8)	32% (6)	.347	√
		PBR	33.9% (8)	35.8% (6)	.774	√
		PBR & NASA	29% (16)	33.9% (12)	.354	√
	All	NASA Technique	33.3% (16)	39.9% (12)	.452	√
		PBR	42% (16)	43.8% (11)	.839	√
		PBR & NASA	37.6% (32)	41.7% (23)	.504	√

Table 14 –Experience Writing Requirements (CMSC 435 Fall 1998, 735 Fall 1999, USC)

			Requirements Experience (Writing)			
Experiment	Artifact	Technique	None/Class	Industry	p-value	
435 Fall 1998	LA (Req)	Checklist	9.8% (19)	27.6% (1)	.081	√
		PBR (U)	10.3% (20)	12.9% (4)	.301	√
		Both	10.1% (39)	15.9% (5)	.099	√
735 Fall 1999	LA (Req)	PBR (U&T)	17.4% (8)	--	--	--
	PG (Req)	PBR (U&T)	27.3% (4)	14.1% (2)	.164	X
	LA & PG (Req)	PBR (U)	17.1% (7)	3.1% (1)	.165	X
		PBR (T)	25.8% (5)	25% (1)	.945	X
		PBR (U&T)	20.7% (12)	14.1% (2)	.409	X
USC Fall 2000	--	PBR	20.4% (18)	18.1% (11)	.839	X
USC Spring 2001	--	PBR	24.7% (18)	21.6% (13)	.799	X

Table 15 –Experience Writing Requirements (NASA 94)

			Experience Writing Requirements			
Experiment	**Artifact**	**Technique**	**None**	**Some**	**p-value**	
NASA 1994	NASA A	NASA Technique	8.3% (4)	13.9% (2)	.285	√
		PBR	15.6% (5)	27.8% (1)	.247	√
		PBR & NASA	12.3% (9)	18.5% (3)	.268	√
	NASA B	NASA Technique	12% (5)	40% (1)	.042	√
		PBR	13.3% (4)	0% (2)	.132	X
		PBR & NASA	12.6% (9)	13.3% (3)	.933	√
	PG (Req)	NASA Technique	18.1% (5)	18.8% (1)	.945	√
		PBR	24.2% (4)	20.3% (2)	.735	X
		PBR & NASA	20.8% (9)	19.8% (3)	.875	X
	ATM (Req)	NASA Technique	19.6% (4)	33.9% (2)	.141	√
		PBR	35% (5)	28.6% (1)	.669	X
		PBR & NASA	28.2% (9)	32.1% (3)	.639	√
	NASA (A & B)	NASA Technique	10.4% (9)	22.6% (3)	.081	√
		PBR	14.6% (9)	9.3% (3)	.458	X
		PBR & NASA	12.5% (18)	15.9% (6)	.481	√
	PG & ATM (Req)	NASA Technique	18.8% (9)	28.9% (3)	.107	√
		PBR	30.2% (9)	23.1% (3)	.412	X
		PBR & NASA	24.5% (18)	26% (6)	.791	√
	All	NASA Technique	14.6% (18)	25.7% (6)	.021	√
		PBR	22.4% (18)	16.2% (6)	.339	X
		PBR & NASA	18.5% (36)	20.9% (12)	.548	√

Table 16 –Experience Writing Requirements (NASA 95)

			Experience Writing Requirements			
Experiment	**Artifact**	**Technique**	**None**	**Some**	**p-value**	
NASA 1995	NASA A	NASA Technique	37.8% (3)	40% (5)	.91	√
		PBR	57.3% (5)	46.7% (1)	.709	X
		PBR & NASA	50% (8)	41.1% (6)	.513	X
	NASA B	NASA Technique	54.7% (5)	40% (1)	.682	X
		PBR	48.9% (3)	46.7% (4)	.938	X
		PBR & NASA	52.5% (8)	45.3% (5)	.685	X
	PG (Req)	NASA Technique	34.3% (5)	28.6% (1)	.811	X
		PBR	27.4% (3)	38.6% (5)	.208	√
		PBR & NASA	31.7% (8)	36.9% (6)	.521	√
	ATM (Req)	NASA Technique	19.8% (3)	25.2% (5)	.526	√
		PBR	37% (5)	25.9% (1)	.455	X
		PBR & NASA	30.6% (8)	25.3% (6)	.455	X
	NASA (A & B)	NASA Technique	48.3% (8)	40% (6)	.567	X
		PBR	54.2% (8)	46.7% (5)	.65	X
		PBR & NASA	51.3% (16)	43% (11)	.432	X
	PG & ATM (Req)	NASA Technique	28.9% (8)	25.7% (6)	.715	X
		PBR	33.4% (8)	36.5% (6)	.636	√
		PBR & NASA	31.1% (16)	31.1% (12)	.997	X
	All	NASA Technique	38.6% (16)	32.9% (12)	.513	X
		PBR	43.8% (16)	41.1% (11)	.76	X
		PBR & NASA	41.2% (32)	36.8% (23)	.474	X

Table 17 – General Use Case Experience (CMSC 735 Fall 1997)

			Requirements Experience (General)			
Experiment	Artifact	Technique	None/Class	Industry	p-value	
735 Fall 1997	ATM (Req)	Ad hoc	33.5% (59)	32.8% (6)	.913	X
	PG (Req)	PBR (U)	30.1% (19)	33.3 (3)	.718	√
		PBR (D, T)	29% (41)	31% (3)	.781	√
		PBR (U, D, T)	29.3% (60)	32.1% (6)	.599	√
	ATM & PG (Req)	Both	31.4% (119)	32.5% (12)	.804	√

Table 18 – Experience Writing Use Cases (CMSC 435 Fall 1998, USC)

			Use Case Experience (Writing)			
Experiment	Artifact	Technique	None	Industry	p-value	
435 Fall 1998	LA (Req)	Checklist	10.1% (19)	20.7% (1)	.315	√
		PBR (U)	11% (22)	8.6% (2)	.489	X
		Both	10.6% (41)	12.6% (3)	.648	√
USC Fall 200	--	PBR	19.4% (24)	20% (5)	.967	√
USC Spring 2001	--	PBR	26.3% (22)	16.4% (9)	.454	X

Table 19 –Software Design Experience (CMSC 735 Fall 1997, 735 Fall 1999, USC)

			Software Design Experience			
Experiment	Artifact	Technique	None	Industry	p-value	
735 Fall 1997	ATM (Req)	Ad hoc	30% (17)	34.7% (48)	.284	√
	PG (Req)	PBR (D)	29.2% (12)	33.6% (10)	.446	√
	PG (Req)	PGR (U, T)	20% (5)	29.9% (39)	.079	√
	PG (Req)	PBR (U, D, T)	26.5% (17)	30.7% (49)	.226	√
	ATM & PG (Req)	Ad hoc & PBR (U, D, T)	28.2% (34)	32.6% (97)	.112	√
735 Fall 1999	LA (Des)	OORTs	25.2% (5)	25% (2)	.97	X
	PG (Des)	OORTs	18.6% (1)	11.6% (6)	.371	X
	LA & PG (Des)	OORTs	24.1% (6)	15% (8)	.048	X
USC Spring 2001	--	OORTs	22.1% (16)	30% (10)	.341	√

			Beginner	Intermediate	Accomplished		
USC Fall 2000	--	PBR	9% (7)	16.5% (18)	9.8% (7)	.669	--
USC Spring 2001	--	PBR	16.7% (5)	27.5% (18)	25% (5)	.83	--
USC Spring 2001	--	OORTs	22.8% (5)	21.4% (18)	39% (5)	.227	√

Table 20 –Software Design Experience Based on Requirements (CMSC 435 Fall 1998, 735 Fall 1999)

			Software Design Experience (based on Requirements)			
Experiment	Artifact	Technique	None	Industry	p-value	
435 Fall 1998	LA (Req)	Checklist	11.5% (15)	8.3% (5)	.545	X
		PBR (U)	10.8% (16)	10.8% (8)	1	--
		Checklist & PBR (U)	11.1% (31)	9.8% (13)	.597	X
735 Fall 1999	LA (Des)	OORTs	25.2% (5)	25% (2)	.97	X
	PG (Des)	OORTs	11.6% (2)	13% (5)	.825	√
	LA & PG (Des)	OORTs	21.3% (7)	16.4% (7)	.314	X

Table 21 –Structured Design Experience (CMSC 735 Fall 1997)

			Structured Design Experience			
Experiment	Artifact	Technique	None	Industry	p-value	
735 Fall 1997	ATM (Req)	Ad hoc	34.4% (30)	32.6% (35)	.626	X
	PG (Req)	PBR (D)	29.2% (12)	33.6% (10)	.446	√
		PGR (U, T)	27.8% (19)	29.6% (25)	.635	√
		PBR (U, D, T)	28.3% (31)	30.7% (35)	.438	√
	ATM & PG (Req)	Both	31.3% (61)	31.6% (70)	.903	√

Table 22 –OO Design Experience (CSMC 735 Fall 1997, 735 Fall 1999, USC)

			OO Design Experience			
Experiment	Artifact	Technique	None	Industry	p-value	
735 Fall 1997	ATM (Req)	Ad hoc	34.3% (35)	32.4% (3)	.632	X
	PG (Req)	PBR (D)	31.6% (14)	30.3% (8)	.832	X
		PGR (U, T)	29.5% (22)	28.1% (22)	.69	X
		PBR (U, D, T)	30.4% (36)	28.7% (30)	.587	X
	ATM & PG (Req)	Ad hoc & PBR (U,D,T)	32.3% (71)	30.6% (60)	.482	X
735 Fall 1999	LA (Des)	OORTs	27% (2)	24.4% (5)	.62	X
	PG (Des)	OORTs	--	12.6% (7)	--	--
	LA & PG (Des)	OORTs	27% (2)	17.5% (12)	.163	X
USC Spring 2001	--	OORTs	19.4% (18)	38.1% (8)	.025	√

			Beginner	Intermediate	Accomplished		
USC Fall 2000	--	PBR	5% (8)	13.8% (17)	21.8% (7)	.331	√
USC Spring 2001	--	PBR	19.8% (7)	27.4% (16)	25% (5)	.892	√
USC Spring 2001	--	OORTs	17.9% (7)	21.7% (16)	44.3% (5)	.049	√

Table 23 – UML Experience (CMSC 735 Fall 1999, USC)

			UML Experience			
Experiment	Artifact	Technique	None	Industry	p-value	
735 Fall 1999	LA (Des)	OORTs	25.2% (5)	25% (2)	.97	X
	PG (Des)	OORTs	13.3% (4)	11.6% (3)	.761	X
	LA & PG (Des)	OORTs	19.9% (9)	17% (5)	.563	X
USC Spring 2001	--	OORTs	21.3% (16)	31.2% (10)	.229	√

Table 24 –Experience as Tester (NASA 94)

			Experience as Tester			
Experiment	Artifact	Technique	None	Some	p-value	
NASA 94	NASA A	NASA Technique	5.5% (3)	14.8% (3)	.007	√
		PBR	17.6% (6)	[redacted]	--	-
		PBR & NASA	13.6% (9)	14.8% (3)	.83	√
	NASA B	NASA Technique	16.7% (6)	[redacted]	--	-
		PBR	8.9% (3)	10% (4)	.892	√
		PBR & NASA	14.1% (9)	10% (4)	.585	X
	PG (Req)	NASA Technique	18.2% (6)	[redacted]	--	-
		PBR	27.1% (3)	18.8% (3)	.427	X
		PBR & NASA	21.2% (9)	18.8% (3)	.714	X
	ATM (Req)	NASA Technique	15.5% (3)	33.3% (3)	.016	√
		PBR	33.9% (6)	[redacted]	--	-
		PBR & NASA	27.8% (9)	33.3% (3)	.509	√
	NASA (A & B)	NASA Technique	13% (9)	14.8% (3)	.807	√
		PBR	14.7% (9)	10% (4)	.445	X
		PBR & NASA	13.8% (18)	12.1% (7)	.698	X
	PG & ATM (Req)	NASA Technique	17.3% (9)	33.3% (3)	.003	√
		PBR	31.6% (9)	18.8% (3)	.122	X
		PBR & NASA	24.5% (18)	26% 6)	.777	√
	ALL	NASA Technique	15.1% (18)	24.1% (6)	.071	√
		PBR	23.2% (18)	13.8% (7)	.114	X
		PBR & NASA	19.2% (36)	18.5% (13)	.872	X

Table 25 – Experience as Tester (NASA 95)

