A Study of Code Inspection Performance and Personality Traits

by

Thomas J.P. Hall

A Thesis
presented to
The University of Guelph

In partial fulfillment of requirements
for the degree of
Master of Science
in
Computer Science

Guelph, Ontario, Canada
© Thomas Hall, April, 2014
ABSTRACT

A STUDY OF CODE INSPECTION PERFORMANCE AND PERSONALITY TRAITS

Thomas Hall
University of Guelph

Advisor:
Dr. Blair Nonnecke

Since their introduction, software inspections have proven to be an effective and cost efficient means of identifying defects and improving software quality. However, performance can vary significantly between inspectors. The influence of personal characteristics – such as personality – on inspection performance is not well understood.

This thesis used regression analysis to investigate whether or not the Big Five personality traits could be used as predictors of software inspection performance. Undergraduate students completed a personality inventory measuring the Big Five personality traits, as well as a code inspection task, using an online software inspection tool. The personality trait scores were used as predictors in a regression analysis of personality and inspection performance.

Results showed that three personality traits – conscientiousness, agreeableness, and extraversion – were statistically significant predictors of inspection performance. The strength of association between inspection performance and each of these traits was modest, indicating that they are not sufficient to predict performance.
ACKNOWLEDGEMENTS

First and foremost I would like to express my deepest appreciation to Dr. Blair Nonnecke, my advisor, for his patience, support, and wisdom. I would also like to thank Dr. Michael Wirth, the other member of my committee, for the time and energy he spent on my thesis.

I must also thank the following people for making this project possible:

- Mike Corbett, for his invaluable assistance with my statistical analysis.
- Dr. Chris Roney, for his help with the Big Five personality traits and with my statistical analysis.
- Sheryl Cantlon, Consuelo Goegan, Rebecca Reimer, and the SOCS administrative staff.
- Cheryl McGrath and the Computer Science department at the University of Western Ontario.
- Gregory Leaver-Fraser, the Computer Science Course Union (CSCU), and the Department of Computer Science at Ryerson University.
- The staff of Evident Solutions, and especially Juha Komulainen, for going above and beyond to ensure that the CodeRemarks system ran smoothly during my experiment, and for adding features to make my life easier.
- The University of Guelph, for supporting me in carrying out my research.

Finally, I’d like to offer my heartfelt thanks to all of the family and friends who have supported me throughout this process. And of course, to Yve, for her limitless positivity and encouragement.
Contents

List of Tables ...................................................................................................................... vi

Chapter 1 Introduction ........................................................................................................ 1
  1.1 Document summary ..................................................................................................... 2

Chapter 2 Background information .................................................................................. 3
  2.1 An overview of software inspection ........................................................................... 3
  2.2 Background in the relevant personality psychology .................................................. 5
    2.2.1 The Myers-Briggs Type Indicator ........................................................................ 5
    2.2.2 Trait theory and the Big Five personality traits .................................................. 6

Chapter 3 Review of the literature ..................................................................................... 9
  3.1 The state of software inspection research .................................................................. 10
    3.1.1 The importance of debugging, with inspections as part of a good strategy .......... 10
    3.1.2 The effectiveness of software inspections ............................................................ 11
    3.1.3 Advances in the software inspection process ....................................................... 12
    3.1.4 The importance of individuals in the effectiveness of inspection ....................... 13
  3.2 A brief history of personality research in software engineering ............................... 15
  3.3 Criticisms of the Myers-Briggs Type Indicator .......................................................... 17
  3.4 Empirical support for the Big Five ............................................................................ 19
  3.5 The Big Five and work/academic performance ......................................................... 21
  3.6 Big Five research in software engineering .................................................................. 22

Chapter 4 The research question ....................................................................................... 25

Chapter 5 Methodology .................................................................................................... 27
  5.1 Participants .................................................................................................................. 27
    5.1.1 Age and gender .................................................................................................... 28
    5.1.2 Degree program ................................................................................................. 28
    5.1.3 Experience .......................................................................................................... 28
  5.2 Materials .................................................................................................................... 29
    5.2.1 Questionnaire and personality inventory ............................................................. 29
    5.2.2 Source code ........................................................................................................ 30
    5.2.3 Code description, API, and instructions ............................................................... 31
    5.2.4 Incentives .......................................................................................................... 31
  5.3 Experimental procedure ............................................................................................ 32
Chapter 6  Results and discussion

6.1 Personality trait and inspection efficiency scores .......................................................... 34
6.2 Simple regression analysis of personality traits and inspection efficiency .................. 35
6.3 Multivariate linear regression analysis ............................................................................ 40
6.4 Traits as predictors of inspection performance .............................................................. 41

Chapter 7  Conclusions and future work .............................................................................. 43

7.1 Conclusions ...................................................................................................................... 43
7.2 Summary of contributions ............................................................................................... 45
7.3 Limitations ........................................................................................................................ 46
7.4 Recommendation for future work .................................................................................. 48
  7.4.1 Why is conscientiousness a negative influence on inspection performance? ......... 48
  7.4.2 Investigate the influence of personality facets as well as the broader traits .......... 48
  7.4.3 Replicating the study with professional inspectors .................................................. 48
  7.4.4 Investigate the influence of personality on inspections of different document types ............................................................. 49
  7.4.5 Constructing a larger model of influences on inspection performance ................ 49

References ............................................................................................................................. 50
Appendices ............................................................................................................................. 58
  Appendix A: CodeRemarks code inspection tool ............................................................... 58
  Appendix B: Participant consent form ............................................................................... 61
  Appendix C: Demographic questionnaire .......................................................................... 63
  Appendix D: IPIP personality inventory ........................................................................... 66
  Appendix E: Source code for the code inspection ............................................................. 68
  Appendix F: Code inspection instructions ........................................................................ 74
  Appendix G: Code description and API .......................................................................... 77
  Appendix H: Recruitment flyer ......................................................................................... 81
List of Tables

TABLE 1: CORRELATIONS BETWEEN THE BIG FIVE PERSONALITY TRAITS AND EFFECTIVENESS IN SOFTWARE TESTING (N=48) .................. 24
TABLE 2: DESCRIPTIVE VARIABLES FOR PERSONALITY TRAIT AND SOFTWARE INSPECTION EFFICIENCY SCORES (N=36) .................. 34
TABLE 3: SIMPLE LINEAR REGRESSION MODELS OF THE RELATION BETWEEN PERSONALITY TRAITS AND SOFTWARE INSPECTION EFFICIENCY SCORES .......................................................................................................................... 35
TABLE 4: SIMPLE LINEAR REGRESSION MODELS OF THE RELATION BETWEEN PERSONALITY TRAITS AND SOFTWARE INSPECTION EFFICIENCY SCORES, CONTROLLED FOR PROFESSIONAL EXPERIENCE AND EXPERIENCE WITH C .......................................................................................................................... 37
TABLE 5: MULTIVARIATE LINEAR REGRESSION MODEL OF THE RELATION BETWEEN BIG FIVE PERSONALITY TRAITS AND SOFTWARE INSPECTION EFFICIENCY SCORES .......................................................................................................................... 40
Chapter 1 Introduction

For nearly thirty years, software inspection has been a highly effective means of improving software quality. First described at IBM (Fagan, 1976), inspections – in which one or more individuals review a software artifact to find defects (such as source code or a design document) – are still the most cost effective means of reducing defects in software projects (Jones, 2008).

Although there has been a good deal of research into inspections over the intervening years, leading to advances such as streamlining the inspection process itself, and confirming their effectiveness in asynchronous and geographically distributed projects, there has been only limited research into what individual attributes and characteristics constitute an effective inspector.

The lack of research on the influence of individuals on the software engineering field is not limited to software inspections; it is an area of study that has been underserved by empirical research for a number of years. One potential influence on individual differences is personality. Most of the work that has been done on the influence of personality in software engineering has used psychometric instruments which are outdated or lacking in empirical support, such as the Myers-Briggs Type Indicator. In these cases, it appears most likely that the instruments were selected based on their popularity in industry, rather than empirical evidence of their efficacy.

While research in personality psychology has converged on the Big Five personality traits, research in software engineering based on Big Five instruments has been sparse. Software inspection research has been similarly underserved by Big Five research. As such, while software inspection research in general has continued to confirm its intrinsic value, there has been little additional insight into what personality traits might constitute an effective inspector.
This thesis attempts to clarify the relationship between personality - measured by an empirically supported Big Five psychometric instrument – and performance in a software inspection task. It will attempt to discover if any of the Big Five traits of personality can be used as predictors of software inspection performance.

1.1 Document summary

The remainder of this document is divided into six sections. Chapter 2 provides background information on software inspections and the relevant personality psychology principles. Chapter 3 reviews the literature on software inspections, the history of personality research in software engineering, criticisms of the Myers-Briggs Type Indicator psychometric instrument, support for the Big Five description of personality, and Big Five research in relevant fields. Chapter 4 summarizes the research question of this thesis. Chapter 5 describes the methodology used in the experiment to answer the research question. Chapter 6 presents the results of the experiment and provides a discussion and analysis of those results. Chapter 7 contains the conclusions, summary of contributions, limitations, and recommendations for future work.
Chapter 2  Background information

This thesis discusses material from both the computer science and personality psychology disciplines. In this chapter, background information on the relevant topics is provided so that readers will be properly equipped to follow the material.

It contains two sections:

- 2.1 An overview of software inspection
- 2.2 Background in the relevant personality psychology

Section 2.1 provides a brief history of software inspections and an overview of the software inspection process. Section 2.2 summarizes the theory behind the Myers-Briggs Type Indicator personality instrument, the Big Five personality traits, and the theoretical differences between the Myers-Briggs Type Indicator, the Big Five, and the differences between type and trait theory in general.

2.1 An overview of software inspection

A software inspection is a form of peer-review used for defect prevention as well as detection (Jones, 2008). Sometimes the term formal technical review is used in the literature in place of inspection, but this thesis will use the latter term.

Formal software inspections were first described by IBM (Fagan, 1976), and can be performed on a wide range of software development artifacts, such as requirements documents, UML diagrams, testing scripts, and source code.

The formal inspection process originally described by Fagan contained five stages (and a follow-up):

- Planning: the inspectors decide which artifacts will be inspected, which participants will take part (usually no more than four), and where the participants will meet.
• **Overview:** The participants are educated on the artifacts being inspected, and roles are assigned to the participants. Roles include a moderator (who manages the inspection team), the inspectors themselves, and the creator of the artifact.

• **Preparation:** The inspectors inspect the artifact(s), on their own, making a note of any possible defects.

• **Inspection meeting:** The team gathers to discuss and finalize a list of defects in the artifact(s).

• **Rework:** The creator(s) of the artifact(s) resolve the list of defects uncovered in the inspection meeting.

• **Follow-up:** The moderator of the inspection verifies that the defects have been resolved in the rework stage.

Software inspections have become a well-known and cost-effective method of verifying computer code in software development (Miller & Yin, 2004). Since first being described, variations on the formal inspection process have emerged. However, most methods tend to have the following features in common (Sauer, Jeffery, Land, & Yetton, 2000):

• They are used to detect and correct defects

• They can be performed at any stage in the development cycle

• They have at least two stages: individual inspection, followed by group inspection

• There are roles; specifically, a moderator, creator, and inspectors

• There is interaction between team members

While inspections can be performed on a number of different software development artifacts, they are especially important for source code, because the potential for defects is higher in coding than in requirements, design, or other documentation (Jones, 2008).
2.2 Background in the relevant personality psychology

2.2.1 The Myers-Briggs Type Indicator

The Myers-Briggs Type Indicator (MBTI) is possibly the most widely known and commercially implemented personality measure (Feldt, Torkar, Angelis, & Samuelsson, 2008; Hannay, Arisholm, Engvik, & Sjoberg, 2010; Pittenger, 1993). It was developed by Isabel Briggs Myers and Katherine Briggs, a mother and daughter, based on Carl Jung’s theory of psychological type. Jung’s theory proposed three dichotomies that could explain differences in human behaviour (Bess & Harvey, 2002; R. McCrae & Costa, 1989; Pittenger, 1993):

**Extraversion (E) vs. Introversion (I):** This dichotomy indicates a preference for the person to look inward to their own thoughts and experiences for reactions to their environment (introversion), or a preference to look outward to other people and the external world (extraversion).

**Sensing (S) vs. Intuition (N):** Indicates a preference for perceiving information that can be sensed, or perceived as real (sensing), or a preference for more intuitive meaning (intuition).

**Thinking (T) vs. Feeling (F):** Indicates a preference to make logical and deductive decisions (thinking), or a preference to make decisions based on emotional reactions and considerations about other people (feeling).

Myers and Briggs further extended Jung’s work by added a fourth dichotomy of their own (R. McCrae & Costa, 1989):

**Judgment (J) vs. Perception (P):** A dichotomy that indicates a preference to have something decided (judgment), or a preference to remain open to new options (perception). This can also be seen as a preference to use the Thinking-Feeling function (judgment) or the Sensing-Intuition function (perception).
Since each of these four dichotomies has two distinct preferences, they can be combined to categorize a person into one of 16 possible types. For example, a person with preferences for extraversion (E), intuition (N), thinking (T), and judgment (J) would be an “ENTJ” type.

