# "The Myers Briggs Personality Type as a Predictor of Success in the Code-Review Task."


A Thesis Submitted to the University of Newcastle upon Tyne for the Degree of MPhil in Psychology

By

Alessandra Devito da Cunha

School of Biology - Psychology

Faculty of Science, Agriculture and Engineering

December - 2003

**ABSTRACT**

This MPhil project is part of an interdisciplinary research project "Interdisciplinary Research Collaboration on Dependability (DIRC); and the aim was to identify which personality factors influence the performance of code-reviewers (or bug-finding) in software production. DIRC is interested in finding solutions and techniques to develop reliable and dependable software and systems and for this a study of the personality factors involved in the software review was necessary, so that when software is produced by the most appropriate professional it would be done in less time and the cost would be reduced.

Abilities and knowledge of second year students from Newcastle University were measured through a self-rating questionnaire. Then a code-review exercise of a Java program (with bugs added) was performed. Finally the students completed the Myers Briggs Personality Type Indicator (MBTI). The data were analysed and it was found that students with "Intuitive" preference had a higher performance in code-reviews than the non-intuitive students.

This finding prompts the question of whether other parts of programming might also be under the influence of other personality types, as each part of the programming process has specific characteristics and requires specific qualities of a programmer.

## ACKNOWLEDGMENTS

First of all I would like to tanks my husband, Rodrigo, for his support and companionship. I also would like to acknowledge two other people who helped make this project possible: Dr. Cristina Gacek for her support and friendship and for introducing me to DIRC and people involved there, and Professor Cliff B Jones for his encouragement, and for helping me secure DIRC funding.

To Dr. George Erdos, who accepted me under his supervision and shared some thoughts on this theme, my sincere thanks. Special thanks also to David Greathead, for his help with the collection and analysis of data, and language checking, and Dr. Budi Arief, who wrote the code and kindly explained it to me many times. I would also like to express my gratitude to Dr. Denis Besnard, for technical support with the word processor; and to Tony Lawrie, Dr. Jeremy Bryans and Joey Coleman, for sharing their thoughts and ideas.

I am grateful to Dr. Chris Philips and to Professor Pete Lee of the School of Computing Science, who kindly allowed me to use some of their lecture slots to give the tasks to the students. Although the data was not included in this current research, I would like to thank Prof. Peter Ayton, Lorenzo Stringini and Peter Popov for allowing me to visit City University and repeat the experiments there.

The valuable comments from my examiners Prof. Peter Ayton and Peter Andras enabled me to clarify several points of this final version of my dissertation and I am very grateful to them.

This research would not have been done without the students' participation. I really appreciate those who turned up in the three tasks, making it possible to correlate the data.

Finally, I would like to acknowledge my friends and family, especially my parents, for their encouragement and support.

## TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

## LIST OF APPENDIX

# 1 INTRODUCTION

This research is part of a Project Activity related to computer-based systems called 'Effective Collaboration in the Design of Dependable Software' which is in turn part of a major interdisciplinary research project called **DIRC -** *Interdisciplinary Research Collaboration in Dependability*. Therefore, before explaining the purpose of this research, a brief outline of DIRC and the relevant Project Activity on 'Collaboration in Dependable Software' is given.

The most important aspiration of DIRC is to improve the dependability of computer-based systems, in other words its objectives cover branches in reliability, security, and availability in human-computer interactions. This is being tackled using an interdisciplinary approach which includes computer scientists, statisticians, sociologists and psychologists. These professionals are distributed around five universities including Newcastle University which is DIRC's coordinating site (DIRC's webpage: http://www.dirc.org.uk/).

As a way of covering the major problems in computer-based systems and of seeking a better understanding of human-computer interaction, a series of Project Activities were initiated in DIRC. A study called "Collaboration in Dependable Software" is one of these Project Activities and it aims to supply solid guidance and advice for the development of reliable software systems, reflecting the complexity of activities during the design and development of software, which

integrates people, software, tools, methods and techniques (DIRC's PA8 webpage: http://www.dirc.org.uk/research/activities/P8-description.htm, Dec. 2002).

Some activities in DIRC are concerned with technical aspects of computing science such as the design of fault-tolerance mechanisms. But the design and development of computer systems is clearly a human activity and it seems reasonable to ask what contribution sciences which look at human beings can make to increasing dependability. There is for example an obvious reason for looking at management aspects of programming teams.

Many studies (Boehm, 1981; Naur, 1992; Pressman, 1992; Shneiderman, 1980; Weinberg, 1971; and others) have reported surprisingly large differences in the productivity and accuracy of programmers working as individuals. Even for mental (rather than physical) tasks, these variations of between a factor of 10 and even up to a factor of 30 (approximately) suggest that there might be inherent differences in ability (Boehm, 1981). So one contribution that psychology might make to the creation of Dependable Systems is to attempt to investigate these differences. This is the purpose of the research reported in this dissertation.