			Experience as Tester			
Experiment	**Artifact**	**Technique**	**None**	**Some**	**p-value**	
NASA 95	NASA A	NASA Technique	43.3% (4)	35% (4)	.66	X
		PBR	46.7% (4)	73.3% (2)	.188	√
		PBR & NASA	45% (8)	47.8% (6)	.839	√
	NASA B	NASA Technique	36.7% (4)	83.3% (2)	.025	√
		PBR	50% (4)	44.4% (3)	.845	X
		PBR & NASA	43.3% (8)	60% (5)	.337	√
	PG (Req)	NASA Technique	25% (4)	50% (2)	.122	√
		PBR	33% (4)	35.7% (4)	.771	√
		PBR & NASA	29% (8)	40.5% (6)	.143	√
	ATM (Req)	NASA Technique	23.1% (4)	23.1% (4)	1	X
		PBR	30.6% (4)	44.4% (2)	.205	√
		PBR & NASA	26.9% (8)	30.2% (6)	.631	√
	NASA (A & B)	NASA Technique	40% (8)	51.1% (6)	.443	√
		PBR	48.3% (8)	56% (5)	.643	√
		PBR & NASA	44.2% (16)	53.3% (11)	.38	√
	PG & ATM (Req)	NASA Technique	24.1% (8)	32.1% (6)	.335	√
		PBR	31.8% (8)	38.6% (6)	.279	√
		PBR & NASA	27.9% (16)	35.4% (12)	.151	√
	ALL	NASA Technique	32% (16)	41.6% (12)	.269	√
		PBR	40.1% (16)	46.5% (11)	.462	√
		PBR & NASA	36.1% (32)	44% (23)	.194	√

Table 26 – Experience as Tester (USC)

			Experience as Tester			
Experiment	Artifact	Technique	None	Industry	p-value	
USC Fall 2001	--	PBR	17.4% (13)	20% (17)	.814	√
USC Spring 2001	---	PBR	20.2% (18)	27.8% (13)	.534	√

Table 27 – Functional Testing Experience (CMSC 735 Fall 1997)

			Functional Testing Experience			
Experiment	Artifact	Technique	None	Industry	p-value	
735 Fall 1997	ATM (Req)	Ad hoc	33.7% (23)	33.3% (42)	.898	X
	PG (Req)	PBR (T)	22.7% (8)	28.3% (14)	.124	√
		PGR (U, D)	29.2% (15)	31.9% (29)	.527	√
		PBR (U, D, T)	27.5% (33)	30.7% (43)	.31	√
	ATM & PG (Req)	Both	30.6% (46)	32% (85)	.598	√

Table 28 – Testing Based on Requirements (CMSC 435 Fall 1998)

			Testing (Requirements) Experience			
Experiment	Artifact	Technique	None/Class	Industry	p-value	
435 Fall 1998	LA (Req)	Checklist	10.3% (12)	11.2% (8)	.855	√
		PBR (U)	10.8% (16)	10.8% (8)	1	--
		Checklist & PBR (U)	10.6% (28)	11% (16)	.865	√

Table 29 – Equivalence Partition Testing Experience (CMSC 735 Fall 1997, USC)

			Equivalence Partition Testing Experience			
Experiment	Artifact	Technique	None	Industry	p-value	
735 Fall 1997	ATM (Req)	Ad hoc	33.2% (61)	36.7% (4)	.666	√
	PG (Req)	PBR (T)	27.6% (21)	17.9% (1)	.329	X
		PGR (U, D)	30.3% (41)	38.1% (3)	.338	√
		PBR (U, D, T)	29.4% (62)	33% (4)	.568	√
	ATM & PG (Req)	Ad hoc & PBR (U, D, T)	31.3% (123)	34.9% (8)	.485	√
USC Fall 2000	--	PBR	21.7% (26)	0% (3)	.226	X
USC Spring 2001	--	PBR	25.1% (28)	7.4% (3)	.382	X

Table 30 – Perspective Experience (CMSC 735 Fall 1999)

			Perspective Experience			
Experiment	Artifact	Technique	None	Industry	p-value	
735 Fall 1999	LA (Req)	PBR (U&T)	13.8% (6)	27.8% (2)	.069	√
	PG (Req)	PBR (U&T)	16.7% (3)	29.2% (2)	.165	√
	LA & PG (Req)	PBR (U)	14.3% (7)	22.2% (1)	.46	√
		PBR (T)	16.5% (2)	30.2% (4)	.043	√
		PBR (U&T)	14.8% (9)	28.6% (5)	.007	√

Table 31 – Secondary Perspective Experience (CMSC 735 Fall 1999)

			Perspective (Secondary) Experience			
Experiment	**Artifact**	**Technique**	**None**	**Industry**	**p-value**	
735 Fall 1999	LA (Req)	PBR (U&T)	16.7% (7)	22.2% (1)	.627	√
	PG (Req)	PBR (U&T)	21.9% (5)	28.1% (1)	.645	√
	LA & PG (Req)	PBR (U)	14.3% (7)	22.2% (1)	.46	√
		PBR (T)	25.1% (5)	28.1% (1)	.788	√
		PBR (U&T)	18.8% (10)	25.2% (2)	.431	√

Process Experience

Table 32 – Comfort Reviewing Requirements (NASA 94)

			Comfort Reviewing Requirements			
Experiment	Artifact	Technique	None/Low	Medium/High	p-value	
NASA 94	NASA A	NASA Technique	5.6% (1)	11.1% (4)	.495	√
		PBR	16.7% (1)	17.8% (5)	.924	√
		PBR & NASA	11.1% (2)	14.8% (9)	.599	√
	NASA B	NASA Technique	20% (1)	16% (5)	.824	X
		PBR	6.7% (1)	11.7% (4)	.721	√
		PBR & NASA	13.3% (2)	14.1% (9)	.942	√
	PG (Req)	NASA Technique	18.8% (1)	18.1% (5)	.945	X
		PBR	40.6% (1)	17.2% (4)	.067	X
		PBR & NASA	29.7% (2)	17.7% (9)	.104	X
	ATM (Req)	NASA Technique	10.7% (1)	24.1% (4)	.228	√
		PBR	32.1% (1)	34.3% (5)	.888	X
		PBR & NASA	21.4% (2)	29.8% (9)	.403	√
	NASA (A & B)	NASA Technique	12.8% (2)	13.8% (9)	.911	√
		PBR	11.7% (2)	15.1% (9)	.677	√
		PBR & NASA	12.2% (4)	14.4% (18)	.702	√
	PG & ATM (Req)	NASA Technique	14.7% (2)	20.8% (9)	.345	√
		PBR	36.4% (2)	2.7% (9)	.366	X
		PBR & NASA	25.6% (4)	23.7 % (18)	.779	X
	ALL	NASA Technique	13.8% (4)	17.3% (16)	.527	√

		PBR	24% (4)	20.9% (18)	.678	X
		PBR & NASA	18.9% (8)	19.1% (36)	.966	√

Table 33 – Comfort Reviewing Requirements (NASA 95)

			Comfort Reviewing Req.			
Experiment	**Artifact**	**Technique**	**None/Low**	**Medium/High**	**p-value**	
NASA 95	NASA A	NASA Technique	37.8% (3)	40% (5)	.91	√
		PBR	66.7% (1)	61.7% (4)	.815	X
		PBR & NASA	45% (4)	49.6% (9)	.759	√
	NASA B	NASA Technique	20% (1)	66.7% (4)	.142	√
		PBR	48.9% (3)	46.7% (4)	.938	X
		PBR & NASA	41.7% (4)	56.7% (8)	.434	√
	PG (Req)	NASA Technique	17.9% (1)	41.1% (4)	.334	√
		PBR	46.7% (4)	38.6% (5)	.642	X
		PBR & NASA	40.9% (5)	39.7% (9)	.923	X
	ATM (Req)	NASA Technique	19.8% (3)	25.2% (5)	.526	√
		PBR	33.3% (1)	38.9% (4)	.721	√
		PBR & NASA	23.1% (4)	31.3% (9)	.309	√
	NASA (A & B)	NASA Technique	33.3% (4)	51.9% (9)	.253	√
		PBR	53.3% (4)	54.2% (8)	.962	√
		PBR & NASA	43.3% (8)	52.9% (17)	.402	√
	PG & ATM (Req)	NASA Technique	19.3% (4)	32.2% (9)	.162	√
		PBR	44% (5)	38.7% (9)	.649	X
		PBR & NASA	33% (9)	35.5% (18)	.75	√
	ALL	NASA Technique	26.3% (8)	42% (18)	.104	√
		PBR	48.1% (9)	46% (17)	.83	X
		PBR & NASA	37.8% (17)	44% (35)	.386	√

Table 34 – Experience Reviewing Requirements (CMSC 435 Fall 1998, 735 Fall 1999, USC)

			Experience Reviewing Requirements			
Experiment	**Artifact**	**Technique**	**None/Class**	**Industry**	**p-value**	
CMSC 435 Fall 1998	LA (Req)	Checklist	10.3% (17)	12.6% (3)	.722	√
		PBR (U)	10.3% (19)	12.4% (5)	.369	√
		Checklist & PBR (U)	10.3% (36)	12.5% (8)	.461	√
CMSC 735 Fall 1999	LA (Req)	PBR (U&T)	16.7% (7)	22.2% (1)	.627	√
	PG (Req)	PBR (U&T)	23.4% (2)	22.7% (4)	.943	X
	LA & PG (Req)	PBR (U)	16.2% (6)	12.7% (2)	.67	X
		PBR (T)	22.1% (3)	29.2% (3)	.369	√
		PBR (U&T)	18.2% (9)	22.6% (5)	.454	√
USC Fall 2000	--	PBR (T, U & D)	18.3% (20)	19.9% (10)	.895	√
USC Spring 2001	--	PBR (T, U & D)	26% (20)	18.9% (11)	.562	X

Table 35 – OO Reading Experience (CMSC 735 Fall 1999, USC)

			OO Reading Experience			
Experiment	**Artifact**	**Technique**	**None/Class**	**Industry**	**p-value**	
CMSC 735 Fall 1999	LA (Des)	OORT	30% (2)	23.2% (5)	.154	X
	PG (Des)	OORT	20.9% (1)	11.2% (6)	.193	X
	LA & PG (Des)	OORT	27% (3)	16.7% (11)	.084	X
USC Spring 2001	--	OORT	22.4% (16)	29.4% (10)	.4	√

Table 36 – Software Inspection Experience (CMSC 735 Fall 1999, USC)

			Software Inspection Experience			
Experiment	**Artifact**	**Technique**	**None/Class**	**Industry**	**p-value**	
CMSC 735 Fall 1999	LA (Req)	PBR (U&T)	17.4% (8)	--	--	--
	PG (Req)	PBR (U&T)	26.6% (2)	21.1% (4)	.609	X
	LA & PG (Req)	PBR (U)	17.1% (7)	3.1% (3)	.165	X
		PBR (T)	24.2% (3)	27.1% (3)	.726	√
		PBR (U&T)	19.2% (10)	21.1% (4)	.764	√
	LA (Des)	OORT	25.1% (7)	--	--	--
	PG (Des)	OORT	16.3% (3)	9.9% (4)	.23	X
	LA & PG (Des)	OORT	22.5% (10)	9.9% (4)	.008	X
USC Fall 2000	--	PBR	23.4% (19)	15% (8)	.51	X
USC Spring 2001	--	PBR	24.4% (26)	14.6% (4)	.593	X
	--	OORT	27.7% (22)	19.6% (4)	.463	X

Working Language Experience

Table 37 – Native English Speaker (735 Fall 1999, USC)

			Native English Speaker			
Experiment	**Artifact**	**Technique**	**Non-Native**	**Native**	**p-value**	
735 Fall 1999	LA (Req)	PBR (U&T)	14.8% (6)	25% (2)	.216	√
	PG (Req)	PBR (U&T)	25% (3)	20.8% (3)	.682	X
	LA & PG (Req)	PBR (U)	13.9% (5)	17.7% (3)	.605	√
		PBR (T)	23.6% (4)	29.7% (2)	.475	√
		PBR (U&T)	18.2% (9)	22.5% (5)	.466	√
	LA (Des)	OORTs	22% (4)	29.3% (3)	.074	√
	PG (Des)	OORTs	15.7% (4)	8.5% (3)	.168	X
	LA & PG (Des)	OORTs	18.8% (8)	18.9% (6)	.987	√
USC Fall 2000	--	PBR	21.7%	8%	.332	X
USC Spring 2001	--	PBR	24% (24)	21.4% (7)	.861	X
	--	OORTs	23.7% (25)	34.9% (6)	.216	√

Table 38 – Reading Comprehension (CMSC 735 Fall 1999, USC)