According to the MBTI, each of these types is mutually exclusive – quantitatively and qualitatively different from the others – with a person of each type having a distinct preference for how they will behave in any given situation (Bess & Harvey, 2002; R. McCrae & Costa, 1989; Pittenger, 1993). For example, a person of the ENTJ type would be expected to behave distinctly differently than a person of the INTJ type, in spite of sharing preferences in three of the dichotomies. They would also be expected to behave in a similar manner to other people of the same ENTJ type.

This type-based view of personality is actually fairly uncommon in personality psychology, and is conceptually quite different than the trait theory of personality used by most personality tests (Bess & Harvey, 2002; Pittenger, 2005). The next section will discuss some of these differences.

2.2.2 Trait theory and the Big Five personality traits

In personality psychology, trait theory proposes that personality can be measured using a number of mostly orthogonal dimensions, or traits. Different instruments have proposed different numbers of traits, but in recent decades, trait research has converged on the Big Five personality traits (Digman, 1990; R. McCrae & Costa, 1987, 1989). As a result, the Big Five have become widely used in the personality psychology field (Kanij, Merkel, & Grundy, 2013; R. McCrae & Costa, 1989).

The Big Five is so named because it attempts to describe human personality using five broad traits, each with a number of discrete and specific associated facets. For example, the conscientiousness trait includes facets such as self-discipline and dutifulness, while the extraversion trait includes facets such as friendliness and assertiveness.
In contrast to the MBTI, which views each of its dichotomies as mutually exclusive preferences, the Big Five (like most trait-based personality instruments) instead measures each of its dimensions as a continuous variable on a spectrum between two extremes. In this way, a person’s score along the dimension can be seen as reflecting the degree or magnitude to which they exhibit that trait (Pittenger, 2005).

Briefly, the five personality traits are (Digman, 1990; R. McCrae & Costa, 1987, 1989):

**Openness to experience**, sometimes referred to as **intellect**, includes characteristics such as imaginativeness, openness and depth of feeling, curiosity, broadness of interests, and flexibility of thought.

**Conscientiousness** measures characteristics like organization, persistence, will to achieve, and self-discipline.

**Extraversion**: This trait includes sociability, activity, talkativeness, assertiveness, and positive emotion. It’s worth noting that this dimension is not the same as the Jungian extraversion dichotomy described by the MBTI.

**Agreeableness**: On one end of the spectrum, this trait includes altruism, emotional support, sympathy, cooperation, and trust. The other end of the spectrum includes antagonistic concepts such as hostility, jealousy, and spitefulness.

**Neuroticism (or Emotional Stability)** measures the tendency to experience negative emotions such as anxiety, anger, insecurity, or depression.

Unlike the MBTI, the Big Five is not based on any single theory of personality and does not, in itself, offer any theoretical explanation for why these five dimensions are the correct ones (Digman, 1990; R. McCrae & Costa, 1989). Instead, the dimensions were arrived at independently by a number of different
researchers using factor analysis. The theory based on the Big Five is called the Five Factor Model (Digman, 1990).
Chapter 3 Review of the literature

This chapter reviews the literature related to this thesis. It contains six sections:

- 3.1 The state of software inspection research
- 3.2 A brief history of personality research in software engineering
- 3.3 Criticisms of the Myers-Briggs Type Indicator
- 3.4 Empirical support for the Big Five
- 3.5 The Big Five and work/academic performance
- 3.6 Big Five research in software engineering

The first section provides an overview of research into software inspections since they were first introduced by Fagan. This includes research into advancing the inspection process, the effectiveness of inspections, and the influence of individuals on inspection effectiveness.

Section 3.2 offers a brief overview of personality research in the field of software engineering. Section 3.3 provides criticism of the Myers-Briggs Type Indicator, and provides the reasoning for why it is not an appropriate personality instrument for this thesis. Conversely, section 3.4 provide an overview of the empirical support for the Big Five description of personality, and demonstrates why it is a useful personality instrument.

Section 3.5 summarizes the research into the relationship between the Big Five and performance in work and academic tasks. Section 3.6 summarizes Big Five research in software engineering.
3.1 The state of software inspection research

3.1.1 The importance of debugging, with inspections as part of a good strategy

Quality assessment and debugging is a very important part of the software development process, with some estimates indicating as much as 50-60% of the effort involved in producing large software systems falls to these tasks (Osterweil, 1996). In order to reduce the time spent correcting software defects, they should be found and corrected as close to the beginning of the development cycle as possible. If defects are caught early by combining a number of different tools and techniques, development time and cost can be significantly reduced, since a large number of defects are already gone (Jones, 2008). Early research estimated that defects caught early were anywhere from 10-100 times less expensive than those discovered in the last half of the development process (Ackerman, Buchwald, & Lewski, 1989; Fagan, 1976).

A 2009 survey of software inspection literature recommended a best practice for debugging is to combine software inspections with a number of other tools and techniques (Kollanus & Koskinen, 2009). This recommendation appears to be supported by data in the field. In 2008, Capers Jones, a specialist in software engineering methodologies, released a study of data from over 600 companies and 13,000 software development projects. He found that the average defect removal efficiency of software projects in the US combining inspections with other techniques (such as unit testing, regression testing, or external beta testing) was estimated at 85%, higher than using any of these techniques on its own (Jones, 2008).

To underline the importance of debugging, projects averaging 95% defect discovery had the lowest development costs, shortest development schedules, highest customer satisfaction, and highest developer morale. Defect discovery in Jones’ study was calculated using an estimated number of total defects, based on the complexity of the software, as measured by function points.
3.1.2 The effectiveness of software inspections

While it is a recommended best practice for software inspections be combined with other tools and techniques for quality assessment (Ackerman et al., 1989; Jones, 2008), software inspections appear to be a particularly effective quality assurance method. Early detection is critical for reducing development costs, and inspections by their very nature are well equipped for finding problems early. They can be performed as the artifacts are created, before coding even begins, and performed on new code before it has been integrated into the system.

After developing inspections at IBM, Michael Fagan reported that 80-90% of defects uncovered in their software projects were found during formal inspection (Fagan, 1976). In a more specific example, when IBM applied inspections to a large database project, they found the number of defects was reduced over 50% from the previous release, the development schedule was 15% shorter, and customer satisfaction improved. In another IBM business application, inspections were responsible for 93% of the defects discovered (Ackerman et al., 1989).

These successes were certainly not been limited to IBM. Two independent assessments of inspections in the early 1990s found a return on investment of greater than 30 times for every hour given to inspection (Doolan, 1992; Russell, 1991). Later, another assessment found that inspections tend to discover more defects than any competing technology – again up to 90% – and they do it at a lower cost (Glass, 1999). There a number of similar reports from this period.

These findings continue to be true today. In Jones’ wide-ranging study of software development quality assessment, he concluded that “formal design and code inspections are the most effective defect removal activity in the history of software” (Jones, 2008).

Jones defined a metric he called defect removal efficiency, the percentage of potential defects removed before completion of the software. He found most forms of testing average around 30-35% defect
removal efficiency, and seldom top 50%. Inspections, on the other hand, often top 85%, and average 65%. Code inspections in particular can achieve up to 95% defect discovery. Jones’ numbers are consistent with those reported by the earlier case studies (Ackerman et al., 1989; Doolan, 1992; Fagan, 1976; Russell, 1991), indicating that inspections remain as effective today as they were when they were introduced.

Inspection of source code can be especially critical. Fagan reported that more than 80% of defects discovered by inspections at IBM were in found source code, the majority of which were semantic in nature (Fagan, 1976). Semantic defects occur when the code is syntactically correct, but does not function as intended. Similarly, Jones found that the potential for defects is much higher in source code than in other software development artifacts (Jones, 2008).

3.1.3 Advances in the software inspection process

Although the effectiveness of inspection appears to have remained steady over time, there have been a number of changes to the inspection process. The inspections originally described by Fagan were quite rigidly organized (hence the term formal inspections), but since then there has been a significant amount of evidence supporting less formal inspections. Meetings were originally an integral part of formal inspections. Group members were meant to meet at regular intervals to discuss their findings. The primary motivator behind these meetings was synergy, the idea that a group can achieve better results than the members of the group can achieve on their own (Johnson, 1998). However, a number of studies have since found either no appreciable difference in inspections with and without meetings (Perpich, Perry, Porter, Votta, & Wade, 1997; Porter & Johnson, 1997), or that inspections without meetings may actually be preferable (Johnson, 1998; Vitharana & Ramamurthy, 2003). In short, meetings appear to be of “dubious” value (Glass, 1999).
This is a significant development, as meetings can result in hidden costs such as the interval time between meetings (time wasted trying to get everyone booked into a meeting at the same time), and idle time costs incurred by most people in group meetings, as not everyone will be participating at all times (Johnson, 1998). In contrast, inspections where the team members report their findings asynchronously can be more practical (Stein, Riedl, Harner, & Mashayekhi, 1997).

Although formal meetings can be safely dropped from the inspection process, it is still important to have a group stage of some kind where a final list of defects can be agreed upon (Sauer et al., 2000). As long as there is a method in place for the inspectors to agree on a set of defects, it isn’t be necessary to have them agree on this list at the same time or in the same place.

The rise of asynchronous inspection is fortuitous, given that the ubiquity of the internet has enabled software development to become far more geographically and organizationally dispersed than when Fagan first described inspections. Web-based inspection tools have been shown to be at least as effective at discovering and tracking defects, as well as cost-effective for a team to coordinate (Perpich et al., 1997; Stein et al., 1997).

3.1.4 The importance of individuals in the effectiveness of inspection

In a recent survey on the software inspection literature, Kollanus and Koskinen (2009) found that research into the different factors influencing the effectiveness of inspections has been one of the biggest issues. So, the majority of research into the software inspection process has been centred on increasing effectiveness.

Before diving deeper into this topic, it is necessary to define what exactly effectiveness is in the software inspection context. At this point, there is no standardized method, test, or metric for the assessment of inspection effectiveness (Kanij et al., 2013). However, Kollanus and Koskinen identified two important terms in the literature: efficacy and efficiency (Kollanus & Koskinen, 2009). Efficacy refers to the number
of defects discovered in the inspection, while efficiency refers to the number of defects discovered per hour of inspection. Efficiency tends to be the more relevant metric, except in cases where time is not limited (usually in a research setting). In general, then, and for the purposes of this thesis, an increase in effectiveness can be considered an increase in efficiency, unless otherwise noted.

Some of the process factors influencing effectiveness were already covered in the previous section. In addition to process factors, one of the most researched areas of inspection research is individual performance. This makes sense, as the inspections themselves are performed by individual people and not automated in any meaningful way. Humans are inherently important to the process.

The amount and type of material being inspected appears to be a significant factor influencing the effectiveness of an inspection. The type of material can refer to the inherent difficulty of the concepts being presented. Specialized concepts, such as a complicated telecom equipment system, might be more difficult for the average inspector to analyze compared to a more intuitive system such as a program for booking taxis (Wohlin, Aurum, Petersson, Shull, & Ciolkowski, 2002).

The amount of material is also important, so as to avoid overloading the inspectors. Too much material can reduce an inspection’s potential to identify critical defects (Kollanus & Koskinen, 2009; Stein et al., 1997). Fagan himself noted this in a follow-up to his original paper (Fagan, 1986). Conversely, if inspectors aren’t given as much material as they have the capacity to inspect, then the inspection process can take longer than necessary. In a time-sensitive development environment, this is not desirable either.

Research into the ideal amount of material to present to an inspector has led to somewhat varied results. In terms of source code, which is the perhaps the most quantifiable development artifact and the most relevant to this thesis, estimates generally range anywhere from around 100 lines of code per
inspection hour (Glass, 1999) to 200 lines per inspection hour (Dunsmore, 2000), but can also be higher in some cases (Kollanus & Koskinen, 2009).

There is strong agreement among researchers that one of the biggest factors influencing the effectiveness of inspection is individual expertise. This has been demonstrated in a number of studies (Beer & Ramler, 2008; Clark, 2003), with some claiming it is the single most important factor (Hatton, 2008; Sauer et al., 2000). Individual expertise can include factors such as experience with the programming language in question (Knight & Myers, 1993), familiarity with the materials being reviewed, experience in software inspection in general, an “innate talent” for debugging (Pressman & Ince, 1992), or some combination of these. Clark (2003) found that inexperienced inspectors needed more detailed inspection instructions to overcome their lack of experience.

The idea of “innate talent” is an interesting one; if programmers with the same background and the same materials perform at different levels, as Pressman found, what could account for the difference? A survey of professional testers found that the respondents believed “different personal characteristics” (along with experience) contribute significantly to variance in tester performance (Merkel & Kanij, 2010). Kanij et. al (2013) later theorized that personality traits were related to those personal characteristics. Feldt et al. (2008) had earlier argued that personality was one psychometric factor that should be given more weight in software engineering studies.