From an interdisciplinary point of view, this research (as with parts of other studies in DIRC) aims to analyse whether code-review is influenced by personality factors.

To understand the link between both disciplines involved in this research – computing science and psychology, it is necessary to understand some concepts in

both disciplines. There will therefore be readers from both disciplines and consequently theories are explained using simple terminology as this will help understanding from both sides. Although the reader from the psychology field may find that the theoretical basis of personality is too basic or too superficial (the same applies to a reader from computing science while reading about the programming process), it has been carefully judged so that a reader from either field can comprehend the other field without problems.

To begin with the whole process of programming with special consideration to the debugging process and its importance will be introduced; this will be followed by an introduction on software reliability and dependability, then the concept of software verification and validation will be reviewed and last but not least the specific process of code review will be described in more detail.

Regarding the psychology field, concepts such as personality and its implications in the process of selection and recruitment, in the workplace and its possible relations with productivity will be described. Finally, examples of previous studies with computer programming and psychology will be presented.

# 2 COMPUTER PROGRAMMING

## 2.1 Software Design

Before discussing the whole process of software development it is necessary to understand the nature of software, i.e. the context in which software can be developed. Moreover it is also important to know the organizational context in which software companies can function.

Software is produced in two situations; one is when a non-software company contracts a software company (or consultancy firm) to develop specific software. The other situation, known as ‘in house’ software, is when software companies develop products to satisfy the needs of the market. There is also a growing trend for the development and acquisition of generic packages like SAP.

Software can be classified as “closed” or “open”. Closed programs are those in which it is possible to have a specification in a fixed domain; and ‘open’ programs, also called “computer-based systems”, are ones where the task of establishing the specification itself is complex and error-prone.

When a company is producing software, programmers are hired to work in groups, teams and/or projects with the objective of creating a reliable (see 2.2) and easy to use system. Weinberg highlights some of the differences between groups and teams. (Weinberg, 1971-1998).

Work developed by a group can be described as done by either a formal or an informal organization, depending how the workers have been distributed. It might be possible to detect a small amount of influence from other employees in parts of the software, but usually all work is done by individual.

When two or more employees have to work together to complete a job, a team is formed. In this sense, team work is related to a group collaborating in their professional work or in some activity or task. It is possible to conclude that programming teams are formed when programmers work with combined effort where each team will be responsible for a specific part of the task. However, this does not mean that the members of the team will be integrating code all the time, as they may develop their part of the job and integrate it later.

While *group work* means people working in the same place with some common perspective and without relationships between their work, *team work* is when they get together to develop a task. The *programming project* can be defined as:

> *"When two teams must work together to accomplish some programming end, a coordination function emerges just as it did when two programmers had to work together. New types of social organizations emerge, just as it did when teams were formed from individuals."*(Weinberg, 1971-1998)

Teams are the most common organisational framework used during the development of software. As it is a complex task (especially if a complex system is being developed) this working structure has to be done – as in any other creation

process, following a series of actions (steps) until its result is achieved. Although the programming process is an element in the development of any kind of software, it is during the development of "closed programs" that the steps of this process can be most easily noticed.

It is important to say here that although the employees are organised in groups, teams, or projects; most of the work is done alone, with the various components of the product being integrated later. In this current research, the main aim is to detect whether individual aspects of personality influence performance in the detection of bugs in programs. It is however important to describe briefly how the employees are organised during the software development stage, as they may sometimes have to cooperate with each other. Each company has its own policy which will determine how the employees will be organised. However, independently of its structure, the manager will always be responsible for the programmers.

### 2.2 Reliability and Dependability

When software is described as reliable, it means that the software will not break or fail frequently, i.e. the product will provide the utility which people are expecting from it. The most common way to attempt to produce reliable software is to test it during its development and make some statistical study of the failure rate of the frequency of failures exhibited by the software. (There are other approaches often grouped under the phrase "correct by construction" which aim to avoid errors. To some extent, the code review process looked at in this dissertation is an attempt to avoid – rather than locate by testing – errors.) In any case, a piece of software will be judged to be reliable if the mean time between failures is sufficiently infrequent,

or when the failure in some part of the system does not affect the overall output (one then says that the failure of a part of the system is masked or the fault is contained).

In the DIRC project, the main aim is to provide dependable computer-based systems. However, when dealing with computer-based systems, there are many aspects which may lead to malfunctions; consequently the systems will be less dependable. To create dependable systems it is necessary to investigate solid bases and techniques for the design and development of computer-based systems.