			Reading Comprehension			
Experiment	**Artifact**	**Technique**	**Low**	**High**	**p-value**	
735 Fall 1999	LA (Req)	PBR (U&T)	18.1% (4)	8.3% (2)	.295	X
	PG (Req)	PBR (U&T)	--	25% (3)	--	--
	LA & PG (Req)	PBR (U)	13.9% (2)	13.9% (3)	1	--
		PBR (T)	22.2% (2)	25% (2)	.832	√
		PBR (U&T)	18.1% (4)	18.3% (5)	.968	√
	LA (Des)	OORTs	25% (2)	19% (2)	.198	X
	PG (Des)	OORTs	15.7% (4)	--	--	--
	LA & PG (Des)	OORTs	18.8% (6)	19% (2)	.969	√
USC Fall 2000	--	PBR	12.1% (4)	23.5% (22)	.496	√
USC Spring 2001	--	PBR	24.2% (10)	23.8% (14)	.982	X
	--	OORTs	28.5% (13)	18.9% (11)	.263	X

Table 39 – Listening and Speaking (CMSC 735 Fall 1999, USC)

			Listening and Speaking			
Experiment	**Artifact**	**Technique**	**Low**	**High**	**p-value**	
735 Fall 1999	LA (Req)	PBR (U&T)	14.8% (6)	--	--	--
	PG (Req)	PBR (U&T)	--	25% (3)	--	--
	LA & PG (Req)	PBR (U)	11.1% (4)	25% (1)	.071	√
		PBR (T)	22.2% (2)	25% (2)	.832	√
		PBR (U&T)	14.8% (6)	25% (3)	.13	√
	LA (Des)	OORTs	25% (2)	19% (2)	.198	X
	PG (Des)	OORTs	15.7% (4)	--	--	--
	LA & PG (Des)	OORTs	18.8% (6)	19% (2)	.969	√
USC Fall 2000	--	PBR	9.7% (5)	24.6% (21)	.327	√
USC Spring 2001	--	PBR	22% (11)	25.6% (13)	.815	√
	--	OORTs	30% (12)	18.3% (12)	.168	X

Appendix C – Background Data Collection Forms

This appendix contains the forms used to collect background metrics on the subjects. For each questionnaire included here, the mapping from the raw data to the grouping of "high" and "low" is shown. Beside each background item, there are either arrows that indicate which choices map to "high" and which map to "low", or the mapping is described with text.

NASA 1994 Study

1. How many years/months experience have you had in each function?

~~a. Manger~~
b. Developer (**“low” = < 3 years “high” = > 5 years**)
c. Tester (**“low” = 0 years “high” = > 0 years**)
~~d. Other~~

2. How comfortable are you with reading or reviewing requirements documents? (Choose One)

0 = Not at all ← **“low”**
1 = Low ←
2 = Moderate ← **“high”**
3 = High ←

3. How many years/months of experience do you have using requirements documents?
(**“low” = < 3 years “high” = > 5 years**)

4. How many years/months of experience do you have writing requirements documents?
(**“low” = 0 years “high” = > 0 years**)

~~5. What do you think you might get out of this experiment? (Choose One)~~

~~0 = Nothing~~
~~1 = Little~~
~~2 = Moderate~~
~~3 = A lot~~

NASA 1995 Study

1. How many years/months experience have you had in each function?

    ~~a. Manger~~
    b. Developer (“low” = < 3 years “high” = > 5 years)
    c. Tester (“low” = 0 years “high” = > 0 years)
    ~~d. Analyst~~
    ~~e. Other~~

2. How comfortable are you with reading or reviewing requirements documents? (Choose One)

    0 = Not at all → **“low”**
    1 = Low
    2 = Moderate → **“high”**
    3 = High

3. How many years/months of experience do you have using requirements documents?

    (“low” = < 3 years “high” = > 5 years)

4. How many years/months of experience do you have writing requirements documents?

    (“low” = 0 years “high” = < 0 years”)

~~5. What do you think you might get out of this experiment? (Choose One)~~

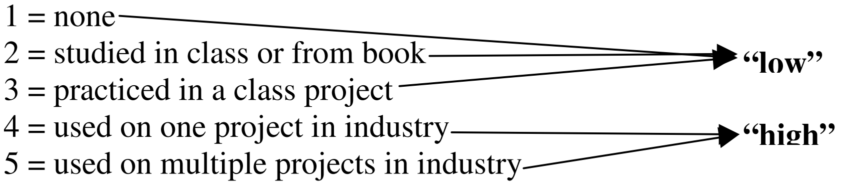
~~0 = Nothing~~
~~1 = Little~~
~~2 = Moderate~~
~~3 = A lot~~

Questionnaire

CMSC 735 Fall 1997

Name ____

Please, grade your experience with respect to the following 5-point scale



1 = none
2 = studied in class or from book
3 = practiced in a class project
4 = used on one project in industry
5 = used on multiple projects in industry

Analysis

- Have you experience in requirements analysis? 1 2 3 4 5
- Have you experience in use-case analysis technique? 1 2 3 4 5

Design

- Have you experience in software design? 1 2 3 4 5
- Have you experience in Structured Design? 1 2 3 4 5
- Have you experience in Object-Oriented Design? 1 2 3 4 5

Testing

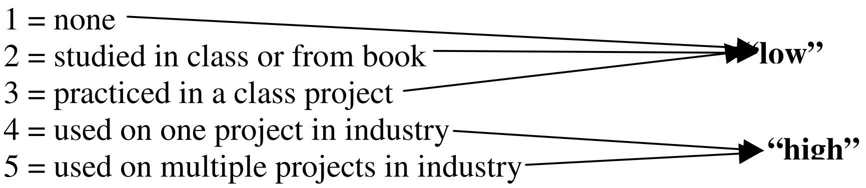
- Have you experience in functional testing? 1 2 3 4 5
- Have you experience in equivalence partitioning technique? 1 2 3 4 5

Questionnaire

CMSC 435 Fall 1998

Name ____

Please grade your experience with respect to the following 5-point scale:



1 = none
2 = studied in class or from book
3 = practiced in a class project
4 = used on one project in industry
5 = used on multiple projects in industry

Experience with Requirements

• Experience writing requirements	1	2	3	4	5
• Experience writing use cases	1	2	3	4	5
• Experience reviewing requirements	1	2	3	4	5
• Experience reviewing use cases	1	2	3	4	5
• Experience changing requirements for maintenance	1	2	3	4	5

Experience in Design and Coding

• Experience in design of systems from requirements	1	2	3	4	5
• Experience in design of systems from use cases	1	2	3	4	5
• Experience changing designs for maintenance	1	2	3	4	5
• Experience in coding, based on requirements	1	2	3	4	5
• Experience in coding, based on use cases	1	2	3	4	5
• Experience in maintenance of code	1	2	3	4	5

Experience in Testing

• Experience in testing, based on requirements	1	2	3	4	5
• Experience in testing, based on use cases	1	2	3	4	5

Experience Questionnaire

CMSC 735 Fall 1999

Name ____

General Background

Please estimate your English-language background: **“Native”**

__ I am a native speaker. **“Non-Native”**

__ English is my second language. [Please complete both of the following.]

My reading comprehension skills are:

__ could be better

__ moderate **“low”**

__ high **“high”**

__ very high

My listening and speaking skills are:

__ could be better

__ moderate **“low”**

__ high **“high”**

__ very high

What is your previous experience with software development in practice? (Check the bottom-most item that applies.)

__ I have never developed software. **“low”**

__ I have developed software on my own.

__ I have developed software as a part of a team, as part of a course.

__ I have developed software as a part of a team, in industry. **“high”**

Please explain your answer. Include the number of semesters or number of years of relevant experience. (E.g. “I worked for 10 years as a programmer in industry.”)

Software Development Experience

Please rate your experience in this section with respect to the following 5-point scale:

1 = none

2 = studied in class or from book **“low”**

3 = practiced in a class project

4 = used on one project in industry **“high”**

5 = used on multiple projects in industry

Experience with Requirements

• Experience writing requirements	1	2	3	4	5
• Experience writing use cases	1	2	3	4	5
• Experience reviewing requirements	1	2	3	4	5

- Experience reviewing use cases 1 2 3 4 5
- Experience changing requirements for maintenance 1 2 3 4 5

Experience in Design

- Experience in design of systems 1 2 3 4 5
- Experience in design of systems from requirements/use cases 1 2 3 4 5
- Experience with creating Object-Oriented (OO) designs 1 2 3 4 5
- Experience with reading OO designs 1 2 3 4 5
- Experience with the Unified Modeling Language (UML) 1 2 3 4 5
- Experience changing designs for maintenance 1 2 3 4 5

Experience in Coding

- Experience in coding, based on requirements/use cases 1 2 3 4 5
- Experience in coding, based on design 1 2 3 4 5
- Experience in coding, based on OO design
- Experience in maintenance of code 1 2 3 4 5

Experience in Testing

- Experience in testing software 1 2 3 4 5
- Experience in testing, based on requirements/use cases 1 2 3 4 5
- Experience with equivalence-partition testing 1 2 3 4 5

Other Experience

- Experience with software project management? 1 2 3 4 5
- Experience with User Interface (UI) design? 1 2 3 4 5
- Experience with software inspections? 1 2 3 4 5

Experience in Problem Domains

We will use answers in this section to understand how familiar you are with various systems we may use as examples or for assignments during the class.

Please rate your experience in this section with respect to the following 3-point scale:

1 = I'm really unfamiliar with the concept. I've never done it.
3 = I've done this a few times, but I'm no expert.
5 = I'm very familiar with this area. I would be very comfortable doing this.

"low"
"high"

How much do you know about…

- Applying for a loan? 1 3 5
- Applying for a mortgage? 1 3 5
- Using a parking garage? 1 3 5
- Using an ATM? 1 3 5
- Renting movies from a video rental store (e.g. Blockbusters)? 1 3 5

USC studies

General Background

Put an X next to the phrase that best describes your English-language background:
__ I am a native speaker. **“Native”**
__ English is my second language. [Please complete both of the following.]
My reading comprehension skills are: **“Non-Native”**
__ could be better **“low”**
__ moderate
__ high **“high”**
__ very high
My listening and speaking skills are:
__ could be better **“low”**
__ moderate
__ high **“high”**
__ very high

What is your previous experience with software development in practice, not including this class?
(Place an X next to the bottom-most item that applies.)
__ I have never developed software. **“low”**
__ I have developed software on my own.
__ I have developed software as a part of a team, as part of a course.
__ I have developed software as a part of a team, in industry. **“high”**

Please explain your answer. Inclue the number of semesters or number of years of relevant experience.
(E.g. "I worked for 10 years as a programmer in industry.")

Software Development Experience

Please rate your experience in this section with respect to the following 5-point scale:
(In each case, choose the highest number that applies)
1 = none
2 = studied in class or from book **“low”**
3 = practiced in a class project
4 = used on one project in industry
5 = used on multiple projects in industry **“high”**

<u>Experience with Requirements</u>
__ Experience writing requirements
__ Experience writing use cases
__ Experience reviewing requirements
__ Experience reviewing use cases

__ Experience changing requirements for maintenance

Experience in Design

__ Experience in design of systems
__ Experience in design of systems from requirements/use cases
__ Experience with creating Object-Oriented (OO) designs
__ Experience with reading OO designs
__ Experience with the Unified Modeling Language (UML)

Experience in Testing

__ Experience in testing software
__ Experience in testing, based on requirements/use cases
__ Experience with equivalence-partition testing

Other Experience

__ Experience with software inspections?

Appendix D – Brazilian Results (R1)

This appendix contains the analyzed data from the first replication run in Brazil (R1) discussed in Chapter 5. The data is grouped and analyzed based on the background and experience variables defined in Chapter 4. Because each variable is defined by a series of metrics, there is a table of data for each metric. The tables are all in the following format:

Example Table

			Metric			
Experiment	**Artifact**	**Technique**	**Group 1**	**Group 2**	**p-value**	
R1	ATM	Checklist	10.7% (18)	10.3% (2)	.96	X
	PG	PBR	11% (22)	8.6% (2)	.489	X

Where:

- *Experiment* – R1 – first replication in Brazil.
- *Artifact* – The artifact that was inspected by the subjects in the treatment group.
  - o PG – Parking Garage
  - o ATM – Automated Teller Machine
- *Technique* – The technique used by the subjects in the treatment group.
  - o Checklist – A non-procedural list of items to look for in a software artifact during an inspection.
  - o PBR – Perspective Based Reading; all 3 perspectives (Tester, User, Designer) were used, but there were not enough subjects to analyze each perspective separately.
- *Group 1/Group 2* – These two columns represent the average percent of the known defects found by the subjects in each of the two treatments represented by the columns. The number in parenthesis is the number of subjects that were in that group.
- *p-value* – The p-value obtained from running a t-test between the two treatment groups
- *√/X* – A '√' indicates that the more experienced group found more defects than the less experienced group, while an 'X' indicates that the less experience group found more defects than the more experience group.

In each table below, shaded rows indicate that the data in that row is an aggregation of data appearing in other rows.