3.2 A brief history of personality research in software engineering

A number of early theorists noted variations in individual performance in software engineering and considered that these variations might be partly attributable to human factors such as personality (Pressman & Ince, 1992; Shneiderman, 1980; Weinberg, 1971). Shneiderman reported that some programmers outperformed others with similar backgrounds by a factor of as much as 10 to 1. He
proposed that personality variables could play a critical role in the work style of individual programmers. Similarly, Weiner predicted that research into personality could make “a substantial contribution to increased programmer performance” (Weinberg, 1971).

Further to programming, some researchers proposed that different software development tasks may suit different individuals. Performance differences in these tasks could be due in part to differences in individual characteristics, with certain characteristics being better suited to each task. Evidence collected by Bishop-Clark showed some level of support for this idea (Bishop-Clark, 1995). She proposed that the stages of software development included problem representation, design, coding, and debugging/testing, and that more empirical research on human factors – such as personality – needed to be done in order to investigate these relationships. However, empirical research was largely overlooked for a number of years (Feldt et al., 2008).

Experts have proposed a number of characteristics that could suit the debugging/testing stage, which would include software inspection. Capretz and Ahmed analyzed job descriptions for software engineers in a various development roles, and found that good candidates should be methodical, systematic, persistent, able to work independently, and be able to deal with frustrating and emotionally challenging situations (Capretz & Ahmed, 2010). Others have similarly proposed the ability to cope with a monotonous or tedious task (Kanij et al., 2013; Pettichord, 2000), persistence, and the ability to deal with frustration (Bishop-Clark, 1995). However, these proposals have been largely anecdotal.

One empirical method of investigating the relationship between personality characteristics and performance is through psychometric assessment (Feldt et al., 2008). One of the objectives of psychometric assessment is to objectively measure skills, attitudes, personality traits, and achievement.

Da Cunha and Greathead attempted such an investigation by examining code inspection performance and personality, as measured by the Myers-Briggs Type Indicator, or MBTI (Da Cunha & Greathead,
They asked 64 undergraduate computer science students to take a Myers-Briggs Type Indicator (MBTI) personality assessment and then complete a code inspection. There were not enough participants to analyze personality by the full four-letter personality types, so the investigators instead focused on the Sensing-Intuition (S-N) and Thinking-Feeling (T-F) dichotomies. They found that Intuitive-Thinking (NT) types performed significantly better than non-NT types.

Unfortunately, the Da Cunha and Greathead study, like a large number of empirical studies attempted in this area so far, used an outdated psychometric instrument (the MBTI) not suitable for this purpose. In fact, the MBTI has been the most commonly used instrument in studies of personality in software engineering (Bishop-Clark, 1995; Cruz, da Silva, Monteiro, Santos, & dos Santos, 2011; Feldt et al., 2008).

A large number of psychometric instruments exist, such as the Personality and Preference Inventory (PAPI), the Minnesota Multiphasic Personality Inventory (MMPI), the Sixteen Personality Factor Questionnaire (16PF), True Colors, and the Rorschach inkblot test. However, these instruments have not typically been employed in software engineering research.

To understand why the MBTI is not suitable for the purpose of empirical research in software engineering, this thesis will now investigate some of its major criticisms, and investigate why an alternative Big Five measure is preferable.

### 3.3 Criticisms of the Myers-Briggs Type Indicator

The MBTI treats personality as quantifiably and qualitatively different populations of people, mutually exclusive to one another. The primary theory is that each person’s personality fits into one of 16 types, and only one of those types (Bess & Harvey, 2002; R. McCrae & Costa, 1989; Pittenger, 1993, 2005). It should not be surprising, given that the MBTI’s distinct personality types are quite different to trait-based personality instruments that measure personality as a continuous variable between two
extremes, that there exists a great deal of debate among personality psychologists as to whether the distinct types of the MBTI actually exist (Bess & Harvey, 2002).

If the MBTI was correct, we would expect scores in a sample population on any of the dichotomies to be bimodal; two curves with little or no overlap. If this bimodal nature cannot be demonstrated, then the MBTI cannot be correct (R. McCrae & Costa, 1989; Pittenger, 1993).

A large body of evidence shows that the bimodal score distribution implied by the MBTI theory is not present in large populations (Bess & Harvey, 2002; R. McCrae & Costa, 1989; Pittenger, 1993, 2005). Instead, there appears to be a continuous distribution across each of the four dichotomies measured by the MBTI, with most scores falling between the two extremes (Pittenger, 2005). This casts considerable doubt on the theory behind the MBTI. In the MBTI theory, people scoring as E and I types should be qualitatively different from one another, exhibiting distinct preferences in behaviour. If the difference between the scores of different types on each of the four dichotomies can be minimal, as the evidence suggests, then as Pittenger suggested: “the four-letter type formula (of the MBTI) may create the impression that there is meaningful difference between the personality profiles of two individuals when no such difference exists” (Pittenger, 2005).

The distribution of MBTI results, where the majority of scores fall between the two extremes, is instead more in line with what we would expect from a continuous personality measure such as the Big Five. More recent revisions of the MBTI do include a continuous scoring element, however the developers have continued to emphasize the necessity of the dichotomous types (Bess & Harvey, 2002), and have supplied a tie-breaking procedure to avoid intermediate scores (Pittenger, 2005). This insistence on dichotomies has yet more disadvantages from a statistical perspective. In general, using dichotomous scores reduces the predictive power of a continuous scale, increases the risk of Type I errors (where a true null hypothesis is incorrectly rejected), and removes information related to variability (Hunter &
Examining the MBTI in particular, Harvey and Murry found that the dichotomizing procedure of MBTI scoring caused a 26% to 32% loss of information for each of the dichotomies (Harvey & Murry, 1994).

The type theory of the MBTI has not been supported by factor analysis either. In one study, where four factors would have been expected (corresponding to the four dichotomies), the investigators instead found six (Sipps, Alexander, & Friedt, 1985). They also found that the MBTI failed to account for 83% of the variability among the 1,291 participants. In another study, McCrae and Costa found that the Judgment-Perception and Sensing-Intuition dichotomies were correlated with one another (R. McCrae & Costa, 1989), again at odds with MBTI theory.

In short, the literature indicates that there is little empirical support for the theory behind the MBTI. It appears to have become a popular instrument for reasons unrelated to its reliability and validity (Pittenger, 1993, 2005), and personality psychologists in general have been less than enthusiastic about it for some time now (R. McCrae & Costa, 1989).

The MBTI does not appear to be an appropriate psychometric instrument for investigating personality effects in software engineering. It is likely that the software engineering researchers that have used the MBTI (Bishop-Clark, 1995; Cruz et al., 2011; Da Cunha & Greathead, 2007; Feldt et al., 2008) chose to do so because of its popularity.

3.4 Empirical support for the Big Five

In contrast to the MBTI, the body of empirical support for the Big Five dimensions is extensive. Their origins can be traced to five independent investigators who, over a number of years, with a robust set of studies, all came to the independent conclusion that five factors (openness to experience, conscientiousness, extraversion, agreeableness, and neuroticism) could adequately describe the domain of human personality (Digman, 1990). Goldberg argued that not only was the Big Five solution adequate,
but that any model for structuring personality would likely encompass the Big Five dimensions, or something like them (Goldberg, 1981). McCrae and Costa agreed; in a study of self-report data and peer ratings from more than 500 people, they found clear evidence that the Big Five factors, and only those factors, were invariant across observers (McCrae & Costa, 1985).

Adding to the robustness of the Big Five, there is considerable evidence that all of the five factors can be found in different cultures, with different languages, and using different instruments (McCrae & Costa, 1987; Mount & Barrick, 1995).

In Digman’s chapter on the Big Five in the *Annual Review of Psychology*, he concluded:

> “At a minimum, research on the (Big Five) has given us a useful set of very broad dimensions that characterize individual differences. These dimensions can be measured with high reliability and impressive validity. Taken together, they provide a good answer to the question of personality structure.” (Digman, 1990)

However, the Big Five model is not without criticism. One of the primary criticisms is the lack of a compelling explanatory theory for why these five are the correct dimensions, as opposed to some other dimensions (Block, 1995). Block suggested that the Big Five’s exclusively data-driven origins in factor analysis had a potential arbitrariness of solution. He also criticized a number of instruments used to measure the Big Five, citing evidence that the dimensions are not always orthogonal to one another, a point on which Saucier agreed (Saucier, 2002). Non-orthogonal dimensions are not considered problematic in and of themselves, but if the goal is to describe personality in as few dimensions as possible, having non-orthogonal dimensions could indicate some level of redundancy.

Although the Big Five is largely an empirical and not theory-driven finding, some of its dimensions do have conceptual similarities with the much more theory-based MBTI. McCrae and Costa noted these
similarities and found that each of the four MBTI dichotomies showed evidence of convergence with one of the Big Five dimensions (Furnham, 1996; McCrae & Costa, 1989).

While agreement on the Big Five is not unanimous among personality psychology researchers, research on these five dimensions of personality has been approaching something like consensus (Hogan, Hogan, & Roberts, 1996; Mount & Barrick, 1995; Saucier, 2002). They are widely used in the field (McAdams, 2001), with their utility for psychometric applications having been “amply demonstrated” (Saucier, 2002).

3.5 The Big Five and work/academic performance

The Big Five have been employed in a number of studies on work performance. Mount and Barrick (Mount & Barrick, 1995) assessed the predictive validity of the Big Five for performance in five occupational groups (including police, managers, sales, skilled/semi-skilled, and the somewhat more nebulous “professionals”). They found that conscientiousness correlated positively with job performance in all five of the occupations. They later summarized that (Mount & Barrick, 1998):

“Individuals who are dependable, persistent, goal directed and organized tend to be higher performers on virtually any job; those who are careless, irresponsible, low achievement striving and impulsive tend to be lower performers on virtually any job.”

Mount and Barrick also found that extraversion was a valid predictor of job performance for the managers and sales occupations, where employees would be expected to frequently interact with others. They concluded that traits such as sociability, talkativeness, and assertiveness contributed to good performance.

Mount and Barrick continued their investigation of work performance and the Big Five. In a later meta-analysis, they found that neuroticism (or rather its positive pole, emotional stability), agreeableness, and
conscientiousness were positively related to job success (Mount, Barrick, & Stewart, 1998). While they now found that traits other than conscientiousness could be meaningfully related to job performance, the predictive efficiency of those traits was more situationally specific.

Another meta-analysis around the same time found that three other traits, agreeableness, openness to experience, and emotional stability had more validity as predictors of job performance than conscientiousness (Tett, Jackson, & Rothstein, 1991). This was a significantly different finding than Mount and Barrick, a discrepancy that Goldberg called “befuddling” (Goldberg, 1993).

On the whole, the Big Five personality measures appear to be valid predictors of real world performance in virtually all occupations (Hogan et al., 1996). However, while it appears to be a valid predictor, personality’s general effect on job performance appears to be modest (Barrick, Mount, & Judge, 2001), and that the majority of job performance reflects other influences that must be acknowledged as well (Pittenger, 2005).

The Big Five has also shown value in predicting academic achievement. Conscientiousness has been shown to stably predict exam performance, which could make it an especially useful trait for attaining high levels of academic achievement (Chamorro-Premuzic & Furnham, 2003). Conversely, neuroticism appears to be negatively associated with academic achievement (Komarraju, Karau, Schmeck, & Avdic, 2011).

3.6 Big Five research in software engineering

There has been only limited research on software engineering performance using Big Five measures. The results so far have been, on the whole, inconclusive.
Feldt, Angelis, and Samuelsson investigated the relationship of the personality of software professionals and their attitudes towards work (Feldt et al., 2008). They found that differences in personality, and especially the conscientiousness trait, could predict attitudes toward work and work procedures.

A study of personality traits and peer programmer performance (Hannay et al., 2010) found that extraversion had only a modest predictive value on pair programming performance, and that the general predictive effects of personality on performance were not consistent. The investigators suggested that more research effort should be spent on elaborating the personality effects, as personality may manifest itself in longer studies. In another study investigating individual programming performance and the Big Five, no significant effects were found (Darcy & Ma, 2005).

Of particular interest to this thesis are studies of the relationship between personality – measured with a Big Five instrument – and the testing phase of software development, especially inspection. There appears to be only one such study. Assessing that there was “insufficient evidence to make definitive statements on the relationship between personality and testing effectiveness”, Kanij, Merkel, and Grundy assessed the relationship between Big Five traits and performance in a software testing task (Kanij et al., 2013).