To deliver dependable software, it is necessary that the software is reliable, so that it can be trusted not to present significant failures. But for software to be reliable, it is necessary that it has gone through all the steps of its development, i.e. from its specifications until its documentation, and that the testing and debugging (as well as all other steps) are carried out by competent and qualified professionals.

This is then the link between dependability/reliability and bug-finding. If the software has been well designed and well coded but badly debugged, it will have to stay in this repeated process (code, test, debug) until the program behaves as expected. Dependable and reliable software will then, apart from increasing the cost, take longer to be produced.

### 2.3 The programming process

The programming process involves at least six main phases (this is in addition to the maintenance phase). This process can be also called the “software life-cycle”

(Ghezzi, Jazayeri, & Mandrioli, 2003), and there are various models. However in this research only the distinction between each part of the process is relevant, and not the way in which it is done. For this reason, only one simple model of the programming process will be presented here, i.e. the "waterfall model". The "waterfall model" presents clear distinctions between each phase. In a real situation however, these phases are not always clearly separated, as there is a constant movement between phases (in a cyclic movement). In addition, another concept was introduced in programming, that of "parallelism"[1].

The "waterfall model" is an idealised model of the "software life-cycle", in which a new phase only starts when the previous phase is finished[2]. It is however always possible that the cycle goes back to repair some mistake which has occurred in some previous phase (see Figure 1).

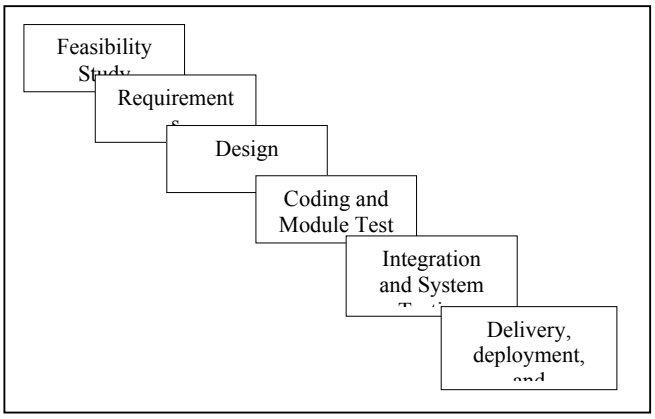


**Figure 1: A Common "Waterfall Model" (Ghezzi *et al.*, 2003)**

[1] Parallelism aims to reduce production time and provide an earlier delivery of the final product. Also called "concurrent engineering" and the steps of programming are developed in parallel, i.e. while one group is concerned with specification other groups, at the same time, are performing the other parts of programming (Ghezzi *et al.*, 2003)

[2] It is idealised because it is quite impossible to define the steps in a production as some backtracking through the previous steps is always necessary.

A study called a *feasibility study* has to be done. The feasibility study is performed before the beginning of the software process, and in this initial step of the development process an analysis of the costs and benefits of the creation of a new product is performed. The possibility of buying a similar product which would cost less than building a new one is also considered in this study. If after all it is decided that new software will be produced, the next step may start.

With the needs of the new software in hand, the *specification* (requirements) begins to achieve these requirements. In other words, the specification phase is when the software's functions are defined. A useful specification should contain a good description of the terms, the input-output relation expected, and also a clear identification of any assumptions (a lack of recorded assumptions is the major problem in software development).

The idea that the *documentation* (program specifications used as a user manual as well as technical information for reference) should be done in this earlier phase is supported by some theorists (Bray, 2002, p213) on the grounds that it would be easier to access it during the development of other phases, and especially during testing; however documentation is usually the last part of programming to be done.

It is important to emphasise here that after a phase is completed, and before the next phase starts, there is an essential intervention which has to be realised. This is called "*verification and validation*" (V&V) and it consists of ensuring that the requirements established during the period of specification have been achieved, and

that the product being constructed is meeting the needs of the software in development.

The next phase of the cycle is the *design* phase in which software engineers will create a software system to meet with the specifications produced in the previous level, i.e. stipulate how the product is to work. As a way to complete the product, the tools have also to be defined in this phase. The main difference between requirements/specification and design is that the former shows and identifies the problems, while the latter solves them.

After specification and design are established it is time to start the next phase which is *coding* the program. It is in this part of the process that a particular system will be coded to meet the objectives established in the design (and thus specification) phases. To transfer these instructions to a system it is necessary to use a computer language which might be JAVA, C++, Basic, or a range of others.

Program design and coding are complex tasks because, in a sense of strength of computers: computers execute their code exactly as written which means that designers and programmers must foresee all possible situations.