Process Experience

Table 40 – Comfort Reading Requirements (R1)

			Comfort Reading Requirements			
Experiment	**Artifact**	**Technique**	**Low**	**High**	**p-value**	
R1	PG	Checklist	15.6% (2)	16.9% (5)	.88	√
		PBR	16.4% (4)	14.4% (5)	.483	X
		PBR & Checklist	16.1% (6)	15.6% (10)	.877	X
	ATM	Checklist	5.4% (4)	9.2% (5)	.395	√
		PBR	12.2% (2)	14.1% (5)	.672	√
		PBR & Checklist	7.7% (6)	11.6% (10)	.228	√
	PG & ATM	Checklist	8.8% (6)	13% (10)	.35	√
		PBR	15% (6)	14.2% (10)	.736	X
		PBR & Checklist	11.9% (12)	13.6% (20)	.494	√

Software Development Experience

Table 41 – Experience as a Developer (R1)

			Experience as a Developer			
Experiment	**Artifact**	**Technique**	**< 1 Year**	**> = 1 Year**	**p-value**	
R1	PG	Checklist	16.6% (3)	16.4% (4)	.973	X
		PBR	14.6% (3)	15.6% (6)	.736	√
		PBR & Checklist	15.6% (6)	15.9% (10)	.926	√
	ATM	Checklist	7.2% (5)	7.7% (4)	.926	√
		PBR	16.2% (3)	11.5% (4)	.211	X
		PBR & Checklist	11.7% (8)	9.2% (8)	.45	X
	PG & ATM	Checklist	11.9% (8)	11.2% (8)	.865	X
		PBR	15.4% (6)	14% (10)	.532	X
		PBR & Checklist	13.7% (14)	12.6% (18)	.661	X

Table 42 –Experience as Tester/Writing Requirements/Using Requirements (R1)

			Experience as Tester Experience Writing Requirements Experience Using Requirements			
Experiment	**Artifact**	**Technique**	**< 1 Year**	**> = 1 Year**		
R1	PG	Checklist	All but one subject had no industrial experience			
		PBR				
		PBR & Checklist				
	ATM	Checklist				
		PBR				
		PBR & Checklist				
	PG & ATM	Checklist				
		PBR				
		PBR & Checklist				

Working Language Experience

Table 43 – Reading Comprehension Level (R1)

			Reading Comprehension			
Experiment	**Artifact**	**Technique**	**Low**	**High**	**p-value**	
R1	PG	Checklist	20.8% (3)	13.3% (4)	.288	X
		PBR	14.6% (3)	15.6% (6)	.736	√
		PBR & Checklist	17.7% (6)	14.6% (10)	.363	X
	ATM	Checklist	7.2% (3)	7.7% (6)	.926	√
		PBR	11.7% (3)	14.9% (4)	.428	√
		PBR & Checklist	9.5% (6)	10.5% (10)	.748	√
	PG & ATM	Checklist	14% (6)	9.9% (10)	.363	X
		PBR	13.1% (6)	15.3% (10)	.338	√
		PBR & Checklist	13.6% (12)	12.6% (20)	.701	X

Table 44 – Listening, Speaking and Writing Skills (R1)

			Listening and Speaking (and Writing)			
Experiment	**Artifact**	**Technique**	**Low**	**High**	**p-value**	
R1	PG	Checklist	18.8% (5)	10.9% (2)	.319	X
		PBR	15.6% (7)	14.1% (2)	.655	X
		PBR & Checklist	16.9% (12)	12.5% (4)	.228	X
	ATM	Checklist	7.3% (7)	8.1% (2)	.888	√
		PBR	12.4% (5)	16.2% (2)	.381	√
		PBR & Checklist	9.5% (12)	12.2% (4)	.47	√
	PG & ATM	Checklist	12.1% (12)	9.5% (4)	.615	X
		PBR	14.3% (12)	15.1% (4)	.743	√
		PBR & Checklist	13.2% (24)	12.3% (8)	.76	X

Appendix E – Brazilian Results (R2)

This appendix contains the analyzed data from the second replication run in Brazil (R2) discussed in Chapter 5. The data is grouped and analyzed based on the background and experience variables defined in Chapter 4. Because each variable is defined by a series of metrics, there is a table of data for each metric. The tables are all in the following format:

Example Table

			Metric			
Experiment	**Artifact**	**Technique**	**Group 1**	**Group 2**	**p-value**	
R2	ATM	Checklist	10.7% (18)	10.3% (2)	.96	X
	PG	PBR	11% (22)	8.6% (2)	.489	X

Where:

- *Experiment* – R2 – first replication in Brazil.
- *Artifact* – The artifact that was inspected by the subjects in the treatment group.
    - PG – Parking Garage
    - ATM – Automated Teller Machine
- *Technique* – The technique used by the subjects in the treatment group.
    - Checklist – A non-procedural list of items to look for in a software artifact during an inspection.
    - PBR – Perspective Based Reading; all 3 perspectives (Tester, User, Designer) were used, but there were not enough subjects to analyze each perspective separately.
- *Group 1/Group 2* – These two columns represent the average percent of the known defects found by the subjects in each of the two treatments represented by the columns. The number in parenthesis is the number of subjects that were in that group.
- *p-value* – The p-value obtained from running a t-test between the two treatment groups
- *√/X* – A '√' indicates that the more experienced group found more defects than the less experienced group, while an 'X' indicates that the less experience group found more defects than the more experience group.

In each table below, shaded rows indicate that the data in that row is an aggregation of data appearing in other rows.

Process Experience

Table 45 – Comfort Reading Requirements (R2)

			Comfort Reading Requirements			
Experiment	**Artifact**	**Technique**	**Low**	**High**	**p-value**	
R2	PG	Checklist	11.7% (4)	15% (5)	.456	√
		PBR	15.6% (8)	12.5% (1)	.407	X
		PBR & Checklist	14.3% (12)	14.6% (6)	.918	√
	ATM	Checklist	9.5% (8)	16.2% (1)	.163	√
		PBR	13.5% (4)	11.4% (5)	.511	X
		PBR & Checklist	10.8% (12)	12.2% (6)	.562	√
	PG & ATM	Checklist	10.2% (12)	15.2% (6)	.064	√
		PBR	14.9% (12)	11.5% (6)	.105	X
		PBR & Checklist	12.6% (24)	13.4% (12)	.645	√

Software Development Experience

Table 46 – Experience as a Developer (R2)

			Experience as a Developer			
Experiment	**Artifact**	**Technique**	**< 1 Year**	**> = 1 Year**	**p-value**	
R2	PG	Checklist	11.2% (5)	16.4% (4)	.225	√
		PBR	14.8% (8)	18.8% (1)	.292	√
		PBR & Checklist	13.5% (13)	16.9% (5)	.185	√
	ATM	Checklist	9.8% (8)	13.5% (1)	.467	√
		PBR	11.4% (5)	13.5% (4)	.511	√
		PBR & Checklist	10.4% (13)	13.5% (5)	.193	√
	PG & ATM	Checklist	10.4% (13)	15.8% (5)	.052	√
		PBR	13.5% (13)	14.6% (5)	.639	√
		PBR & Checklist	11.9% (26)	15.2% (10)	.069	√

Table 47 – Experience as Tester/Writing Requirements/Using Requirements (R2)

			Experience as Tester Experience Writing Requirements Experience Using Requirements			
Experiment	**Artifact**	**Technique**	**< 1 Year**	**> = 1 Year**		
R2	PG	Checklist	Only one subject hat more than 1 year experience as a Tester. No subjects has more than 1 year experience writing or using requirements.			
		PBR				
		PBR & Checklist				
	ATM	Checklist				
		PBR				
		PBR & Checklist				
	PG & ATM	Checklist				
		PBR				
		PBR & Checklist				

Working Language Experience

Table 48 – Reading Comprehension Level (R2)

			Reading Comprehension			
Experiment	**Artifact**	**Technique**	**Low**	**High**	**p-value**	
R2	PG	Checklist	12.5% (2)	13.8% (7)	.803	√
		PBR	15.6% (5)	14.8% (4)	.749	X
		PBR & Checklist	14.7% (7)	14.2% (11)	.828	X
	ATM	Checklist	8.6% (5)	12.2% (4)	.264	√
		PBR	16.2% (2)	11.2% (7)	.179	X
		PBR & Checklist	10.8% (7)	11.5% (11)	.744	√
	PG & ATM	Checklist	9.7% (7)	13.2% (11)	.193	√
		PBR	15.8% (7)	12.5% (11)	.102	X
		PBR & Checklist	12.8% (14)	12.9% (22)	.951	√

Table 49 – Listening, Speaking and Writing Skills (R2)

			Listening and Speaking (and Writing)			
Experiment	**Artifact**	**Technique**	**Low**	**High**	**p-value**	
R2	PG	Checklist	14.4% (5)	12.5% (4)	.675	X
		PBR	15.6% (8)	12.5% (1)	.407	X
		PBR & Checklist	15.1% (13	12.5% (5)	.31	X
	ATM	Checklist	10.1% (8)	10.8% (1)	.897	√
		PBR	13% (5)	11.5% (4)	.655	X
		PBR & Checklist	11.2% (13)	11.4% (4)	.96	√
	PG & ATM	Checklist	11.8% (13)	12.2% (4)	.895	√
		PBR	14.6% (13)	11.7% (4)	.187	X
		PBR & Checklist	13.2% (26)	11.9% (8)	.493	X

Application Domain Knowledge

Table 50 – Application Domain Knowledge (R2)

			Domain Knowledge			
Experiment	**Artifact**	**Technique**	**Low**	**High**	**p-value**	
R2	PG	Checklist	13% (6)	14.6% (3)	.741	√
		PBR	15.3% (9)	--	--	--
		PBR & Checklist	14.4% (15)	14.6% (3)	.979	√
	ATM	Checklist	9.5% (2)	10.4% (7)	.806	√
		PBR	16.2% (2)	11.2% (7)	.179	X
		PBR & Checklist	12.8% (4)	10.8% (14)	.441	X
	PG & ATM	Checklist	12.1% (8)	11.7% (10)	.865	X
		PBR	15.7% (11)	11.2% (7)	.028	X
		PBR & Checklist	14.1% (19)	11.5% (17)	.113	X

Appendix F – Brazilian Results (R3)

This appendix contains the analyzed data from the third replication run in Brazil (R3) discussed in Chapter 5. The data is grouped and analyzed based on the background and experience variables defined in Chapter 4. Because each variable is defined by a series of metrics, there is a table of data for each metric. The tables are all in the following format:

Example Table

			Metric			
Experiment	**Artifact**	**Technique**	**Group 1**	**Group 2**	**p-value**	
R3	ATM	Checklist	10.7% (18)	10.3% (2)	.96	X
	PG	PBR	11% (22)	8.6% (2)	.489	X

- *Experiment* – R3 – first replication in Brazil.
- *Artifact* – The artifact that was inspected by the subjects in the treatment group.
  - PG – Parking Garage
  - ATM – Automated Teller Machine
- *Technique* – The technique used by the subjects in the treatment group.
  - Checklist – A non-procedural list of items to look for in a software artifact during an inspection.
  - PBR – Perspective Based Reading; all 3 perspectives (Tester, User, Designer) were used, but there were not enough subjects to analyze each perspective separately.
- *Group 1/Group 2* – These two columns represent the average percent of the known defects found by the subjects in each of the two treatments represented by the columns. The number in parenthesis is the number of subjects that were in that group.
- *p-value* – The p-value obtained from running a t-test between the two treatment groups
- *√/X* – A '√' indicates that the more experienced group found more defects than the less experienced group, while an 'X' indicates that the less experience group found more defects than the more experience group.

In each table below, shaded rows indicate that the data in that row is an aggregation of data appearing in other rows.