The participants in their experiment (48 undergraduate students in Computer Science and Software Engineering programs) were asked to fill out a Big Five personality inventory, examine a specification document, locate any deviations from the specification in the corresponding program (which had been seeded with 20 such deviations), and present their findings in a defect report. The results are presented in Table 1.
Table 1: Correlations between the Big Five personality traits and effectiveness in software testing (N=48)

<table>
<thead>
<tr>
<th>Trait</th>
<th>$O_{sum}$</th>
<th>$O_{wsum}$</th>
<th>Bug location rate</th>
<th>Weighted fault density</th>
<th>Bug report quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neuroticism</td>
<td>.034</td>
<td>.036</td>
<td>-.122</td>
<td>.020</td>
<td>.043</td>
</tr>
<tr>
<td>Extraversion</td>
<td>-.267*</td>
<td>-.267*</td>
<td>.038</td>
<td>-.133</td>
<td>-.265*</td>
</tr>
<tr>
<td>Openness to experience</td>
<td>.161</td>
<td>.165</td>
<td>-.025</td>
<td>-.154</td>
<td>.179</td>
</tr>
<tr>
<td>Agreeableness</td>
<td>.167</td>
<td>.173</td>
<td>-.034</td>
<td>-.215</td>
<td>.191</td>
</tr>
<tr>
<td>Conscientiousness</td>
<td>.026</td>
<td>.026</td>
<td>.251*</td>
<td>-.241*</td>
<td>.028</td>
</tr>
</tbody>
</table>

Notes: Bug Location Rate is the number of defects discovered in comparison to the time taken for the inspection. Weighted Fault Density applies weights to the defects discovered, according to the severity of the defect as defined in (Hutcheson, 2003). Bug Report Quality assesses how well the report produced by the participant matches the IEEE standard of Test Documentation. $O_{sum}$ is a sum of the bug location rate, weighted fault density, and bug report quality metrics. $O_{wsum}$ is similar to $O_{sum}$, but giving slightly more weight to bug report quality.

* Significant at the $p < 0.05$ level (1-tailed)

Of particular interest is that conscientiousness appeared to correlate positively with bug (defect) location rate. This is a similar finding to the results of other studies of performance in other fields. However, the negative relationship between conscientiousness and weighted fault density indicated that the more conscientious participants tended to find less critical defects. Extraversion was the only other trait to demonstrate a significant correlation with performance, principally with bug report quality.

This literature review has described the state of the art in the literature relevant to this thesis. On the whole there is room for improvement in the understanding of the relationship between personality and software engineering performance. This is especially true of the relationship between software testing effectiveness and personality, as the majority of research has focused on team building and peer programming (Cruz et al., 2011).
Chapter 4  The research question

The goal of this thesis is to answer the following research question:

RQ: Can the Big Five personality traits be used as predictors of performance in a code inspection task?

Software inspection is highly effective quality assurance method, and a critical part of the most effective quality assurance strategies. The literature has isolated a number of factors influencing software inspection effectiveness, such as individual expertise and the amount of materials to inspect. Human factors, such as the personality traits of the inspectors, have also been proposed as factors in effectiveness.

However, there has been only limited research into personality effects in software engineering. What little research has been done has mostly used outdated and unsuitable psychometric instruments such as the MBTI.

There are only two studies of personality and software inspection effectiveness in the literature. In the first such study (Kanij et al., 2013), the researchers had their participants (undergraduate students) report on deviations from a specification document. The amount of material presented to the participants was quite high compared to recommendations in the literature (over 1000 lines of Java code), and so the density of defects was quite low (around one defect every 50 program lines of code). The researchers employed a one-tailed significance test for their correlations, and so could only test for an effect in a single direction (positive or negative, but not both).

Inspections are especially important in the coding stage, because coding has the highest potential for defects. The majority of the defects discovered during code inspection are semantic in nature. In the second study of personality and software inspection effectiveness, Da Cunha and Greathead investigated the effects of personality on code inspection performance by providing source code seeded
with semantic defects. The amount of materials provided to the participants was in line with the recommendations in the literature, but the psychometric instrument they employed, the MBTI, lacks empirical support for this purpose, loses important information on variance, and increases the risk of Type I errors, or false positives (Hunter & Schmidt, 1990; Maxwell & Delaney, 1993; Pittenger, 2005). If an empirically supported Big Five instrument can be used to determine personality traits predictive of code inspection performance, this could offer further insight into what makes an effective inspector, and potentially aid in inspection team selection and employment success.
**Chapter 5 Methodology**

This chapter describes the methodology used in the experiment. It contains a description of the participants, materials, and the experimental procedure.

### 5.1 Participants

The participants in the experiment were volunteer undergraduate students enrolled in computer science and software engineering programs at the University of Guelph, the University of Western Ontario, and Ryerson University. These schools were selected for recruitment because their undergraduate curriculums covered the concepts necessary for the participants to understand the code used in the inspection task. Undergraduate students were chosen as participants in this study in an attempt to control for professional experience, and to ensure a relatively level knowledge base of the programming concepts required for the inspection task.

Students at the University of Guelph were recruited following a series of brief talks at the beginning of computer science lectures. Participants from the University of Western Ontario were recruited by email through the CSUS student group’s mailing list. Similarly, participants from Ryerson University were contacted through an email mailing list maintained by the student-run Computer Science Course Union (CSCU). In addition, recruitment flyers (Appendix H) were posted in the University of Guelph’s School of Computer Science, and in the Computer Science department at the University of Western Ontario.

Undergraduate students were chosen as the participants because of their ready availability, and to provide a level of control for professional experience and experience in the programming language used in the inspection, two known influences on inspection performance. Undergraduate students have been used in similar studies (Clark, 2003; Vitharana & Ramamurthy, 2003), and prior research has shown that the number of defects discovered in inspections is similar for both undergraduate students and professionals (Porter & Johnson, 1997).
5.1.1 Age and gender

There were 17 participants from the University of Guelph (14 males, 3 females), 11 from the University of Western Ontario (10 males, 1 female), and 8 from Ryerson University (all male), for a total of 36 (32 males and 4 females).

The participants ranged in age from 18 to 39 years old, with a median age of 22. 34 of the participants were less than 30 years of age.

5.1.2 Degree program

All of the participants were in an undergraduate program. The majority of participants (52.8%) were pursuing a Bachelor of Science degree with a major in Computer Science, a further 14 (38.9%) were pursuing a Bachelor of Computing degree, one participant was pursuing a Bachelor of Engineering in Computer Engineering, and the final two (5.6%) were pursuing other majors (Bioinformatics and Finance), but were enrolled in the requisite Computer Science courses. The majority of participants (55.6%) were in their sixth semester of study or less, with the remainder (44.4%) being in at least their seventh semester of study.

5.1.3 Experience

There were 14 participants (38.9%) that had no professional experience at all in computer programming, with 26 participants (72.2%) having less than a year of professional experience. Eight participants (22.2%) had between one and three years of professional experience, with two participants (5.6%) having more than three years of experience. Seven participants (19.4%) had at least some professional experience with the C programming language, while the remainder (80.6%) had none. However, all of the participants were familiar with C from their undergraduate studies. 31 of the participants (86.1%) rated their skill with C as average or better, while five participants (13.9%) considered their skill in C to be poor.
The majority of participants (52.8%) had never performed a code inspection before. Of the 17 participants who had performed an inspection, four had done so for a school assignment, four had done so in a professional environment, and nine had only informally inspected code (such as looking over a friend’s assignment).

5.2 Materials

The participants in the experiment were provided with a demographic questionnaire, a 100-item personality inventory, the source code for a small C program, and a short document with instructions on how to complete an inspection of the source code.

5.2.1 Questionnaire and personality inventory

All participants were provided with a 22-item demographic questionnaire (Appendix C). The questionnaire was used to establish each participant’s age and educational background, experience in computer programming and code inspections, work habits, and software development preferences. Pilot testing showed that the questionnaire took no more than ten minutes to complete.

In addition, the participants were given a 100-item personality inventory to establish their scores in the Big Five personality traits (Appendix D). There are several instruments intended to measure the Big Five dimensions. Possibly the most comprehensive of these is McCrae and Costa’s NEO Personality Inventory, or NEO-PI-R (McCrae & Costa, 1992). Other widely used instruments include the NEO Five Factor Inventory, or NEO-FFI (McCrae & Costa, 1992), as well as a series of public-domain personality measures called the International Personality Item Pool, or IPIP (Goldberg et al., 2006).

While each of these instruments has empirical support, the IPIP measures are especially useful for psychometric research, as they are public-domain, free, and do not require a licensed assessor (Feldt et al., 2008; Goldberg et al., 2006).
A set of IPIP scales that measures constructs similar to the NEO-PI-R was chosen for this experiment. It contained five sets of 20 questions, one for each of the Big Five traits. For each question, the participant was presented with a statement and asked to indicate how accurately the statement described them, using a five-point Likert scale.

The questionnaire and personality inventory were presented using LimeSurvey, a web-based survey tool, hosted at the University of Guelph.

**5.2.2 Source code**

The source code used for the code inspection was based on the Java program used in the Da Cunha and Greathead experiment (Da Cunha & Greathead, 2007). Since the participants in this study were not all familiar with Java, but were familiar with C, the original program’s logic was converted into a C program (Appendix E).

The functionality of the program consisted of searching for the number of occurrences of a specified string in a given ASCII file. The user could choose to search for exact or partial matches, as well as choose if the search was case sensitive or not. The program would return an error message if the specified ASCII file did not exist, or the search string did not exist in the file. Otherwise, it would return a report on the search results. The logic of the program was of a complexity fairly typical in a small undergraduate coding assignment, using concepts that should have been familiar to all of the participants.

The source code of the program was seeded with 17 known defects. Two of these defects were actually the same issue (arguments passed to a function in the incorrect order) that could be reported either for the function call or in the function definition. The defects were all semantic in nature, with none being based on syntax. In all, the program had a length of 319 lines, including comments. This amount of
inspection material was in line with Dunsmore’s recommendation of 200 lines per hour (Dunsmore, 2000).

For the inspection, the source code was hosted on CodeRemarks (CodeRemarks, 2014), a web-based code inspection tool. Using the CodeRemarks interface, the participants were able to flag potential defects and (optionally) add comments describing their findings. See Appendix A for images and a description of the CodeRemarks system.

5.2.3 Code description, API, and instructions

Participants were provided with a document describing the purpose of the C program, examples of correct output from the program, as well as API for several relevant C functions present in the program (Appendix G). A second document provided participants with instructions for completing the inspection task, reminding them of the time limit, and that they were asked only to make a note of defects they discovered, without having to suggest a fix. Instructions on how to use the CodeRemarks system were also provided (Appendix F).

5.2.4 Incentives

A large non-response rate is possible in surveys, such as in the IPIP NEO-PI-R used in this experiment, but higher response rates, and thus larger sample sizes, can significantly reduce the risk of non-response bias (Groves & Peytcheva, 2008). As such, it was our goal to increase the response rate of our participants to increase our sample size.

The literature has shown that incentives are an effective method of improving response rates in research studies. The increase in response rate is most significant when using financial incentives, and can increase response rates even among participants who may otherwise have little interest in participating (Petrolia & Bhattacharjee, 2009; Ryu, 2006; Teisl, 2006).
However, increasing the response rate with incentives does little to improve a study if incentives can negatively influence the behaviour of the participants. The biggest criticism of incentives, historically, is that they may reduce the intrinsic motivation of the participant – the participant’s desire to complete a task for its own sake.

However, this criticism has little (if any) empirical support (Jenkins Jr., Gupta, Mitra, & Shaw, 1998). Eisenberger and Cameron performed a meta-analysis of over a quarter century of literature on the subject of incentives and performance, reaching the conclusion that “any lessening of intrinsic interest resulting from tangible reward for successful task performance or task completion is too small in magnitude to be detected” (Eisenberger & Cameron, 1996). On the other hand, performance-based incentives have been shown to increase productivity, as well as increase fondness and the expressed interest in the task (Eisenberger & Cameron, 1996; Jenkins Jr. et al., 1998).

So, in an effort to get a large enough sample size for our analysis, we offered a reward of $50 to the three highest performing participants in the code inspection task, determined by the highest number of defects found.

However, this incentive proved to be insufficient. After several months, only 14 participants had completed the experiment. A revised incentive system was implemented, in which participants were given a $20 incentive for completing the study, in addition to the original performance-based rewards. The remaining 22 participants were recruited with this revised incentive system.

**5.3 Experimental procedure**

The participants were asked to complete two tasks, online, on their own time:

- A questionnaire and personality inventory
- An inspection of source code using the CodeRemarks web-based inspection tool
For the first task, participants were sent an email containing a hyperlink to the questionnaire. Immediately following the questionnaire was the 100-item IPIP NEO-PI-R personality inventory. The participants were given unlimited time to complete the questionnaire and inventory, though pilot testing showed that this task would take approximately 30 minutes.

After a participant completed the personality inventory, they were sent a second email containing a hyperlink to the source code on CodeRemarks, as well as the instructions, code description, and relevant API. The participants were given a maximum of 90 minutes to complete their inspection. Any defects entered after the 90 minute time limit were discounted from the analysis.

The inspections were scored by the total number of defects discovered, giving an efficiency score in terms of defects discovered per inspection hour. After all of the participants had completed both tasks, the inspection efficiency scores were compared with each Big Five personality trait score using a simple linear regression analysis.

A simple linear regression is a method of predicting an outcome variable from a single predictor variable, by modeling the data with a straight line. In this case, the predictor variable was the Big Five personality trait score, and the outcome variable was the inspection efficiency (Field, 2009). The error in the model was calculated using the ordinary least squares of the residuals.