Process Experience

Table 51 – Experience Reviewing Requirements (R3)

			Experience Reviewing Requirements			
Experiment	**Artifact**	**Technique**	**Non-Industry**	**Industry**	**p-value**	
R3	PG	Checklist	15.6% (7)	10.9% (2)	.311	X
		PBR	16.8% (8)	25% (1)	.471	√
		PBR & Checklist	16.3% (15)	15.6% (3)	.905	X
	ATM	Checklist	12.5% (8)	16.2% (1)	.55	√
		PBR	7.3% (7)	4.1% (2)	.473	X
		PBR & Checklist	10.1% (15)	8.1% (3)	.622	X
	PG & ATM	Checklist	14% (15)	12.7% (3)	.719	X
		PBR	12.4% (15)	11% (3)	.832	X
		PBR & Checklist	13.2% (30)	11.9% (6)	.71	X

Table 52 – Software Inspection Experience (R3)

			Software Inspection Experience			
Experiment	**Artifact**	**Technique**	**Non-Industry**	**Industry**	**p-value**	
R3	PG	Checklist	16.5% (7)	7.8% (2)	.032	X
		PBR	18% (8)	15.6% (1)	.84	X
		PBR & Checklist	17.3% (15)	10.4% (3)	.176	X
	ATM	Checklist	13.2% (8)	10.8% (1)	.706	X
		PBR	6.6% (7)	6.8% (2)	.967	√
		PBR & Checklist	10.1% (15)	8.1% (3)	.622	X
	PG & ATM	Checklist	14.7% (15)	8.8% (3)	.076	X
		PBR	12.6% (15)	9.7% (3)	.642	X
		PBR & Checklist	13.7% (30)	9.3% (6)	.201	X

Software Development Experience

Table 53 – Experience as a Developer (R3)

			Experience as Developer			
Experiment	**Artifact**	**Technique**	**Non-Industry**	**Industry**	**p-value**	
R3	PG	Checklist	17.5% (5)	10.9% (4)	.064	X
		PBR	14.6% (6)	24% (3)	.197	√
		PBR & Checklist	15.9% (11)	16.5% (7)	.879	X
	ATM	Checklist	10.8% (6)	17.1% (3)	.096	√
		PBR	5.9% (5)	7.4% (4)	.702	√
		PBR & Checklist	8.6% (11)	11.6% (7)	.326	√
	PG & ATM	Checklist	13.9% (11)	13.6% (7)	.921	X
		PBR	10.7% (11)	14.5% (7)	.421	√
		PBR & Checklist	12.3% (11)	14.1% (7)	.501	√

Table 54 – Experience Writing Requirements (R3)

			Experience Writing Requirements			
Experiment	Artifact	Technique	Non-Industry	Industry	p-value	
R3	PG	Checklist	16.7% (6)	10.4% (3)	.104	X
		PBR	11.9% (5)	25% (4)	.036	√
		PBR & Checklist	14.5% (11)	18.8% (7)	.277	√
	ATM	Checklist	11.4% (5)	14.9% (4)	.363	√
		PBR	7.2% (6)	5.4% (3)	.659	X
		PBR & Checklist	9.1% (11)	10.8% (7)	.575	√
	PG & ATM	Checklist	14.3% (11)	13% (7)	.639	X
		PBR	9.3% (11)	16.6% (7)	.119	√
		PBR & Checklist	11.8% (22)	14.8% (14)	.26	√

Table 55 – Experience Writing Use Cases (R3)

			Experience Writing Use Cases			
Experiment	Artifact	Technique	Non-Industry	Industry	p-value	
R3	PG	Checklist	16.7% (6)	10.4% (3)	.014	X
		PBR	17.7% (9)	--	--	--
		PBR & Checklist	17.3% (15)	10.4% (3)	.176	X
	ATM	Checklist	12.9% (9)	--	--	--
		PBR	7.2% (6)	5.4% (3)	.659	X
		PBR & Checklist	10.6% (15)	5.4% (3)	.182	X
	PG & ATM	Checklist	14.4% (15)	10.4% (3)	.0244	X
		PBR	13.5% (15)	5.4% (3)	.189	X
		PBR & Checklist	14% (30)	7.9% (6)	.077	X

Table 56 – Software Design Experience (R3)

			Software Design Experience			
Experiment	**Artifact**	**Technique**	**Non-Industry**	**Industry**	**p-value**	
R3	PG	Checklist	17.5% (5)	10.9% (3)	.064	X
		PBR	17.5% (5)	18% (4)	.949	√
		PBR & Checklist	17.5% (10)	14.4% (7)	.433	X
	ATM	Checklist	9.7% (5)	16.9% (4)	.035	√
		PBR	5.9% (5)	7.4% (3)	.702	√
		PBR & Checklist	7.8% (10)	12.2% (7)	.139	√
	PG & ATM	Checklist	13.6% (10)	13.9% (7)	.909	√
		PBR	11.7% (10)	12.7% (7)	.837	√
		PBR & Checklist	12.7% (20)	13.3% (14)	.808	√

Table 57 – Experience in Design based on Requirements (R3)

			Experience in Design Based on Requirements			
Experiment	Artifact	Technique	Non-Industry	Industry	p-value	
R3	PG	Checklist	15.6% (7)	10.9% (2)	.311	X
		PBR	17.5% (5)	18% (4)	.949	√
		PBR & Checklist	16.4% (12)	15.6% (6)	.85	X
	ATM	Checklist	9.7% (5)	16.9% (4)	.035	√
		PBR	7.3% (7)	4.1% (2)	.473	X
		PBR & Checklist	8.3% (12)	12.6% (6)	.166	√
	PG & ATM	Checklist	13.2% (12)	14.9% (6)	.528	√
		PBR	11.6% (12)	13.3% (6)	.725	√
		PBR & Checklist	12.4% (12)	14.1% (6)	.527	√

Table 58 – Experience as Tester (R3)

			Experience as Tester			
Experiment	Artifact	Technique	Non-Industry	Industry	p-value	
R3	PG	Checklist	17.2% (6)	9.4% (3)	.028	X
		PBR	16.9% (7)	20.3% (2)	.701	√
		PBR & Checklist	17.1% (13)	13.8% (5)	.441	X
	ATM	Checklist	12.7% (7)	13.5% (2)	.871	√
		PBR	5.4% (6)	9% (3)	.366	√
		PBR & Checklist	9.4% (13)	10.8% (5)	.664	√
	PG & ATM	Checklist	14.8% (13)	11% (5)	.185	X
		PBR	11.6% (13)	13.5% (5)	.718	√
		PBR & Checklist	13.2% (26)	12.2% (10)	.749	X

Table 59 – Experience Testing based on Requirements (R3)

			Testing (Requirements) Experience			
Experiment	**Artifact**	**Technique**	**Non-Industry**	**Industry**		
R3	PG	Checklist	Only one subject had industrial experience			
		PBR				
		PBR & Checklist				
	ATM	Checklist				
		PBR				
		PBR & Checklist				
	PG & ATM	Checklist				
		PBR				
		PBR & Checklist				

Table 60 – Experience in PBR Perspective (R3)

			Perspective Experience			
Experiment	**Artifact**	**Technique**	**Non-Industry**	**Industry**	**p-value**	
R3	PG	PBR	14.8% (4)	20% (5)	.434	√
	ATM	PBR	5.9% (5)	7.4% (4)	.702	√
	PG & ATM	PBR	9.9% (9)	14.4% (9)	.332	√

Working Language Experience

Table 61 – Reading Comprehension Level (R3)

			Reading Comprehension			
Experiment	**Artifact**	**Technique**	**Low**	**High**	**p-value**	
R3	PG	Checklist	16.7% (3)	13.5% (6)	.451	X
		PBR	17.7% (3)	17.7% (6)	1	--
		PBR & Checklist	17.2% (6)	15.6% (12)	.705	X
	ATM	Checklist	9% (3)	14.9% (6)	.129	√
		PBR	2.7% (3)	8.6% (6)	.119	√
		PBR & Checklist	5.9% (6)	11.7% (12)	.051	√
	PG & ATM	Checklist	12.8% (6)	14.2% (12)	.621	√
		PBR	10.2% (6)	13.1% (12)	.557	√
		PBR & Checklist	11.5% (12)	13.7% (24)	.436	√

Table 62 – Listening and Speaking (and Writing) Skills (R3)

			Listening and Speaking (and Writing)			
Experiment	**Artifact**	**Technique**	**Low**	**High**		
R3	PG	Checklist	Only one subject who had high experience in Listening and Speaking and Writing English			
		PBR				
		PBR & Checklist				
	ATM	Checklist				
		PBR				
		PBR & Checklist				
	PG & ATM	Checklist				
		PBR				
		PBR & Checklist				

Application Domain Knowledge

Table 63 – Application Domain Knowledge (R3)

			Domain Knowledge			
Experiment	**Artifact**	**Technique**	**Low**	**High**	**p-value**	
R3	PG	Checklist	14.6% (9)	--	--	--
		PBR	20.5% (7)	7.8% (2)	.111	X
		PBR & Checklist	17.2% (16)	7.8% (2)	.116	X
	ATM	Checklist	11.6% (7)	17.6% (2)	.179	√
		PBR	6.6% (9)	--	--	--
		PBR & Checklist	8.8% (16)	17.6% (2)	.051	√
	PG & ATM	Checklist	13.3% (16)	17.6% (2)	.293	√
		PBR	12.7% (16)	7.8% (2)	.512	X
		PBR & Checklist	13% (32)	12.7% (4)	.943	X

Appendix G – Brazilian Results (R4)

This appendix contains the analyzed data from the fourth replication run in Brazil (R4) discussed in Chapter 5. The data is grouped and analyzed based on the background and experience variables defined in Chapter 4. Because each variable is defined by a series of metrics, there is a table of data for each metric. The tables are all in the following format:

Example Table

			Metric			
Experiment	**Artifact**	**Technique**	**Group 1**	**Group 2**	**p-value**	
R4	ATM	Checklist	10.7% (18)	10.3% (2)	.96	X
	PG	PBR	11% (22)	8.6% (2)	.489	X

Where:

- *Experiment* – R4 – first replication in Brazil.
- *Artifact* – The artifact that was inspected by the subjects in the treatment group.
    - PG – Parking Garage
    - ATM – Automated Teller Machine
- *Technique* – The technique used by the subjects in the treatment group.
    - Checklist – A non-procedural list of items to look for in a software artifact during an inspection.
    - PBR – Perspective Based Reading; all 3 perspectives (Tester, User, Designer) were used, but there were not enough subjects to analyze each perspective separately.
- *Group 1/Group 2* – These two columns represent the average percent of the known defects found by the subjects in each of the two treatments represented by the columns. The number in parenthesis is the number of subjects that were in that group.
- *p-value* – The p-value obtained from running a t-test between the two treatment groups
- *√/X* – A '√' indicates that the more experienced group found more defects than the less experienced group, while an 'X' indicates that the less experience group found more defects than the more experience group.

In each table below, shaded rows indicate that the data in that row is an aggregation of data appearing in other rows.

Process Experience

Table 64 – Experience Reviewing Requirements (R4)

			Experience Reviewing Requirements			
Experiment	**Artifact**	**Technique**	**Non-Industry**	**Industry**	**p-value**	
R4	PG	Checklist	--	17.7% (6)	--	--
		PBR	11.7% (4)	6.3% (5)	.033	X
		PBR & Checklist	11.7% (4)	12.5% (11)	.885	√
	ATM	Checklist	20.9% (4)	11.4% (5)	.066	X
		PBR	--	8.1% (6)	--	--
		PBR & Checklist	20.9% (4)	9.6% (11)	.017	X
	PG & ATM	Checklist	20.9% (4)	14.8% (11)	.264	X
		PBR	11.7% (4)	7.3% (11)	.191	X
		PBR & Checklist	16.3% (8)	11% (22)	.134	X

Table 65 – Software Inspection Experience (R4)

			Software Inspection Experience			
Experiment	**Artifact**	**Technique**	**Non-Industry**	**Industry**	**p-value**	
R4	PG	Checklist	18.8% (1)	17.5% (5)	.932	X
		PBR	9.4% (6)	7.3% (3)	.506	X
		PBR & Checklist	10.7% (7)	13.7% (8)	.532	√
	ATM	Checklist	18% (6)	10.8% (3)	.222	X
		PBR	10.8% (1)	7.6% (5)	.755	X
		PBR & Checklist	17% (7)	8.8% (8)	.062	X
	PG & ATM	Checklist	18.1% (7)	15% (8)	.527	X
		PBR	9.6% (7)	7.5% (8)	.494	X
		PBR & Checklist	13.9% (14)	11.2% (16)	.409	X

Software Development Experience

Table 66 – Experience as a Developer (R4)

			Experience as Developer			
Experiment	**Artifact**	**Technique**	**Non-Industry**	**Industry**	**p-value**	
R4	PG	Checklist	3.1% (1)	20.6% (5)	.178	√
		PBR	10.4% (3)	7.8% (6)	.401	X
		PBR & Checklist	8.6% (4)	13.6% (11)	.34	√
	ATM	Checklist	18% (3)	14.4% (6)	.559	X
		PBR	0% (1)	9.7% (5)	.318	√
		PBR & Checklist	13.5% (4)	12.3% (11)	.817	X
	PG & ATM	Checklist	14.3% (4)	17.2% (11)	.599	X
		PBR	7.8% (4)	8.9% (11)	8.4	√
		PBR & Checklist	11.1% (8)	13% (22)	.596	√

Table 67 – Experience Writing Requirements (R4)

			Experience Writing Requirements			
Experiment	**Artifact**	**Technique**	**Non-Industry**	**Industry**	**p-value**	
R4	PG	Checklist	--	17.7% (6)	--	--
		PBR	4.7% (2)	9.8% (7)	.12	√
		PBR & Checklist	4.7% (2)	13.5% (13)	.195	√
	ATM	Checklist	16.2% (2)	15.4% (7)	.913	X
		PBR	--	8.1% (6)	--	--
		PBR & Checklist	16.2% (2)	12.1% (13)	.544	X
	PG & ATM	Checklist	16.2% (2)	16.5% (13)	.97	√
		PBR	4.7% (2)	9% (13)	.334	√
		PBR & Checklist	10.5% (4)	12.8% (26)	.622	√