The study was conducted over the internet in order to simplify the logistics of including participants from three separate universities. This was considered an acceptable solution because data collection using the internet has been shown to be reliable, valid, and cost effective (Meyerson & Tryon, 2003), including with studies using psychological variables in particular (Krantz, Ballard, & Scher, 1997; Pasveer & Ellard, 1998).
Chapter 6 Results and discussion

This chapter presents the results of the experiment designed to answer the research question of this thesis. It provides analysis and discussion of those results.

6.1 Personality trait and inspection efficiency scores

The minimum, maximum, mean, and standard deviation of the five personality factors and inspection efficiency scores of the participants in the study are shown in Table 2. Each of the personality factors showed a distribution that did not differ significantly from normality, as measured by the Shapiro-Wilk test. The inspection efficiency scores, however, did differ from normality – again measured by the Shapiro-Wilk test – and displayed a high level of positive skew, as well as being moderately leptokurtic.

Table 2: Descriptive variables for personality trait and software inspection efficiency scores (N=36)

<table>
<thead>
<tr>
<th></th>
<th>Minimum</th>
<th>Maximum</th>
<th>Mean</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Openness to experience</td>
<td>52</td>
<td>95</td>
<td>77.47</td>
<td>10.71</td>
</tr>
<tr>
<td>Conscientiousness</td>
<td>50</td>
<td>95</td>
<td>71.22</td>
<td>11.23</td>
</tr>
<tr>
<td>Extraversion</td>
<td>32</td>
<td>98</td>
<td>66.69</td>
<td>14.10</td>
</tr>
<tr>
<td>Agreeableness</td>
<td>53</td>
<td>90</td>
<td>71.31</td>
<td>8.66</td>
</tr>
<tr>
<td>Neuroticism</td>
<td>25</td>
<td>89</td>
<td>49.92</td>
<td>15.71</td>
</tr>
<tr>
<td>Inspection efficiency</td>
<td>0</td>
<td>10</td>
<td>3.55</td>
<td>2.15</td>
</tr>
</tbody>
</table>

Notes: Inspection efficiency was measured in defects discovered per inspection hour.

The trait with the highest standard deviation was neuroticism, which also had a substantially lower mean than the other four traits. Three participants scored less than one defect discovered per inspection hour (including one who discovered none), with the highest performer managing ten defects
discovered per inspection hour. The mean inspection efficiency score was 3.55 defects per inspection hour, with a standard deviation of 2.15.

### 6.2 Simple regression analysis of personality traits and inspection efficiency

The relationship of each of the traits to inspection efficiency was analyzed using a simple linear regression. The results of these analyses can be found in Table 3.

**Table 3: Simple linear regression models of the relation between personality traits and software inspection efficiency scores**

<table>
<thead>
<tr>
<th>Personality Trait</th>
<th>β</th>
<th>t(34)</th>
<th>r²</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Openness to Experience</td>
<td>.019</td>
<td>.110</td>
<td>.000</td>
<td>.012</td>
</tr>
<tr>
<td>Conscientiousness</td>
<td>-.381*</td>
<td>-2.401</td>
<td>.145</td>
<td>5.765</td>
</tr>
<tr>
<td>Extraversion</td>
<td>-.332*</td>
<td>-2.019</td>
<td>.110</td>
<td>4.078</td>
</tr>
<tr>
<td>Agreeableness</td>
<td>-.281</td>
<td>-1.708</td>
<td>.079</td>
<td>2.919</td>
</tr>
<tr>
<td>Neuroticism</td>
<td>.047</td>
<td>.273</td>
<td>.002</td>
<td>.075</td>
</tr>
</tbody>
</table>

*Notes: The dependent variable was software inspection efficiency (defects discovered per inspection hour); predictors were, individually, the five Big Five personality traits listed in the leftmost column. The presence of outliers was assessed using Cook's distance, revealing one influential outlier in the extraversion trait, which was removed for that analysis.

* Significant at the p < 0.05 (two-tailed) level

The openness to experience, agreeableness, and neuroticism personality traits showed no significant ability to predict inspection efficiency at the p < 0.05 level. The sample size for this study was relatively small; in a larger study, these traits may yet demonstrate some use as predictors of inspection efficiency, though the strength of the association (Pearson’s r) is unlikely to be large.

The conscientiousness and extraversion traits did demonstrate a significant relationship with inspection efficiency scores (p < 0.05). The distribution of residuals in the model for these analyses was examined using the Shapiro-Wilk test, and showed no significant difference from normality in either case.
However, in both cases, the distributions demonstrated a moderate amount of positive skew, as well as positive kurtosis. A square root transformation was applied to the inspection efficiency scores, leaving the residual distributions as approximately symmetric (Tabachnick & Fidell, 2007).

From prior research, professional experience and experience in the programming language used in the inspection (in this case, C) are known to be influences on inspection efficiency. While part of the motivation for using undergraduate students in this experiment was to provide some control for these factors, there were nevertheless some differences between participants. Another series of simple regression analyses were carried out, controlling for these additional factors. Using the demographic information supplied by the participants, professional experience was coded as a variable (ProfExp) with a value of 1 if the participant had some professional experience, and a 0 if they did not. Experience in C was likewise coded as a variable (ExpInC) with a value of 1 if the participant had more than five years of experience, and 0 if they did not. Five years was chosen as the maximum amount of experience in C because it’s longer than an undergraduate computer science or software engineering program. The ProfExp and ExpInC variables were included as a block of variables in the regression analyses, with the personality trait in the next block. The results of these analyses are presented in Table 4.
Table 4: Simple linear regression models of the relation between personality traits and software inspection efficiency scores, controlled for professional experience and experience with C

<table>
<thead>
<tr>
<th>Personality Trait</th>
<th>β</th>
<th>t(34)</th>
<th>r²</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Openness to Experience</td>
<td>.078</td>
<td>.412</td>
<td>.005</td>
<td>.170</td>
</tr>
<tr>
<td>Conscientiousness</td>
<td>-.379*</td>
<td>-2.327</td>
<td>.142</td>
<td>5.414</td>
</tr>
<tr>
<td>Extraversion</td>
<td>-.394*</td>
<td>-2.292</td>
<td>.138</td>
<td>5.254</td>
</tr>
<tr>
<td>Agreeableness</td>
<td>-.375*</td>
<td>-2.239</td>
<td>.133</td>
<td>5.011</td>
</tr>
<tr>
<td>Neuroticism</td>
<td>.149</td>
<td>.808</td>
<td>.020</td>
<td>.654</td>
</tr>
</tbody>
</table>

Notes: The dependent variable was software inspection efficiency (defects discovered per inspection hour); predictors were, individually, the five Big Five personality traits listed in the leftmost column. The control variables were professional experience and experience with C. The presence of outliers was assessed using Cook’s distance, revealing no influential outliers.

* Significant at the p < 0.05 (two-tailed) level

The most noteworthy change in the results after controlling for professional experience and experience in C was that the agreeableness trait now showed a significant relationship with inspection efficiency.

The impact on the conscientiousness and extraversion results was minimal, while the openness to experience and neuroticism traits continued to display no significant relationship with inspection efficiency.

As in the earlier analyses, the distribution of residuals in these models was examined using the Shapiro-Wilk test and none showed no significant difference from normality. As before, each distribution demonstrated a moderate amount of positive skew, so a square root transformation was applied, leaving the residual distributions approximately symmetric.

According to Cohen’s guidelines (Cohen, 1988), effect sizes can be interpreted as small (0.10), medium (0.30), or large (0.50). The relationship between conscientiousness and inspection efficiency, measured
using the sample Pearson correlation coefficient, was of a moderate size ($r = -.379$). Conscientiousness was able to account for 14.2% of the variance in inspection efficiency, implying that a much greater share of the variance was influenced by other variables, and further indicating a modest, though still significant, result.

The negative nature of the relationship between conscientiousness and inspection efficiency was a surprise. As conscientiousness increased, inspection efficiency decreased. This is, if anything, the reverse of the findings of a number of previous studies, where conscientiousness has been positively associated with work performance. Although that result has not been uniform, it has been demonstrated in a number of studies, a number of different occupational groups, and has even been shown to relate positively to exam performance.

The anecdotal characteristics associated with good software testers, as provided by experts, such as the ability to be methodical, systematic, persistent, and an ability to deal with tedium (Bishop-Clark, 1995; Capretz & Ahmed, 2010; Pettichord, 2000), also strongly imply higher conscientiousness scores.

In that light, the negative relationship between conscientiousness and inspection effectiveness demonstrated by this study is, to borrow a term from Goldberg, somewhat befuddling. It could be that a conscientious approach to the inspection – thorough, careful, and systematic – was a detriment in this activity, preventing the more conscientious participants from completing their inspection before they reached the time limit. In a study concerned only with efficacy rather than efficiency (and thus having no time limit), conscientiousness may have demonstrated a different effect. Likewise, this may have been true if there was less material to inspect, perhaps closer to the 100 lines of code per hour benchmark (Glass, 1999), rather than the 200 lines recommended by Dunsmore (Dunsmore, 2000).
However, it is unclear to what extent personality traits actually influenced inspection technique, if at all, as that was beyond the scope of this study. The nature of the relationship between conscientiousness and inspection efficiency could certainly stand to be clarified by further study.

Less surprising than the negative relationship between conscientiousness and inspection efficiency was the negative relationship between extraversion and inspection efficiency. The implication is that as extraversion increased (the participant was more extraverted), inspection efficiency decreased. As extraversion decreased (the participant was more introverted), inspection efficiency increased. Like with conscientiousness, the strength of the association was moderate \((r = -.394)\). Extraversion was able to account for 13.8% of the variance in inspection efficiency, again demonstrating that other variables must have a large influence.

The modest, negative relationship between extraversion and inspection efficiency is similar to the earlier result described by (Kanij et al., 2013) and found in Table 1, in which extraversion was negatively correlated with report quality. However, it is not immediately clear why extraversion would have an influence on inspection efficiency. Extraverts are associated with characteristics such as sociability, talkativeness, and assertiveness, so perhaps the solitary nature of the inspection activity provided more intrinsic motivation for the more introverted participants than their more extraverted counterparts.

The agreeableness trait also demonstrated a moderate negative relationship to inspection efficiency \((r = -.375)\), accounting for 13.3% of the variance. There don’t appear to be any results in the software engineering literature specific to the agreeableness trait by which to compare, but both (Tett et al., 1991) and (Mount et al., 1998) found agreeableness to be positively related to job performance in other fields. In that light, this result could be somewhat surprising. However, the latter study noted that the positive relationship between agreeableness and job performance was situationally specific. It appears...
that the task described in this thesis was one such situation where agreeableness did not contribute positively to performance.

### 6.3 Multivariate linear regression analysis

After demonstrating some value as predictors of inspection efficiency scores, the conscientiousness, agreeableness, and extraversion traits were included as predictors in a multivariate regression analysis. The error in the multivariate regression was again calculated using ordinary least squares, and the result can be found in Table 5.

**Table 5: Multivariate linear regression model of the relation between Big Five personality traits and software inspection efficiency scores**

<table>
<thead>
<tr>
<th></th>
<th>Unstandardized Estimates</th>
<th>Standard Error</th>
<th>Standardized Estimates</th>
<th>t</th>
<th>Sig</th>
<th>Tol</th>
<th>VIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>5.368</td>
<td>.993</td>
<td>5.403</td>
<td>.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conscientiousness</td>
<td>-.015</td>
<td>.010</td>
<td>-.258</td>
<td>-1.415</td>
<td>.167</td>
<td>.736</td>
<td>1.359</td>
</tr>
<tr>
<td>Extraversion</td>
<td>-.007</td>
<td>.009</td>
<td>-.149</td>
<td>-0.726</td>
<td>.473</td>
<td>.584</td>
<td>1.714</td>
</tr>
<tr>
<td>Agreeableness</td>
<td>-.020</td>
<td>.013</td>
<td>-.268</td>
<td>-1.529</td>
<td>.137</td>
<td>.800</td>
<td>1.251</td>
</tr>
<tr>
<td>R²</td>
<td>.244</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes: The dependent variable is software inspection efficiency (defects discovered per inspection hour); predictors are conscientiousness, agreeableness, and extraversion. The influence of multivariate outliers was assessed using Cook’s distance, revealing none with a substantial influence.

Like in the simple regression analyses, the distribution of the residuals displayed a moderate positive skew, and so a square root transformation was applied to the inspection efficiency scores. After the
transformation, the distribution was approximately symmetric. The Shapiro-Wilk test was applied, with the results once again indicating that the distribution did not differ significantly from normality.

However, the analysis revealed no significant predictive ability at the $p < 0.05$ level, so the multiple regression model with conscientiousness, agreeableness, and extraversion as predictors did not appear to be effective for this purpose.

### 6.4 Traits as predictors of inspection performance

The research question for this thesis was: “Can the Big Five personality traits be used as predictors of performance in a code inspection task?” The response, in short, appears to be yes. Conscientiousness, agreeableness, and extraversion, individually, demonstrated a significant (though moderate) ability to predict inspection efficiency. However, when included as predictors in a multiple regression analysis, they were unable to demonstrate a significant level of predictive ability. The remaining traits – neuroticism, and openness to experience – demonstrated no value as predictors of inspection efficiency.