Table 68 – Experience Writing Use Cases (R4)

			Experience Writing Use Cases			
Experiment	Artifact	Technique	Non-Industry	Industry	p-value	
R4	PG	Checklist	19.5% (4)	14.1% (2)	.633	X
		PBR	8.5% (7)	9.4% (2)	.804	√
		PBR & Checklist	12.5% (11)	11.7% (4)	.885	X
	ATM	Checklist	17.8% (7)	8.1% (2)	.139	X
		PBR	8.8% (4)	6.8% (2)	.805	X
		PBR & Checklist	14.5% (11)	7.4% (4)	.167	X
	PG & ATM	Checklist	18.4% (11)	11.1% (4)	.178	√
		PBR	8.6% (11)	8.1% (4)	.881	X
		PBR & Checklist	13.5% (22)	9.6% (8)	.271	X

Table 69 – Software Design Experience (R4)

			Software Design Experience			
Experiment	Artifact	Technique	Non-Industry	Industry	p-value	
R4	PG	Checklist	10.9% (2)	21.1% (4)	.354	√
		PBR	4.7% (2)	9.8% (7)	.12	√
		PBR & Checklist	7.8% (4)	13.9% (11)	.244	√
	ATM	Checklist	16.2% (2)	15.4% (7)	.913	X
		PBR	5.4% (2)	9.5% (4)	.617	√
		PBR & Checklist	10.8% (4)	13.3% (11)	.642	√
	PG & ATM	Checklist	13.6% (4)	17.5% (11)	.482	√
		PBR	5% (4)	9.7% (11)	.171	√
		PBR & Checklist	9.3% (8)	13.6% (20)	.229	√

Table 70 – Experience in Design Based on Requirements (R4)

			Experience in Design Based on Requirements			
Experiment	Artifact	Technique	Non-Industry	Industry	p-value	
R4	PG	Checklist	13.5% (3)	21.9% (3)	.427	√
		PBR	6.2% (4)	10.6% (5)	.112	√
		PBR & Checklist	9.3% (7)	14.8% (8)	.239	√
	ATM	Checklist	14.9% (4)	16.2% (5)	.819	√
		PBR	4.5% (3)	11.7% (3)	.321	√
		PBR & Checklist	10.4% (7)	14.5% (8)	.375	√
	PG & ATM	Checklist	14.3% (7)	18.3% (8)	.412	√
		PBR	5.5% (7)	11% (8)	.057	√
		PBR & Checklist	9.9% (14)	14.7% (16)	.126	√

Table 71 –Experience as a Tester (R4)

			Experience as Tester			
Experiment	Artifact	Technique	Non-Industry	Industry	p-value	
R4	PG	Checklist	3.1% (1)	20.6% (5)	.178	√
		PBR	9.3% (2)	8.4% (7)	.804	X
		PBR & Checklist	7.3% (3)	13.5% (12)	.283	√
	ATM	Checklist	14.9% (2)	15.8% (7)	.892	√
		PBR	0% (1)	9.7% (5)	.318	√
		PBR & Checklist	9.9% (3)	13.3% (12)	.562	√
	PG & ATM	Checklist	10.9% (3)	17.8% (12)	.258	√
		PBR	6.2% (3)	9% (12)	.475	√
		PBR & Checklist	8.6% (6)	13.4% (24)	.221	√

Table 72 – Experience in Testing (based on Requirements) (R4)

			Testing (Requirements) Experience			
Experiment	**Artifact**	**Technique**	**Non-Industry**	**Industry**	**p-value**	
R4	PG	Checklist	10.9% (2)	21.1% (4)	.354	√
		PBR	10.2% (4)	7.5% (5)	.365	X
		PBR & Checklist	10.4% (6)	13.5% (9)	.518	√
	ATM	Checklist	16.9% (4)	14.6% (5)	.696	X
		PBR	1.4% (2)	11.5% (4)	.156	√
		PBR & Checklist	11.7% (6)	13.2% (9)	.753	√
	PG & ATM	Checklist	14.9% (6)	17.5% (9)	.61	√
		PBR	7.2% (6)	9.3% (9)	.515	√
		PBR & Checklist	11.1% (12)	13.4% (18)	.475	√

Table 73 – Experience in PBR Perspective (R4)

			Perspective Experience			
Experiment	**Artifact**	**Technique**	**Non-Industry**	**Industry**	**p-value**	
R4	PG	PBR	Only 1 subject did not have industrial experience in their perspective.			X
	ATM	PBR				√
	PG & ATM	PBR				√

Working Language Experience

Table 74 – Reading Comprehension Level (R4)

			Reading Comprehension			
Experiment	**Artifact**	**Technique**	**Low**	**High**	**p-value**	
R4	PG	Checklist	--	17.7% (6)	--	--
		PBR	5.2% (3)	10.4% (6)	.063	√
		PBR & Checklist	5.2% (3)	14.1% (12)	.119	√
	ATM	Checklist	9% (3)	18.9% (6)	.074	√
		PBR	--	8.1% (6)	--	--
		PBR & Checklist	9% (3)	13.5% (12)	.437	√
	PG & ATM	Checklist	9% (3)	18.3% (12)	.116	√
		PBR	5.2% (3)	9.3% (12)	.287	√
		PBR & Checklist	7.1% (3)	13.8% (12)	.085	√

Table 75 – Listening and Speaking Skills (R4)

			Listening and Speaking			
Experiment	**Artifact**	**Technique**	**Low**	**High**	**p-value**	
R4	PG	Checklist	15.6% (3)	19.8% (3)	.701	√
		PBR	7.6% (7)	12.5% (2)	.14	√
		PBR & Checklist	10% (10)	16.9% (5)	.157	√
	ATM	Checklist	13.1% (7)	24.3% (2)	.075	√
		PBR	8.1% (3)	8.1% (3)	1	--
		PBR & Checklist	11.6% (10)	14.6% (2)	.547	√
	PG & ATM	Checklist	13.9% (10)	21.6% (2)	.125	√
		PBR	7.7% (10)	9.9% (2)	.518	√
		PBR & Checklist	10.8% (20)	15.7% (4)	.137	√

Table 76 – Writing Skills (R4)

			Writing			
Experiment	**Artifact**	**Technique**	**Low**	**High**	**p-value**	
R4	PG	Checklist	17.2% (2)	18% (4)	.947	√
		PBR	8.9% (6)	8.3% (3)	.87	X
		PBR & Checklist	10.9% (8)	13.8% (7)	.54	√
	ATM	Checklist	17.1% (6)	12.6% (3)	.461	X
		PBR	10.8% (2)	6.8% (4)	.617	X
		PBR & Checklist	15.5% (8)	9.2% (7)	.167	X
	PG & ATM	Checklist	17.1% (8)	15.7% (7)	.769	X
		PBR	9.3% (8)	7.4% (7)	.537	X
		PBR & Checklist	13.2% (16)	11.6% (14)	.597	X

Application Domain Knowledge

Table 77 – Application Domain Knowledge (R4)

			Domain Knowledge			
Experiment	**Artifact**	**Technique**	**Low**	**High**	**p-value**	
R4	PG	Checklist	17.5% (5)	18.8% (1)	.932	√
		PBR	8.1% (5)	9.4% (4)	.677	√
		PBR & Checklist	12.8% (10)	11.3% (5)	.757	X
	ATM	Checklist	15.5% (4)	15.7% (5)	.982	√
		PBR	9.2% (5)	2.7% (1)	.523	X
		PBR & Checklist	12% (9)	13.5% (6)	.753	√
	PG & ATM	Checklist	16.6% (9)	16.2% (6)	.931	X
		PBR	8.6% (9)	8% (6)	.852	X
		PBR & Checklist	12.4% (18)	12.5% (12)	.988	√

Appendix H – CMSC 735 Fall 2001

This appendix contains documents related to Study 1 from Chapter 6, which was run in CMSC 735 Fall 2001. Pages 2-3 of this appendix contain questionnaire that was used to collect the background information on the subjects. Pages 4-5 contain the description of the assignment in which the subjects performed the inspection.

Experience Questionnaire

CMSC 735 Fall 2001

Name ____

General Background

Please estimate your English-language background:
__ I am a native speaker.
__ English is my second language. [Please complete both of the following.]
My reading comprehension skills are:
__ low
__ medium
__ high
My listening and speaking skills are:
__ low
__ medium
__ high

What is your previous experience with software development in practice? (Check the bottom-most item that applies.)
__ I have never developed software.
__ I have developed software on my own.
__ I have developed software as a part of a team, as part of a course.
__ I have developed software as a part of a team, in industry.

Please explain your answer. Include the number of semesters or number of years of relevant experience. (E.g. "I worked for 10 years as a programmer in industry.")

Software Development Experience

Please rate your experience in this section with respect to the following 5-point scale:
1 = none
2 = studied in class or from book
3 = practiced in a class project
4 = used on one project in industry
5 = used on multiple projects in industry

Experience with Requirements

- Experience writing requirements 1 2 3 4 5
- Experience writing use cases 1 2 3 4 5

- Experience reviewing requirements 1 2 3 4 5
- Experience reviewing use cases 1 2 3 4 5
- Experience changing requirements for maintenance 1 2 3 4 5

Experience in Coding

- Experience in coding, based on requirements/use cases 1 2 3 4 5
- Experience in coding, based on design 1 2 3 4 5
- Experience in coding, based on OO design
- Experience in maintenance of code 1 2 3 4 5

Experience in Testing

- Experience in testing software 1 2 3 4 5
- Experience in testing, based on requirements/use cases 1 2 3 4 5
- Experience with equivalence-partition testing 1 2 3 4 5

Other Experience

- Experience with software project management? 1 2 3 4 5
- Experience with software inspections? 1 2 3 4 5

Experience in Problem Domains

We will use answers in this section to understand how familiar you are with various systems we may use as examples or for assignments during the class.

Please rate your experience in this section with respect to the following 3-point scale:
1 = I'm really unfamiliar with the concept. I've never done it.
3 = I've done this a few times, but I'm no expert.
5 = I'm very familiar with this area. I would be very comfortable doing this.

How much do you know about…
- Applying for a loan? 1 3 5
- Applying for a mortgage? 1 3 5
- Using a parking garage? 1 3 5
- Using an ATM? 1 3 5
- Renting movies from a video rental store (e.g. Blockbusters)? 1 3 5

CMSC 735 Fall 2002

Inspection Assignment

In order to give you experience dealing with software processes, in this assignment you are asked to reason about a specific procedure for use in software development. Your team will be given a procedure intended to be used for reviewing requirements documents, and must submit a **report** evaluating this procedure

This assignment consists of three (3) parts:
Assignment 2A –to evaluate PBR using the *Observer-Executor* method taught in class. This part of the assignment must be completed by class time on Wednesday, October 24. The defect list created during the inspection will be due at the beginning of class on Oct. 24.
Assignment 2B – Class discussion of experiences from Assignment 2A. After the class discussion on Oct. 24, team members will switch roles and be given a new requirements document and will repeat the procedure for Assignment 2A.
Assignement 2C – (Described in more detail below) Team members will jointly write a report about your experiences with PBR. More details will be provided later as to the exact contents of the report. Things to keep in mind as you are observing each other include, but are not limited to: How feasible is this procedure? Can it be used for the intended purpose in a practical situation? Would it be worth using in some situations or environments? Which ones? Can it be improved?

Assignment 2C (Detailed)
In this assignment you will write a report, as a team, based on your experiences using PBR in Assignments 2A and 2B. In your report you will need to evaluate the PBR technique that you used.
You will be evaluating a Perspective-Based Reading (PBR) procedure. PBR is a set of procedures for detecting defects in requirements documents; your team will be assigned one procedure from the set. The PBR procedures were covered in class on Oct. 10.
You have been given two different requirements documents to evaluate. Each of you has had the opportunity to play both the role of *Executor* and *Observer*.
As the *Executor*, you applied the PBR procedure with the goal of detecting defects in the given requirements document.
As the *Observer,* your role was to: 1) help guide the Executor through the different steps of the PBR procedure; 2) prompt the Executor for specific feedback about the procedure at certain times; 3) take notes on the Executor's experiences with the procedure in practice; and 4) record defects you see which the *Executor* misses, and add these to the end of the defect list with an indication that they were found by the *Observer* and not by the *Executor*.
Also, during Assignment 2B, in addition to the types of observations you made during assignment 2A, in this assignment you want to observe what effect you think using the same PBR technique in Assignment 2A has on your performance in Assignment 2B.