These results may help to provide some insight into what constitutes an effective inspector. Although these three predictive traits appear to be only part of a larger set of influences on performance, the exact reasons for differences in performance of inspectors with different conscientiousness, agreeableness, or extraversion scores are not yet clear.

There have been some suggestions in the literature that personality tests could be included in employment or team selection decisions in software engineering (Da Cunha & Greathead, 2007; Kanij et al., 2013). However, the modest predictive power of the conscientiousness (14.2% of variance in efficiency), agreeableness (13.3% of variance) and extraversion (13.8% of variance) traits – and the lack of any significant predictive power of the other traits – suggests that they would have only limited value on their own in terms of informing team selection or employment decisions.
Still, personality traits may yet provide some value for the employers making these decisions. Conscientiousness, agreeableness, and extraversion scores could be included as variables in part of a broader analytics model, along with other known influences on effectiveness, such as experience in the programming language.
Chapter 7 Conclusions and future work

The final chapter of this thesis provides the conclusions that have been reached, a summary of the contributions, and recommendations for future research on this topic.

7.1 Conclusions

As the results of this thesis show, three of the Big Five personality traits showed a significant ability to predict software inspection efficiency scores. This provided an answer to the research question that was posed:

RQ: Can the Big Five personality traits be used as predictors of performance in a code inspection task?

The answer to this question appears to be yes. Three of the Big Five personality traits can be used as predictors of code inspection performance. Through simple linear regression analysis, the conscientiousness, agreeableness, and extraversion traits, individually, showed a significant ability to predict code inspection efficiency, with a moderate strength of association in each case.

The relationship between the conscientiousness trait and inspection performance was negative; a surprising result. Previous studies on conscientiousness have revealed a positive correlation with both academic performance and work performance in a number of different fields. It is unclear why conscientiousness would have a negative relationship with performance in the case of code inspection. It could be that the more conscientious participants were inspecting the code too thoroughly, and were unable to investigate the entire program before they reached the time limit. In a task without a time limit, their performance may have been different. However, it’s not clear to what extent personality traits influenced the inspection technique of the participants. It may be possible to shed more light on this result by including the six facets associated with conscientiousness in the analysis. These facets are self-efficacy, orderliness, dutifulness, achievement-striving, self-discipline, and cautiousness.
The agreeableness trait likewise showed a negative relationship to inspection performance. This result differs from previous studies of job performance in other fields, though in those cases the positive relationships between agreeableness and performance were considered situationally specific. The extraversion trait also demonstrated a negative relationship with inspection performance, which was similar to a finding from a previous study, found in Table 1. These two results suggest that inspectors who are less socially adjusted and communicative performed the inspection somewhat more effectively. Like with the conscientiousness trait, it may be possible to shed more light on these results by including each trait’s associated facets in the analysis. For extraversion, these facets are: friendliness, gregariousness, assertiveness, activity level, excitement-seeking, and cheerfulness. For agreeableness, the facets are: trust, morality, altruism, cooperation, modesty, and sympathy.

This predictive ability of the extraversion, conscientiousness, and agreeableness traits was modest, indicating that a greater share of the variance in inspection performance was due to other influences. When included together in a multivariate linear regression analysis, the conscientiousness, agreeableness, and extraversion traits were unable to demonstrate a significant ability to predict inspection performance. The remaining personality traits – neuroticism, and openness to experience – did not demonstrate a significant ability to predict code inspection performance.

These results indicate that although three of the Big Five personality traits do appear to have some level of predictive value in regards to software inspection performance, they are far from sufficient for this purpose. They may have more value as part of a larger model of performance, taking a number of other known performance influences into account, such as expertise and experience in the programming language.
The role of human factors – such as personality – in inspection and software quality is an area that has not been well studied. This thesis has helped to further the understanding of the influence of personality on software inspection performance.

7.2 Summary of contributions

Empirical research into human factors in software engineering has been largely overlooked for a number of years. The work presented in this thesis contributes to expanding human factor research and improving the quality of personality research in software engineering. The bulk of previous research on the influence of personality in software engineering has used personality instruments that are either outdated or lacking in empirical support. Although experts have long suggested that individual characteristics, such as personality, likely influence the ability of individuals to perform software inspections, there has been only limited research in this area, and even fewer that have used empirically supported personality instruments.

In this thesis, the role of personality traits in software inspection performance was examined using an empirically supported Big Five personality instrument and regression analysis. The contributions of the thesis may be summarized as follows:

- Three personality traits – conscientiousness, agreeableness, and extraversion – were demonstrated to possess a significant ability to predict software inspection efficiency in a simple linear regression, with a moderate strength of association (Pearson's r) in each case. When included in a multivariate linear regression analysis, they were unable to significantly predict inspection performance. In the simple linear regression analyses, the greater share of variance in inspection efficiency was explained by other factors, indicating that the three traits are not sufficient. However, they may have value when included with other influences in a larger model of inspection performance.
• The two remaining personality traits – openness to experience and neuroticism – were shown to have no significant ability to predict inspection efficiency in simple linear regression analyses.

• The relationship between conscientiousness and inspection performance was found to be negative. This is a surprising result, as the trait has been shown to be positively correlated with both academic performance and with work performance fields other than software engineering. It is unclear why this relationship would be negative in the case of software inspection.

• The relationship between extraversion and inspection performance was negative, confirming similar findings in previous research.

• The relationship between agreeableness and inspection performance was also negative, a somewhat unexpected result that differs from earlier research on job performance and personality traits in other fields.

• In general, there is now a greater clarification of the relationship between personality traits and performance in software inspection, an area that had been short of research.

7.3 Limitations

Although efforts were made to increase the sample size of this study by including students from several universities with a similar curriculum, the N of 36 is still on the low side. A lower sample size increases the chances of the occurrence of a Type II error (a failure to reject the null hypothesis when the null hypothesis is false), so it is possible that a significant effect does exist for one or both of the traits – neuroticism and openness to experience – for which no effect was found in this study.

With a statistical power of 0.8 and a significance level of p < 0.05, the sample size in this study could reasonably have been expected to detect an effect of $r = |.090|$ for neuroticism, and $r = |.131|$ for openness to experience. In each case, the effect would be weak, but the fact remains that there may be such effects that were not detected by this study that may be evident with a larger sample size.
In addition, the participants recruited for this study were undergraduate university students, with only limited professional experience. This was done partly in an effort to ensure a relatively level knowledge base for participants in the study. However, it’s possible that the results discovered for these undergraduate students may not represent the population of professional software testers, in which case the results may not generalize to non-research and non-student applications. It could be beneficial to conduct a similar study of personality traits and software inspection performance using professionals to see if the results hold in that case.

Another threat to validity could be the influence of the incentive scheme on the results of the study. It could be that some participants were more or less motivated by the incentive structure in place; for example, the incentives may not have been a sufficient for the less intrinsically motivated participants to apply themselves. Replication of this study with a different incentive scheme could help to examine this potential influence.

There may have been some element in the nature of the C program itself that aided certain participants, perhaps to do with the types of defects presented (entirely semantic), or with the program’s intended purpose. Replication of this study with using different source code for the inspection could shed some light.

Finally, if the Big Five personality traits measured in a study are non-orthogonal, they become less efficient in a multiple-regression analysis. That was not an issue in this study as the traits appeared reasonably orthogonal when measured by VIF, and the Big Five tend to be more orthogonal when the data is generated from self-descriptions as they were in this study, but further studies into personality traits and inspection performance should be aware of the potential threat to validity.
7.4 Recommendation for future work

Although the results presented in this thesis have helped to clarify the relationship between personality traits and performance in a software inspection, this relationship could be further investigated in a number of ways.

7.4.1 Why is conscientiousness a negative influence on inspection performance?

By far the most surprising conclusion of this thesis is that the conscientiousness trait is negatively associated with inspection performance. The conscientiousness trait has been shown to be positively correlated with work performance in a number of other fields, as well as academic performance. It is unclear why it would be a negative influence in the case of software inspection. A more detailed study of inspection technique or planning may offer some insight into this result.

7.4.2 Investigate the influence of personality facets as well as the broader traits

In the Big Five description of personality, each of the five broad traits has a number of more specific facets associated with them. In the instrument used in this study, each trait has an additional six facets. A study with a larger sample size could more reasonably examine the influence of each of these facets to see if any have a significant relationship with inspection performance, even if the broad trait that they’re organized under does not (in the case of openness to experience or neuroticism), or perhaps add further clarity to the significant relationship between performance and conscientiousness, agreeableness, or extraversion.

7.4.3 Replicating the study with professional inspectors

Although inspection performance has been shown to be similar for both university students and professionals, it could be that the influence of personality on the performance of these two groups is in some way different. This potentiality could be examined by repeating this experiment with professionals...
as participants, either confirming or refuting that the results of this thesis can be generalized to the professional realm.

7.4.4 Investigate the influence of personality on inspections of different document types

This thesis investigated the relationship between Big Five personality traits and code inspection performance. Although code inspections are especially important to software quality because the potential for defects is higher, other document types – such as requirements or design documents – may have a different relationship with personality.

7.4.5 Constructing a larger model of influences on inspection performance

The literature has shown that a number of factors, such as experience in the field and knowledge of the programming language being used, can influence an individual’s software inspection performance. To these factors, conscientiousness, agreeableness, and extraversion scores can now be added. It may be possible to construct a larger model of inspection performance using these factors, and perhaps others, to better predict performance.
References


Clark, J. C. (2003). *The impact of background and experience on software inspections* (Doctor of Philosophy). University of Maryland, Maryland, USA.


workshop on Cooperative and human aspects of software engineering (pp. 49–52). Leipzig, Germany: ACM.


doi:10.1109/METRIC.2002.1011331
Appendices

Appendix A: CodeRemarks code inspection tool

The CodeRemarks inspection tool is provided through a website (CodeRemarks, 2014). Creating a new inspection requires only that the source code be pasted into the text box on the front page. After supplying some source code and clicking the “Review” button, a new review (inspection) is generated.

![CodeRemarks inspection tool](image)

*The front page for the CodeRemarks inspection tool, with a short program about to be prepared for inspection (review)*

Each inspection has a unique URL that can be shared with the inspectors. In the case of this thesis experiment, each participant had their own unique inspection, so they could not see each other’s comments. The inspection adds line numbers to the code, and provides text highlights for a number of different programming languages. The language of the inspection is auto-detected, but can also be specified from a drop-down.
A short program to be inspected on CodeRemarks

During the inspection, the inspector(s) can click on any line of code to open a dialog box in which they can add a comment. This can be used to indicate a potential defect in the code, as well as provide an explanation of what the defect is.

```
#include <stdio.h>

main()
{
    printf("Hello World");
}
```

This will never be true

```
do {
    fputs("Case sensitive? [Y/N]: ", stdout); fflush(stdout);
    fgets(choice, sizeof(choice), stdin);
    if (choice != 'Y' || choice == 'n'));
    wordMatch = (choice == 'Y');
}
```

Entering a comment in a CodeRemarks inspection
The comments are saved into the CodeRemarks system as they’re entered, so the inspector does not need to explicitly save the inspection. After the inspection has been completed, it can be exported as a plain text file.

Each comment is given a time stamp for when it was entered, and also tells how long after the inspection was started that the comment was entered. In this thesis experiment, any comments entered after the time limit had expired were disregarded in the analysis.
Appendix B: Participant consent form

You are asked to participate in a research study conducted by MSc. graduate student Thomas Hall and Prof. Blair Nonnecke in the School of Computer Science at the University of Guelph. The results of this study will contribute to the Master’s thesis of Thomas Hall.

If you have any questions or concerns about the research, please feel free to contact Blair Nonnecke at (519) 824-4120 x56407.

Please proceed to the next page in order to view the consent form and begin the questionnaire.

There are 122 questions in this survey.

CONSENT TO PARTICIPATE IN RESEARCH

PURPOSE OF THE STUDY
The study will examine the relationship between personality traits and performance in a code inspection task. Research has suggested that a number of human factors, including personality, may influence code inspection ability.

PROCEDURES
As a participant in the study you will be asked to complete two tasks online. First, you will be asked to complete a questionnaire and personality inventory. The questionnaire will consist of demographic information (name, age, education, etc.) and questions about your education and professional experience in computer programming. The personality inventory consists of 100 multiple choice questions to be answered on a Likert scale, and will attempt to measure your personality using the Five Factor Model (FFM). The FFM measures five personality traits: openness, conscientiousness, extraversion, agreeableness, and emotional stability. This task should take no more than 30 minutes to complete.

The second task will be to complete a code inspection. Using online code inspection software, you will be presented with a section of computer code (approximately 300 lines) and asked to find as many defects as possible. You will not be asked to fix or propose solutions to the defects, only to identify them. You will be given a maximum of 90 minutes to complete this task. Both of these tasks can be completed on your own time. The two tasks can be completed together or at separate times, depending on your preference. However, the demographic questionnaire and personality inventory must be completed before the code inspection can begin.

POTENTIAL BENEFITS TO SOCIETY
Research has shown code inspection to be a very effective means of reducing defects in software, as well as improving the coding skill of the reviewers. This study will attempt to better the understanding of the human factors than can contribute to making some inspectors more effective than others at discovering defects.