The notes collected by the Observer will be an important source for your evaluation of the procedure. Other sources may include: the type and quality of defects uncovered by the Executor, the Executor's subjective opinions about the procedure, etc.

Your main goal in writing this report is to draw conclusions about PBR. Make sure that you support the conclusions you draw with experiences observed during your inspections. Your report should address *at least* the following issues, and any other issues you think are interesting and relevant.

1. How feasible is PBR?
2. Is PBR be worth using in a practical situation? Which ones?
3. What would you do to improve PBR?
4. For the team member playing the role of *Observer* in Assignment 2A, how did that experience affect your performance in the role of *Executor* in the second inspection?

For Assignment 2C You Should Turn In:

1. The list of defects found by the Executor. A form for reporting defects will be placed on the class web page.
2. Forms A and B (you may use more than one copy of them).
3. A copy of the notes taken by the Observer. (You should keep the original notes because you will need them to write your report for Assignment 2C.
4. The report described above.

The due date for this assignment is IN CLASS on Nov. 5.

Your grade for Assignment 2 will be based on: the quality of your final report, as determined by the instructor, and how well you conformed to the procedures that you were asked to apply (PBR and the Observer roles). Your grades will NOT depend on your specific answers, e.g. the number of faults that you report, or whether or not you found the techniques valuable.

NOTE: This assignment is part of a study. As always, working with another student will be considered cheating, but for the purposes of the study it is especially crucial that you do not discuss your work with other students in the class. The motivation and design of the study will be discussed in class later this semester.

Appendix I – CMSC 735 Fall 2002

This appendix contains documents related to Study 1 from Chapter 6, which was run in CMSC 735 Fall 2001. Pages 2 this appendix contains version one of the PBR technique, the version with a low level of detail. Pages 3-4 contain version two of the PBR technique, the version with a high level of detail. Pages 5-6 contain the questionnaire that was used to collect the background information on the subjects. Page 7 contains the questionnaire that the subjects using version one of PBR filled out after the study. Page 8 contains the questionnaire that subjects using version two of PBR filled out after the study.

Perspective Based Reading

(Version 1 - High Level)

Perspective based reading is the concept that the various stakeholders of a document should read it to find out if their needs are satisfied by the document. In doing so, it is hoped that the reader will find defects and be able to assess the document from their particular point of view.

Test-Based Reading Technique

For each requirement or functional specification, generate a test case or set of test cases that allow you to ensure that an implementation of the system satisfies the requirement or functional specification.

Inputs: **A set of requirements or functional specifications**

Outputs: **An initial list of test cases**
A list of defects in the requirements

Use any standard testing approach that you are familiar with, to generate a set of test cases incorporating test criteria into the test suite. In doing so, ask yourself the following questions about each test case:

3) Do you have all the information necessary to identify the item being tested and the test criteria? Can you generate a reasonable test case for each item based upon the criteria?
4) Can you be sure that the tests generated will yield the correct values in the correct units?
5) Are there other interpretations of this requirement that the implementer might make based upon the way the requirement or functional specification is defined? Will this affect the tests you generate?
6) Is there another requirement or functional specification for which you would generate a similar test case but would get a contradictory result?
7) Does the requirement or functional specification make sense from what you know about the application or from what is specified in the general description?

Perspective Based Reading

(Version 2 – Low Level)

Reading Technique for Error Guessing

Create statements about "error-prone situations" that allow you to ensure that an implementation of the system will satisfy the requirements and the general description. Follow the procedure below to generate the statements about "error-prone situations" and the associated test cases (input and expected output), and use the questions provided to help identify defects in the requirements.

Inputs: **A set of requirements**
Output: **A list of test cases**
A list of defects that should be fixed in the system

1) Read the general description of the system and elaborate information that you will explore:

- Possible errors or error-prone situations;
- Information that was omitted, either by accident or because the requirements author felt that it was obvious;
- Assumptions that the tester might make when reading the specification.

In doing so, ask yourself the following questions:

Q1.1 Based on your domain knowledge, does the general description make sense?

Q1.2 Has any necessary information been omitted? Has any "obvious", but relevant information been omitted?

Q1.3 Is any information specified that is not needed?

Q1.4 Is it possible for the tester to make multiple interpretations of the given description?

2) Read through the requirements and, using your experience and intuition, for each

requirement, elaborate a the "error-prone situations" that you will explore:

- Possible errors or error-prone situations;
- Information that was omitted, either by accident or because the requirements author felt that it was obvious;
- Assumptions that the tester might make when reading the requirement;
- Information that might be contradictory with another requirement.

In doing so, ask yourself the following questions:

Q2.1 Does the requirement make sense from what you know about the application or from what is specified in the general description?

Q2.2 Has any necessary information been omitted? Has any "obvious", but relevant information been omitted?

Q2.3 Is any information specified that is not needed for this requirement?

Q2.4 Is it possible for the tester to make multiple interpretations of this requirement based on its description?

3) Considering all the “error-prone situations” think about:

- Possible conflicts among the requirements; and
- Redundant information.

Q3.1 Are there multiple requirements for which you would generate similar “error-prone situation” statements but would get a contradictory results (test cases) ?

Q3.2 Are there multiple requirements for which you would generate similar statements, indicating the possibility of the same information having been declared in different requirements? Are they consistent?

Experience Questionnaire

CMSC 735 Fall 2002

Name ____

General Background

Please estimate your English-language background:
__ I am a native speaker.
__ English is my second language. [Please complete both of the following.]
My reading comprehension skills are:
__ low
__ medium
__ high
My listening and speaking skills are:
__ low
__ medium
__ high

What is your previous experience with software development in practice? (Check the bottom-most item that applies.)
__ I have never developed software.
__ I have developed software on my own.
__ I have developed software as a part of a team, as part of a course.
__ I have developed software as a part of a team, in industry one time.
__ I have worked on multiple projects in industry.

Please explain your answer. Include the number of semesters or number of years of relevant experience. (E.g. "I worked for 10 years as a programmer in industry"; "I worked on one large project in industry"; "I developed software as part of a class project"; etc…)

____

____

Software Development Experience

Please rate your experience in this section with respect to the following 5-point scale:
(Please include any relevant comments below each section)
1 = No experience; **2** = learned in class or from book; **3** = used on a class project;
4 = used on one project in industry; **5** = used on multiple projects in industry

Experience with Lifecycle Models

- Experience with the *Waterfall Model* 1 2 3 4 5
- Experience with the *Spiral Model* 1 2 3 4 5
- Experience with the *Incremental Development Model* 1 2 3 4 5

Comments:

Experience with Requirements

- Experience writing requirements 1 2 3 4 5
- Experience interacting with users to write requirements 1 2 3 4 5
- Experience writing use cases 1 2 3 4 5
- Experience reviewing requirements 1 2 3 4 5
- Experience reviewing use cases 1 2 3 4 5
- Experience changing requirements for maintenance 1 2 3 4 5

Comments:

Experience in Design

- Experience in design of systems 1 2 3 4 5
- Experience in design of systems from requirements/use cases 1 2 3 4 5
- Experience with creating Object-Oriented (OO) designs 1 2 3 4 5
- Experience with creating Structured Designs 1 2 3 4 5
- Experience with reading OO designs 1 2 3 4 5
- Experience with the Unified Modeling Language (UML) 1 2 3 4 5
- Experience changing designs for maintenance 1 2 3 4 5

Comments:

Experience in Coding

- Experience in coding, based on requirements/use cases 1 2 3 4 5
- Experience in coding, based on design 1 2 3 4 5
- Experience in coding, based on OO design 1 2 3 4 5
- Experience in maintenance of code 1 2 3 4 5

Comments:

Experience in Testing

- Experience in testing software 1 2 3 4 5
- Experience in testing, based on requirements/use cases 1 2 3 4 5
- Experience with Unit Testing 1 2 3 4 5
- Experience with Integration Testing 1 2 3 4 5
- Experience with System Testing 1 2 3 4 5
- Experience with Acceptance Testing 1 2 3 4 5
- Experience with _______________ (fill in name of a technique 1 2 3 4 5

that you are familiar with)

- Experience with _______________ (fill in name of a technique that you are familiar with) 1 2 3 4 5

Comments:

Other Experience

- Experience with software project management? 1 2 3 4 5
- Experience with software inspections? 1 2 3 4 5

Comments:

Experience in Problem Domains

We will use answers in this section to understand how familiar you are with various systems we may use as examples or for assignments during the class.

Please rate your experience in this section with respect to the following 3-point scale:
1 = I am really unfamiliar with the concept. I have never done it.
3 = I have done this a few times, but I am not an expert.
5 = I am very familiar with this area. I would be very comfortable doing this.

How much do you know about:

- Applying for a loan or mortgage? 1 3 5
- Using a parking garage? 1 3 5
- Using an ATM? 1 3 5
- Renting movies from a video rental store (e.g. Blockbusters)? 1 3 5

Post-study Questionnaire

(Subjects using PBR Version 1 - High Level)

Name: ________________________________

Please note that your answers on this questionnaire will *not* affect your grade in any way. These are questions we need to know in order to make effective use of the data from the study.

1. Training

How effective did you think the training was? Did it help you understand the procedures? Was there anything that was missing or could have been done better?

Describe in detail the method that you used to develop your test cases. If you used a predefined method, give the name of that method.

2. The Techniques

What did you think of the level of detail provided to you in the reading technique? Was there enough information for you to do your job? Was there too much information? Explain.

How would you change the technique to make it more effective?

Did you have any problems using the technique?

Would you use the technique again? Explain.

3. Executing the techniques in inspections

Was there background or training other than what you received in class that you needed to apply PBR?

Were the defect classes well defined? Useful? Complete? Why/Why not?

Did you find defects that you didn't report? Why/Why not?

Post-study Questionnaire

(Subjects using PBR Version 1 - High Level)

Name: __________________________________
Please note that your answers on this questionnaire will *not* affect your grade in any way. These are questions we need to know in order to make effective use of the data from the study.

1. Training

Had you learned about or used Category Partition Testing prior to this class? Explain?

How effective did you think the training was? Did it help you understand the procedures?Was there anything that was missing or could have been done better?

Did you get a clear understanding of the Category Partition Testing technique from the training? What could have been done better? Would you use Category Partition Testing again? Explain.

2. The Techniques

What did you think of the level of detail provided to you in the reading technique? Was there enough information for you to do your job? Was there too much information? Explain.

How would you change the technique to make it more effective?

Did you have any problems using the technique?

Would you use the technique again? Explain.

3. Executing the techniques in inspections

Was there background or training other than what you received in class that you needed to apply PBR?

Were the defect classes well defined? Useful? Complete? Why/Why not?

Did you find defects that you didn't report? Why/Why not?

REFERENCES

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

[Adelson88] Adelson, B. and Soloway, E. “A Model of Software Design.” In *The Nature of Expertise*, Chi, M.H, Glaser, R. and Farr, M. (eds.). Lawrence Erlbaum Associates, 1988.

[Alexander00] Alexander, P.A. “Toward a Model of Academic Development: Schooling and the Acquisition of Knowledge.” *Educational Researcher* 29(2): 28-33,44. March 2000.

[Basili81] Basili, V.R., Weiss, D.M. “Evaluation of a Software Requirements Document By Analysis of Change Data.” *Proceedings of the 5th International Conference on Software Engineering*, 1981, 314-323.

[Basili84] Basili, V.R., and Perricone, B.T. “Software Errors and Complexity: An Empirical Investigation.” *Communications of the ACM*, 27(1), 42-52.

[Basili85] Basili, V.R., “Quantitative Evaluation of Software Engineering Methodology.” *Proceedings of the First Pan Pacific Computer Conference*, Melbourne, Australia, Sept. 1985.

[Basili86] Basili, V.R., Selby, R.W., and Hutchens, D.H. ”Experimentation in Software Engineering.” *IEEE Transactions on Software Engineering* 12(7): 733-743.

[Basili88] Basili, V.R., Rombach, H.D. “The TAME Project: Towards Improvement-Oriented Software Environments.” *IEEE Transactions on Software Engineering*, 14(6): 758-773, June 1998.

[Basili94] Basili, V.R., Caldiera, G., and Rombach, D. “The Experience Factory.” *Encyclopedia of Software Engineering*. Wiley. 1994.

[Basili94b] Basili, V.R., and Green, S. “Software Process Evolution at the SEL.” *IEEE Software*, July 1994, 58-66.

[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.

[Berliner77] Berliner, D.C. and Rosenshine, B. "The Acquisition of Knowledge n the Classroom." In *Schooling and the Acquisition of Knowledge*. Anderson, R.C., Spiro, R.J. and Montague, W.E. (eds.), Lawrence Erlbaum Associates, New Jersey, 1977. pp. 375-396.

[Boehm88] Boehm, B.W. "A Spiral Model of Software Development and Enhancement." *IEEE Computer*, 21(5), May 1988, pp. 61-72.