PAYMENT
Participants will receive $5 for participating in the first part of the study (the demographic questionnaire and personality inventory). They will receive a further $15 for participating in the second part of the study (the code inspection). Participants will also be eligible for further reward. The three participants (of approximately 50 in total) that discover the most defects in the code inspection task will each receive a $50 bonus. In the event of a tie, the names of the tied participants will be put in a hat. Names will be selected at random until all three bonuses have been awarded. Participants must complete both stages of the study to be eligible for the bonus.

CONFIDENTIALITY
Every effort will be made to ensure the confidentiality of any identifying information obtained in this study. Your data will be identified by the researchers using a unique 15-character coded token. This token will be used to correlate data from the two online tasks that you complete. The information you supply is for research purposes only and will not be published in any way which can be associated with you. The publication of this study will use only aggregated data.
PARTICIPATION AND WITHDRAWAL
Participation in this study is voluntary. If you volunteer to participate in this study, you may withdraw at any time. You may exercise the option of removing your data from the study at any time. Withdrawing participants will still receive $5 if they have participated in at least the first task in the study, and will receive the full $20 if they have participated in both tasks in the study. Participants that do not complete the code inspection will not be eligible for a $50 bonus.

To withdraw from the study, please email thall@uoguelph.ca or nonnecke@uoguelph.ca stating your desire to withdraw. The investigator may withdraw you from this research if circumstances arise that warrant doing so.

RIGHTS OF RESEARCH PARTICIPANTS
You may withdraw your consent at any time and discontinue participation without penalty. You are not waiving any legal claims, rights, or remedies because of your participation in this research study. This study has been reviewed and received ethics clearance through the University of Guelph Ethics Board. If you have questions regarding your rights as a research participant, please contact:
Research Ethics Coordinator
University of Guelph
437 University Centre
Telephone: (519) 824-4120, ext. 56606
E-mail: sauld@uoguelph.ca
Fax: (519) 821-5236

CONSENT TO PARTICIPATE IN RESEARCH
Should you wish to consent to participate in this study, I kindly ask you to answer “yes” at the bottom of this page which indicates that between September 6th, 2012 and November 5th, 2013 you consent to allowing us to use your questionnaire answers, personality test results, and the statistics of your code inspection in our study. If you have any questions or concerns about the research, please feel free to contact Blair Nonnecke at (519) 824-4120 x 56407.

We thank you very much for your cooperation in this matter.
Sincerely,
Thomas Hall
School of Computer Science
University of Guelph
thall@uoguelph.ca

1 [C1]I attest that I am at least 18 years of age, and I consent to have my answers to the demographic questionnaire, personality test, and statistics of completed activities included in the aforementioned research study. *
Please choose only one of the following:
Yes, No
Appendix C: Demographic questionnaire

Instructions
In this questionnaire you will be asked to answer both closed and open questions. The majority of the questions will be closed, and will consist of multiple choice, checkboxes, and rating on a Likert scale. For some of these questions you may be asked to add a few words to briefly elaborate upon your answer. For the open questions, you will be asked to provide a longer, written answer. The clearer and more detailed you are in your response, the more useful the data will be.

1. What is your age?
Please write your answer here:

2. What is your gender?
Please choose only one of the following:
   - Female
   - Male

3. Is English your first language?
Please choose only one of the following:
   - Yes
   - No

4. Which university do you attend?
Please choose only one of the following:
   - University of Guelph
   - Ryerson University
   - University of Western Ontario

5. What is your program of study?
Please choose only one of the following:
   - Bachelor of Computing
   - Bachelor of Science - Computer Science
   - Bachelor of Arts - Computer Science
   - Other

6. What semester of study are you in?
Please choose only one of the following:
   - 1-2
   - 3-4
   - 5-6
   - 7+

7. Please indicate the level that best describes your computer programming knowledge:
Please choose the appropriate response for each item:
   - Before starting university: Very poor, Poor, Average, Good, Expert
   - At your current level of study: Very poor, Poor, Average, Good, Expert

8. Do you have any professional experience in computer programming?
Please choose only one of the following:
   - Yes
• No

9. If you answered "yes" to question 8, how much professional experience do you have?
Please choose only one of the following:
• Less than 6 months
• Between 6 months and 1 year
• Between 1 and 3 years
• More than 3 years

10. If you answered "yes" to question 8, which programming language(s) did you use?
Please choose all that apply:
• Java
• C
• C++
• C#
• VB.net
• Other:

11. If you answered "yes" to question 8, please briefly describe your experience.
Please write your answer here:

12. Please describe your experience with the following programming languages
Please choose the appropriate response for each item:
• C: None, Up to a year, 1 to 3 years, 3 to 5 years, More than 5 years
• C++: None, Up to a year, 1 to 3 years, 3 to 5 years, More than 5 years
• Java: None, Up to a year, 1 to 3 years, 3 to 5 years, More than 5 years
• Other (specify below): None, Up to a year, 1 to 3 years, 3 to 5 years, More than 5 years
• Other (specify below): None, Up to a year, 1 to 3 years, 3 to 5 years, More than 5 years

If you answered "other" to question 12, please specify which language(s), if any.
Please write your answer here:

13. How would you rate your skill with the following programming languages?
Please choose the appropriate response for each item:
• C: Very poor, Poor, Average, Good, Expert
• C++: Very poor, Poor, Average, Good, Expert
• Java: Very poor, Poor, Average, Good, Expert
• Other (specify below): Very poor, Poor, Average, Good, Expert
• Other (specify below): Very poor, Poor, Average, Good, Expert

If you answered "other" to question 13, please specify which language(s), if any.
Please write your answer here:
14. How hard is it for you to find errors in programs:
Please choose the appropriate response for each item:
- Written by yourself? Very hard, Hard, Average, Easy, Very easy
- Written by other people? Very hard, Hard, Average, Easy, Very easy

15. Have you ever performed a code inspection?
Please choose only one of the following:
- Yes
- No

If you answered "yes" to question 15, please describe.
Please write your answer here:

16. How often do you sit at your computer to write a program?
Please choose only one of the following:
- Daily
- A few times per week
- Once a week
- Less than once a week

17. How long (on average) do you sit at your computer in one programming session?
Please choose only one of the following:
- Less than 1 hour
- Between 1 and 3 hours
- Between 3 and 5 hours
- Between 5 and 8 hours
- More than 8 hours

18. How would you rank your preference of the following aspects of computer programming? *
Please number each box in order of preference from 1 to 5
- Specification
- Design
- Coding
- Testing/debugging
- Documentation
Appendix D: IPIP personality inventory

Personality Inventory

On the following pages, there are phrases describing people's behaviors. Please use the rating scale below to describe how accurately each statement describes you. Describe yourself as you generally are now, not as you wish to be in the future. Describe yourself as you honestly see yourself, in relation to other people you know of the same sex as you are, and roughly your same age. So that you can describe yourself in an honest manner, your responses will be kept in absolute confidence. Please read each statement carefully, and then fill in the bubble that corresponds to the number on the scale.

Response Options
1: Very Inaccurate, 2: Moderately Inaccurate, 3: Neither Inaccurate nor Accurate, 4: Moderately Accurate, 5: Very Accurate

[4] Have a good word for everyone.
[8] Do not like art.
[9] Believe that others have good intentions.
[10] Find it difficult to get down to work.
[14] Respect others.
[15] Do just enough work to get by.
[16] Have frequent mood swings.
[17] Am the life of the party.
[18] Do not enjoy going to art museums.
[19] Accept people as they are.
[20] Don't see things through.
[21] Seldom feel blue.
[22] Have little to say.
[23] Believe in the importance of art.
[26] Feel comfortable with myself.
[27] Keep in the background.
[28] Have a vivid imagination.
[29] Cut others to pieces.
[31] Rarely get irritated.
[33] Tend to vote for liberal political candidates.
[34] Suspect hidden motives in others.
[37] Don't like to draw attention to myself.
[38] Carry the conversation to a higher level.
[40] Carry out my plans.
[41] Panic easily.
[42] Know how to captivate people.
[43] Tend to vote for conservative political candidates.
[44] Make people feel at ease.
[45] Shirk my duties.
[47] Start conversations.
[48] Do not like poetry.
[49] Am concerned about others.
[50] Mess things up.
[51] Feel threatened easily.
[52] Warm up quickly to others.
[53] Rarely look for a deeper meaning in things.
[54] Trust what people say.
[55] Leave things unfinished.
[56] Get stressed out easily.
[57] Talk to a lot of different people at parties.
[58] Believe that too much tax money goes to support artists.
[59] Sympathize with others' feelings.
[60] Don't put my mind on the task at hand.
[61] Am very pleased with myself.
[62] Don't talk a lot.
[63] Enjoy hearing new ideas.
[64] Insult people.
[65] Make plans and stick to them.
[66] Am relaxed most of the time.
[67] Avoid contact with others.
[68] Enjoy thinking about things.
[69] Believe that I am better than others.
[70] Complete tasks successfully.
[71] Seldom get mad.
[72] Am hard to get to know.
[73] Can say things beautifully.
[74] Contradict others.
[75] Do things according to a plan.
[76] Am not easily frustrated.
[77] Retreat from others.
[78] Enjoy wild flights of fantasy.
[79] Make demands on others.
[80] Am exacting in my work.
[81] Fear for the worst.
[82] Don't mind being the center of attention.
[83] Am not interested in theoretical discussions.
[84] Am easy to satisfy.
[85] Make a mess of things.
[86] Remain calm under pressure.
[87] Find it difficult to approach others.
[88] Get excited by new ideas.
[89] Hold a grudge.
[90] Finish what I start.
[91] Worry about things.
[92] Cheer people up.
[93] Have difficulty understanding abstract ideas.
[94] Treat all people equally.
[95] Need a push to get started.
[96] Rarely lose my composure.
[97] Keep others at a distance.
[98] Have a rich vocabulary.
[99] Am out for my own personal gain.
[100] Follow through with my plans.
Appendix E: Source code for the code inspection

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAXLENGTH 100
#define PREDELI MLISTLENGTH 10
#define POSTDELI MLISTLENGTH 15
#define BUFSIZE 1024
#define true 1
#define false 0

/* IMPORTANT NOTE AND REMINDER: */
/* ----------------------------- */
/* You do not need to worry about any of the dynamic memory allocation (malloc, realloc, free). */
/* For the purposes of this activity, you can assume that these lines are correct, that the allocation */
/* will not fail (there is no exception handling on these statements), and that efficiency is not a */
/* concern (the allocated memory is not freed). */
/* */
/* This is for the sake of simplicity, as we are more concerned with the program's basic logic and not */
/* with dynamic memory allocation. */
/* */

int isPreDelim(char ch);
int isPostDelim(char ch);
char * getCompletedWord(char * src, int length, int index);
char ** doSearch(char * txt, char * pat, int txtlen, int patlen, int caseSensitive, int wordMatch);

char preDeLimList[PREDELI MLISTLENGTH] = {
    ' ',
    '(',
    '{',
    '[',
    '<',
    '-',
    '/',
    '*',
    '\',
    '\',
    '\',
    '\'};

char postDeLimList[POSTDELI MLISTLENGTH] = {
    ' ',
    ')',
    '}',
    ']',
    '>',
    '-',
    '/',
    '.',
    ':',
    ':'
};
```
int main() {
    char ** lines;
    char *tempString;
    char pattern[MAXLENGTH];
    char filename[MAXLENGTH];
    char choice[MAXLENGTH];
    char c_choice;
    FILE *fp;
    int wordMatch = false;
    int caseSensitive = false;
    int counter = 0;
    int num_lines = 0;

    fputs("Enter (part of) the word to search: ", stdout); fflush(stdout);
    fgets(pattern, sizeof(pattern), stdin);
    if(pattern[strlen(pattern) - 1] == '\n') {
        pattern[strlen(pattern) - 1] = '\0';
    }

    do {
        fputs("Exact word match? [Y/N]: ", stdout); fflush(stdout);
        fgets(choice, sizeof(choice), stdin);
        c_choice = tolower(choice[0]);
    } while(! (c_choice == 'y' || c_choice == 'n'));
    wordMatch = (c_choice == 'Y');

    do{
        fputs("Case sensitive? [Y/N]: ", stdout); fflush(stdout);
        fgets(choice, sizeof(choice), stdin);
        c_choice = tolower(choice[0]);
    } while(! (c_choice == 'y' || c_choice == 'n'));
    caseSensitive = (c_choice == 'y');

    fputs("From which file?: ", stdout); fflush(stdout);
    fgets(filename, sizeof(filename), stdin);
    if(filename[strlen(filename) - 1] == '\n') {
        filename[strlen(filename) - 1] = '\0';
    }

    if((fp = fopen(filename, "r")) == NULL) {
        fprintf(stderr, "can't read from file \\
"%s"!
", filename);
        flush(stderr);
        return(1);
    }

    int n = 1;
    tempString = (char*) malloc(sizeof(char) * BUFSIZE);
    lines = (char**) malloc(sizeof(char*) * BUFSIZE);
    while((fgets(tempString, sizeof(char) * BUFSIZE, fp) == NULL)) {
        if(tempString[strlen(tempString) - 1] == '\n') {
            tempString[strlen(tempString) - 1] = '\0';
        }
        if(n != 1) {
            lines = (char**) realloc(lines, (n) * sizeof(char*));
        }
        lines[n - 1] = tempString;
        n++;
    }

    return(0);
}
lines[n-1] = (char*) malloc(sizeof(char) * strlen(tempString) + 1);
strcpy(lines[n-1], tempString);
n++;
}
num_lines = n - 1;
fclose(fp);

printf("Searching for \"%s\" in file \"%s\"...
", pattern, filename);
fflush(stdout);

int i, j;
int foundPerLine;
char * aLine;
char ** resultsPerLine;
for(i = 0; i <= num_lines; i++) {
aLine = (char *)malloc(sizeof(char) * (strlen(lines[i]) + 1));
strcpy(aLine, lines[i]);
resultsPerLine = (char **)doSearch(aLine, pattern, strlen(aLine),
strlen(pattern), wordMatch, caseSensitive);
foundPerLine = atoi(resultsPerLine[0]);
if(foundPerLine == 0) {
    counter += foundPerLine;
    printf("\tFound %d occurence(s) on line %d", foundPerLine, i+1);
    if(!wordMatch) {
        printf(": "); fflush(stdout);
        char * aWord;
        for(j = 0; j < foundPerLine; j++) {
            printf("%s ", resultsPerLine[j+1]); fflush(stdout);
        }
    }
    printf("\n");
}

printf("\nThe search is case ");
if(caseSensitive) {
    printf("sensitive. ");
} else {
    printf("in-sensitive. ");
}
if(counter > 0) {
    if(wordMatch) {
        printf("The word \"%s\" is found %d times in file \"%s\".
", pattern, counter, filename);
    } else {
        printf("The pattern \"%s\" is found %d times in file \"%s\".
", pattern, counter, filename);
    }
    else {
        printf("Sorry, there is no \"%s\" in file \"%s\".
", pattern, filename);
        fflush(stdout);
    }
}
return 0;
}

int isPreDelim(char ch) {
    int i;
    for (i=1; i<PREDELMILISTLENGTH; i++) {
        if(ch == preDeLimList[i]) {
            return true;
        }
    }
    return false;
}

int isPostDelim(char ch) {
    int i;
    for (i=0; i<POSTDELMILISTLENGTH; i++) {
        if(ch == postDeLimList[i]) {
            return true;
        }
    }
    return false;
}

char * getCompletedWord(char * src, int srclength, int index) {
    int begin = 0;
    int end = srclength-1;
    char * substring;
    char c;

    int i = index;
    while(i > 0) {
        c = src[i-1];
        if(isPreDelim(c)) {
            begin = i;
            break;
        }
        i++;
    }

    i = index;
    while(i < srclength) {
        c = src[i+1];
        if(isPostDelim(c)) {
            end = i;
            break;
        }
        i--;
    }

    substring = (char *)malloc(sizeof(char) * (end - begin + 2));
    for(i = begin; i <= end; i++) {
        substring[i-begin] = src[i];
    }
    substring[end-begin+2] = '\0';
    return(substring);
}
char ** doSearch(char * txt, char * pat, int txtlen, int patlen, int caseSensitive, int wordMatch) {
    int totalFound = 0;
    char ** results;
    char * toCompare;

    results = (char**)malloc(sizeof(char *));
    results[0] = (char *)malloc(MAXLENGTH*sizeof(char));

    int i, j;
    for(i = 0; i < txtlen - patlen; i+=patlen) {
        toCompare = (char *)malloc(sizeof(char) * (patlen + 1));
        for(j = 0; j < patlen; j++) {
            toCompare[j] = txt[j];
        }
        toCompare[patlen] = '\0';

        if(!caseSensitive) {
            for(j = 0; j < patlen; j++)
                { pat[j] = tolower(pat[j]);
                    toCompare[j] = tolower(toCompare[j]);
                }
        }

        if(toCompare == pat) {
            if(wordMatch) {
                int exactMatch = false;
                if(patlen == txtlen) {
                    exactMatch = true;
                }
                else {
                    char pre = (char)0;
                    char post = (char)0;

                    if(i > 0) {
                        pre = txt[i-1];
                    }

                    if(i+patlen < txtlen) {
                        post = txt[i+patlen];
                    }

                    if(i == 0) {
                        if(post != (char)0) {
                            exactMatch = isPostDelim(post);
                        }
                        else {
                            exactMatch = true;
                        }
                    }
                    else if(i+patlen == txtlen) {
                        if(pre != (char)0) {
                            exactMatch = isPreDelim(post);
                        }
                        else {
                            exactMatch = true;
                        }
                    } else
                }
            }
        }
    }
    return totalFound;
}
else {
    if(pre != (char)0 && post != (char)0) {
        if(isPreDelim(pre) &&
            isPostDelim(post)) {
            exactMatch = true;
        }
    } else {
        exactMatch = true;
    }
}

if(exactMatch) {
    totalFound++;
}
else {
    char * completedWord;
    completedWord = getCompletedWord(pat, txtlen, i+1);
    totalFound++;
    results = (char **)realloc(results, sizeof(char *) *
        (totalFound+1));
    results[totalFound] = (char *)malloc(sizeof(char) *
        (strlen(completedWord)+1));
    strcpy(results[totalFound], completedWord);
}

char tempString[MAXLENGTH];
itoa(totalFound, tempString, 10);
results[0] = (char *)realloc(results[0], sizeof(char) * (strlen(tempString) +
    1));
strcpy(results[0], tempString);
return(results);
Appendix F: Code inspection instructions

INSTRUCTIONS ON COMPLETING THE CODE REVIEW TASK

Thank you for participating in this research project. Please read these instructions carefully. Read the whole document before beginning your code review.

If you have any questions about the research, please feel free to contact Dr. Blair Nonnecke at (519) 824-4120 x 56407.

General Instructions

This is a code review. You are to identify as many defects in the code as you can. The code is written in C, and all of the defects will be semantic in nature. Semantic defects are logical errors that cause the program to operate incorrectly but not necessarily terminate or crash. They are valid code that the compiler understands, but does not do what the programmer intended.

Some examples of semantic defects are: a call to the wrong function, the use of an incorrect or uninitialized variable, or putting operations in the wrong order. There will be no syntax defects, such as missing semi-colons or misspelled function names.

In addition, you do not need to worry about any of the dynamic memory allocation in the program. For the sake of this activity, you can assume that those lines of code are correct, that the allocation will not fail (there is no exception handling), and that the memory never needs to be freed. This is for the sake of simplicity, as we are more concerned with examining the program’s basic logic and not with dynamic memory allocation.

You do not need to correct the defects, just make a note of the ones that you find.

You have 1 hour and 30 minutes to complete the code review. This is the maximum time allotted for the review.

This is an individual task. Please do not communicate with other people during the code review, or discuss the review with other participants after you have finished.

Instructions on how to perform the code review will be given on the following pages.

To begin inspecting the code, follow the link provided to you by the researcher. If you did not receive one, please contact thall@uoguelph.ca

HOW TO COMPLETE THE CODE REVIEW USING THE CODEREMARKS TOOL
The code inspection task will be completed online using the CodeRemarks tool. The link provided by the researcher will bring you to the code to be reviewed. It will look something like the screenshot below (showing an example Java program).

Optional: Print the code

If you’d prefer, you can print off the code. You can accomplish this by clicking the “View raw” button. This will show you the code in a text-only, printer friendly format, as shown below.

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

You can then print the code using your browser’s print function. You can mark up the paper however you’d like. If you want to inspect the code on paper rather than on your computer, that’s up to you. However, you will have to flag the defects you’ve found in the CodeRemarks software before you have exceeded the time limit.

Flagging defects

In order to flag a defect, click on the line of code where you found it. This will open up a dialog box, as seen in the example below.
When the dialog box appears, type a nickname in the box labeled “Nick” (this can be anything you want), and then provide a brief description of the defect that you found. For example, if you find that an operation contains a divide by zero error, your comment might read: “Defect caused by a divide by zero”. After you've finished describing the defect, click Save.

You will now see that your comment has been added.

When you’ve finished reviewing all of the code and have added all of your comments, notify the investigator (thall@uoguelph.ca) and they will save your review for analysis. No other participants will see your comments.
Appendix G: Code description and API

File: WordSearch.c

Purpose:
The program searches for occurrences of a string (which can be a word or part of a word) in an ASCII file. Based on the search criteria specified by the user, it prints out a report of the search.

Inputs:
- A string to search for
- A yes/no choice of whether the search should return only exact matches, or partial matches (substrings) as well.
- A yes/no choice of whether or not the search is case-sensitive
- The name of the ASCII file which should be searched

Outputs:
- An error message if:
  - The specified ASCII file does not exist
  - The string being searched for does not exist in the file
- Otherwise, a report is printed to the screen containing the following information:
  - A summary of the search options
  - The line number(s) in the ASCII file where the string being searched for can be found, as well as the number of occurrences on that line. If an exact match was not desired, then for each occurrence of a partial match, the full word containing the partial match will also be printed
  - The total number of occurrences of the string in the file

Examples:

Suppose that the program is supplied with an ASCII file (called “sample.txt”) which contains:

Hold, take my sword. There's husbandry in heaven;
Their candles are all out. Take thee that too.
A heavy summons lies like lead upon me,
And yet I would not sleep: merciful powers,
Restrain in me the cursed thoughts that nature
Gives way to in repose!

The following would be the correct output:
$ WordSearch
Enter (part of) the word to search: the
Exact word match? [Y/N]: n
Case Sensitive? [Y/N]: n
From which file?: sample.txt
Searching for “the” in file “sample.txt”...
   Found 1 occurrence(s) on line 1: There’s
   Found 2 occurrence(s) on line 2: Their thee
   Found 1 occurrence(s) on line 5: the
The search is case in-sensitive. The pattern “the” is found 4 times in file “sample.txt”

$ WordSearch
Enter (part of) the word to search: the
Exact word match? [Y/N]: y
Case Sensitive? [Y/N]: n
From which file?: sample.txt
Searching for “the” in file “sample.txt”...
   Found 1 occurrence(s) on line 5: the
The search is case in-sensitive. The pattern “the” is found 1 times in file “sample.txt”

$ WordSearch
Enter (part of) the word to search: The
Exact word match? [Y/N]: y
Case Sensitive? [Y/N]: y
From which file?: sample.txt
Searching for “The” in file “sample.txt”...
The search is case sensitive.
Sorry, there is no “The” in file “sample.txt”

Relevant C API:

Function: strcpy

char *strcpy(char *restrict s1, const char *restrict s2);

The strcpy() function shall copy the string pointed to by s2 (including the terminating null byte) into the array pointed to by s1. If copying takes place between objects that overlap, the behavior is undefined. The strcpy() function shall return s1; no return value is reserved to indicate an error.

Example:

char string[4];
strcpy(string, “cat”);
Function: strlen

size_t strlen(const char *s);

The strlen() function shall compute the number of bytes in the string to which s points, not including the terminating null byte. The strlen() function shall return the length of s; no return value shall be reserved to indicate an error.

Example:

char string[4];
strcpy(string, “cat”);
int len;
len = strlen(string);

Function: tolower

int tolower(int c);

If the argument of tolower() represents an uppercase letter, and there exists a corresponding lowercase letter, the result shall be the corresponding lowercase letter. All other arguments in the domain are returned unchanged. Upon successful completion, tolower() shall return the lowercase letter corresponding to the argument passed; otherwise, it shall return the argument unchanged.

Function: malloc

void *malloc(size_t size);

The malloc() function shall allocate unused space for an object whose size in bytes is specified by size and whose value is unspecified. Upon successful completion with size not equal to 0, malloc() shall return a pointer to the allocated space.

Function: realloc

void *realloc(void *ptr, size_t size);

The realloc() function shall change the size of the memory object pointed to by ptr to the size specified by size. The contents of the object shall remain unchanged up to the lesser of the new and old sizes. Upon successful completion with a size not equal to 0, realloc() shall return a pointer to the (possibly moved) allocated space.

Sources

The above API was sourced from The Open Group Base Specifications Issue 6,
Undergrad Comp Sci students needed for a research study

Participants can earn anywhere from $20 to $70!

The details:

We are looking for undergraduate computer science students (2\textsuperscript{nd} year +) to participate in a study conducted by MSc. graduate student Thomas Hall and Professor Blair Nonnecke from the University of Guelph.

Your participation will take place entirely online and on your own time.

You will receive $20 for your participation in the study, and based on your performance, you could also earn a $50 bonus*. All you have to do is:

- Complete a small (confidential) demographic questionnaire
- Fill out a (confidential) multiple-choice personality questionnaire
- Inspect the source code of a small (300 lines) C program for errors (you don’t have to fix them)

*The three participants that find the most errors in the source code will each receive a gift card. There will be a maximum of 50 participants in the study.

To participate in this study (or for more information), please contact us at thall@uoguelph.ca