[Boehm95] Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., and Selby, R. "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0." *Annals of Software Engineering: Special Volume on Software Process and Product Measurements*. Arthur, J.D. and Henry, S.M. (eds.). 1995, Vol. 1, pp. 57-94.

[Boehm99] Boehm, B., Port. D., Abi-Antoun, M., and Egyed, A. "Guidelines for the Life Cycle Objectives (LCO) and the Life Cycle Architecture (LCA) deliverables for Model-Based Architecting and Software Engineering (MBASE)." USC Technical Report USC-CSE-98-519, University of Southern California, Los Angeles, CA, 90089, February 1999.

[Boehm01] Boehm, B.W. and Basili, V.R. "Software Defect Reduction Top 10 List." *IEEE Software* 18(1): 135-137, Jan. 2001.

[Box78] Box, G.E.P., Hunger, W.G., and Hunter, J.S. *Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building*. John Wiley & Sons, 1978.

[Carver03] Carver, J., Shull, F. and Basili. V. "Investigating the Effect of Process Experience on Inspection Effectiveness." University of Maryland Technical Report CS-TR-4442. February, 2003.

[Cheng96] Cheng, B. and Jeffery, R. "Comparing Inspection Strategies for Software Requirements Specifications." In *Proceedings of the 1996 Australian Software Engineering Conference*, p. 203-211.

[Chillarege92] Chillarege, R., Bhandari, I., Chaar, J., Halliday, M., Moebus, D., Ray, B., and Wong, M. "Orthogonal Defect Classification – A Concept for In-Process Measurements." *IEEE Transactions on Software Engineering* 18(11), 943-956.

[Collins77] Collins, A. "Processes in Acquiring Knowledge." In *Schooling and the Acquisition of Knowledge*." Anderson, R.C., Spiro, R.J., and Montague, W.E. (eds.). Lawrence Erlbaum Associates, New Jersey, 1977, pp. 339-363.

[Collins89] Collins, A., Brown, J.S., and Newman, S.E. "Cognitive Apprenticeship: Teaching the Crafts of Reading Writing and Mathematics." In *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser*, L.B. Resnik (ed.), Hillsdale, NJ: Erlbaum, 1989, p. 453-494.

[Cusuamano97] Cusumano, M.A., and Selby, R.W. "How Microsoft Builds Software." *Communications of the ACM*, June 1997, 40(6): 53-61.

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

[Dromey95] Dromey, R.G. "A Model for Software Product Quality." *IEEE Transactions on Software Engineering* 21(2), Feb. 1995, p. 146-162.

[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. "Design and Code Inspections To Reduce Errors in Program Development." *IBM Systems Journal*. 15(3), 1976, 182-211.

[Fagan86] Fagan, M.E. "Advances in Software Inspections." *IEEE Transactions on Software Engineering*. 12(7): 744-751.

[Fenton94] Fenton, N., Pfleeger, S.L., and Glass, R. "Science and Substance: A Challenge to Software Engineers." *IEEE Software*, 11(4): 86-95, July 1994.

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

[Freedman90] Freedman, D.P., and Weinberg, G.M. *Handbook of Walkthroughs, Inspections and Technical Reviews*. Dorset House Publishing, 1990;

[Fusario97] Fusario, P., Lanubile, F., and Visaggio, G. "A Replicated Experiment to Asses Requirements Inspection Techniques." *Empirical Software Engineering: An International Journal* 2(1), 1997. p. 39-57.

[Gervasi00] Gervasi, V, and Nuseibeh, B. "Lightweight Validation of Natural Language Requirements: a case study." In *Proceedings of 4th International Conference on Requirements Engineering*, 2000.

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

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

[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

[IEEE87] IEEE. Software Engineering Standards. IEEE Computer Society Press, 1987.

[Khoshgoftaar96] Khoshgoftaar, T.M., Allen, E.B., Kalaichelvan, K.S., and Goel, N. “Early Quality Prediction: A Case Study in Telecommunications.” *IEEE Software*, January 1996, 65-71.

[Kitchenham95] Kitchenham, B., Pickard, L., and Pfleeger, S.L. “Case Studies for Method and Tool Evaluation.” *IEEE Software* , 12(4): 52-62, July 1995.

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

[Krishan99] Krishan, M.S., and Kellner, M.I. ”Measuring Process Consistency: Implications for Reducing Software Defects.” *IEEE Transactions on Software Engineering*, 25(6): 800-815.

[Laitenberger99] Laitenberber, O., and Atkinson, C., “Generalizing Perspective-based Inspection to Handle Object-Oriented Development Artifacts.” *Proceedings of the 21st International Conference on Software Engineering*, 1999, 494-503.

[Laitenberger00] Laitenberger, O., Atkinson, C., Schlich, M. and El Emam, K. “An Experimental Comparison for Reading Techniques for Defect Detection in UML Design Documents.” *Journal of Systems and Software*, 53(2), August 2000, 183-204.

[Lanubile98] Lanubile, F., Shull, F., Basili, V. R. “Experimenting with Error Abstraction in Requirements Documents.” In *Proc. 5th International Symposium on Software Metrics*. Bethesda, MD., 1998.

[Linger79] Linger, R.C., Mills, H.D., and Witt, B.I. *Structured Programming: Theory and Practice*. The Systems Programming Series. Addison Wesley. 1979.

[Martin90] Martin, J., and Tsai, W.T. “N-Fold Inspection: A Requirements Analysis Technique.” *Communications of the ACM*, 33(2): 225-232, Feb. 1990.

[McClave95] McClave, J.T. and Sincich, T. *A First Course in Statistics, 5th ed.* Prentice Hall, 1995.

[McConnell02] McConnell, S. “Real Quality For Real Engineers.” *IEEE Software* 19(2), March/April 2002. p. 5-7.

[McConnell93] McConnell, S. *Code Complete*. Microsoft Press, 1993.

[Melo2001] Melo, W.; Shull, F. and Travassos, G.H., 2001. Software Review Guidelines. Technical Report ES-556/01. Systems Engineering and Computer Science
Program. COPPE. Federal University of Rio de Janeiro. September. http://www.cos.ufrj.br/publicacoes/reltec/es55601.pdf

[Miller01] Miller, J. “Can results from Software Engineering experiments be safely combined?” *IEEE Metrics Symposium*, London, UK, 2001.

[Offut02] Offutt, J. ”Quality Attributes of Web Software Applications.” *IEEE Software* 19(2), March/April 2002. p. 25-32.

[Ostrand88] Ostrand, T.J. and Balcer, M.J. “The Category Partition Method for Specifying and Generating Functional Tests.” *Communications of the ACM;* 31(6): 676-686. June 1988.

[Parnas85] Parnas, D.L. and Weiss, D.M. “Active Design Reviews: Principles and Practice**.”** *Proceedings of the 8th International Conference on Software Engineering***,** 1985, 132-136.

[Paulk93] Paulk, M.C., Curtis, B., Chrissis, M.B., and Weber, C. “Capability Maturity Model for Software, Version 1.1” *Technical Report CMU/SEI-93-TR-024*, Software Engineering Institute, Pittsburgh, 1993.

[Paulk95] Paulk, M.C. “How the ISO 9001 Compares with the CMM.” *IEEE Software*, 12(1): 74-83. Jan. 1995.

[Pearse95] Pearse, T., and Oman, P. “Maintainability Measurements on Industrial Source Code Maintenance Activities.” *Proceedings of 1995 International Conference on Software Maintenance*, 295-300.

[Pfleeger98] Pfleeger, S. *Software Engineering: Theory And Practice*. Prentice-Hall, 1998.

[Pintrich96] Pintrich, P.R. and Schunk, D.H. *Motivation in Education: Theory Research and Practice*. “Chapter 5: Other Social Cognitive Processes.” Englewood Cliffs: Prentice Hall, 1996. pp.153-197.

[PITAC99] President's Information Technology Advisory Committee, *Information Technology Research: Investing in our Future*, tech. report Nat'l Coordination Office for Computing, Information, and Communications, Washington, D.C., 1999; www.ccic.gov/ac/report.

[Potts94] Potts, C., Takahashi, K., and Anton, A.I. "Inquiry-Based Requirements Analysis." *IEEE Software*, March 1994, 21-32.

[Porter90] Porter, A.A., and Selby, R.S. "Empirically Guided Software Development Using Metric-Based Classification Trees." *IEEE Software*, March 1990, 46-54.

[Porter95] Porter, A.A., Votta, L.G., and Basili, V.R. "Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment." *IEEE Transactions on Software Engineering*, 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*, 23(3): 129-145.

[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* 1(2): 188-204.

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

[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* 1(2), April 1992. p. 188-204.

[Shneiderman98] Shneiderman, B. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley. 1998.

[Shull98] Shull, F. "Developing Techniques for Using Software Documents: A Series of Empirical Studies." PhD Thesis, Computer Science Dept., University of Maryland. 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." *IEEE Transactions on Software Engineering:* 26(11), Nov. 2000.

[Shull00b] Shull, F., Rus, I., and Basili, V.R. “How Perspective-Based Reading Can Improve Requirements Inspections.” *IEEE Computer*, 33(7), July 2000, p. 73-79.

[Shull01] Shull, F., Carver, J., and Travassos, G.H. “An Empirical Methodology for Introducing Software Processes.” In *Proceedings of the 8th European Software Engineering Conference*, Vienna, Austria, Sept. 10-14, 2001. p. 288-296

[Shull03] Shull, F., Carver, J., Travassos, G.H., Maldonado, J.C., Conradi, R. and Basili, V. "Replicated Studies: Building a Body of Knowledge about Software Reading Techniques." To Appear in *Volume on Software Engineering Empirical Methods*, Eds. Natalie Juristo and Ana Moreno, 2003.

[Siegal56] Siegal, S. *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill. 1956.

[Singer96] Singer, J. and Lethbridge, T.C. “Methods for Studying Maintenance Activities.” In *Proceedings of the 1st International Workshop on Empirical Studies of Software Maintenance*, Monterey, CA. 1996.

[Siy96] Siy, H.P. “Identifying the Mechanisms Driving Code Inspection Cost and Benefits.” *Ph.D. Dissertation, University of Maryland*, 1996.

[Soloway88] Soloway, E., Adelson, B., and Ehrlich, K. “Knowledge and Processes in the Comprehension of Computer Programs.” In *The Nature of Expertise*, Chi, M.H, Glaser, R. and Farr, M. (eds.). Lawrence Erlbaum Associates, 1988.

[Takahashi97a] Takahashi, R., Muraoka, Y., and Nakamura, Y. “Building Software Quality Classification Trees: Approach, Experimentation, Evaluation.” *Proceedings of the 8th International Symposium on Software Reliability Engineering*, 222-233.

[Takahashi97b] Takahashi, R. “Software Quality Classification Model Based on McCabe’s Complexity Measure.” *Journal of Systems and Software*, 1997; 38:61-69.

[Travassos99] Travassos, G.H., Shull, F., Fredericks, M., and Basili, V.R. “Detecting Defects in Object Oriented Designs: Using Reading Techniques to Improve Software Quality.” *Proceedings of OOPSLA ’99*, Denver, CO. November, 1999.

[Travassos02] G. Travassos, F. Shull, J. Carver, V. Basili. "Reading Techniques for OO Design Inspections." University of Maryland Technical Report CS-TR-4353. April 2002.

[VanSomeren94] VanSomeren, M.W., Bernard, Y.F., and Sandberg, J.A.C. *The Think Aloud Method: A Practical Guide to Modeling Cognitive Processes*. Academic Press, 1994.

[VanVeenendaal02] VanVeenendaal, E. *The Testing Practitioner*. UTN Publishers; The Netherlands. 2002

[Voas95] Voas, J.M., Miller, K.W. "Software Testability: The New Verification." *IEEE Software* 12(3), May 1995. p. 17-28.

[Votta93] Votta, L.G., Jr. "Does Every Inspection Need a Meeting?" *Proceedings of ACM SIGSOFT '93 Symposium Foundations of Software Engineering*. ACM, Dec. 1993.

[Walston77] Walston, C.E., and Felix, C.P. "A method of programming measurement and estimation." *IBM Systems Journal* 16, No. 1, 54-73. 1977.

[Weller93] Weller, E.F. "Lessons from Three Years of Inspection Data." *IEEE Software* 10(5), September 1993. p. 38-45.

[Yourdon89] Yourdon, E. *Modern Structured Analysis*. Yourdon Press. 1989.

[Zhang99] Zhang, Z., Basili, V.R., and Shneiderman, B. "Perspective-based Usability Inspections: An Empirical Validation of Efficacy." *Empirical Software Engineering: An International Journal*. 4(1), March 1999.

[Zhu02] Zhu, H., Jin, L., Diaper, D., Bai, G. "Software Requirements Validation via Task Analysis." *Journal of Systems and Software*, 61, 145-169.