If a program contains an error, the error will either prevent the program from running, or (permit it to run, but) the program will produce an output different from that expected. It is obviously desirable that all such errors are removed before a program is handed over to its users. As indicated above, one approach to detecting

errors is to *test* the program, and by removal of the identified errors to produce a program containing as few errors as possible. There are two types of errors: *syntactic* and *semantic*, details of which are discussed later in this chapter.

Errors in computing are often referred to as “bugs”. There is a possibly apocryphal story this name derives from a moth (bug) which get caused an early electro-mechanical component of a computer to malfunction. Whatever its origin, the term “bug” has stuck and is commonly used for errors in software: this dissertation follows this terminology.

The danger of the presence of bugs in a program gives rise to a new phase in the software development process, as the program has to be *debugged.* The term *“debugging”* is used broadly to cover both finding and removing the bugs. However it is necessary to understand that not all bugs can be removed at once, and that a program might have to be tested and re-tested until the program presents the correct output. Furthermore, it frequently happens that new errors are inserted when the original error is being corrected.

It must also be understood that, because of the astronomical number of paths in a large program, it can be never be fully tested. Edsger Dijkstra’s famous aphorism on the subject is that “program testing can only show the presence of bugs, never their absence”.

So, even when the product is ready to be commercialised, this does not mean that the program will never come back to the company, as it may occasionally require some maintenance. There are two types of maintenance: “corrective maintenance” and “enhancement maintenance”. The first one is performed when the product presents some fault after deployment and needs to be corrected, while the second is when a product has been used for some period of time, and requires updating, meaning that functions may be altered, added or excluded.

It is possible to see that the development of software is a complex task, and that debugging is a significant step in software production. It is in this phase that errors will be located and removed, allowing the software to run as it was intended to.

The debugging process will be further described as well the importance of studying it. Also some previous work about debugging and psychology (which includes some personality factors) will be discussed. Writers on this topic have agreed that finding the bugs in the programs is more difficult than correcting them (Winder & Roberts, 1999). In this sense, attention will be focused on the Verification and Validation (V&V) process and in particular to code-review, which is a subtask of V&V.

### 2.4 Software validation – Verification and Validation

V&V has to occur between each part of the program process to make sure that the software is being produced under the requirements specified. Sommerville (1992, pg. 374) points out two objectives of V&V: “the discovery of the defects of the system under development, and assess whether or not the system is usable in an

operational situation". The difference between validation and verification is that while the former checks whether the correct product is being built, the latter analyses whether it is being developed correctly.

This dissertation will focus on verification rather than validation, with particular reference to the code-reviewing which is performed during and after coding.

There are two reasons for testing: defect testing and statistical (reliability) testing. The second one is aimed at establishing the software reliability by running a (hopefully representative) collection of tests and seeing what percentage fail. Reliability testing tries to provide quality control. The first sort of testing attempts to locate errors during development so that they can be removed. However if after the defect testing has run, and no errors are detected in it, it does not mean that the software is free of errors, as there is the possibility that the testing has failed to detect all errors and defects.

### 2.4.1 Testing

Just as software is built according to a progress of steps, so large systems are also built according to a certain structure (*modules*, *sub-systems*, and *systems*). A module is a fraction of a large program which has had to be divided as a way to facilitate its implementation and development. When there is a large enough number of complete modules, they are grouped into a bigger structure, called a sub-system. This will later be grouped into the final product, the system. An analogy can be made to the building of a large mechanical system, such as an aeroplane.

Various stages are also established during the testing process. Sommerville (1994, pg. 376) suggests that there are five stages, divided into 3 main groups – *component testing, integration testing* and *user testing*.

*Component testing* involves two stages of testing itself, these being *unit testing* and *module testing. Unit testing* ensures that the components are working correctly (they have to be tested individually). *Module testing* is the second part of the process and it is related to the testing a collection of dependent components.

*Integration testing* is when the modules are integrated into *sub-systems* and the sub-systems are integrated into *systems*. In the first one, it is important to detect whether there are interface errors, as the sub-systems could have been developed independently. S*ystem testing is* responsible for finding any errors that were not detected during the previous phase, and also to ensure that the specifications have been achieved.

The final part is related to *user testing*, and is also called *acceptance testing*. Its objective is to certify whether or not the performance and functionality of the product have been achieved. It is also called *alpha testing* and it goes on until all the requirements established become acceptable (for both customer and the program developer).

If the system goes to the market, i.e. it was not ordered but built according to a necessity, then another stage of testing is carried out. This is called *beta testing* and

consists of delivery of the product to a number of potential customers, trial, and finally the return of the product with customers' comments and reports on errors. Then the product is ready for general sale.

When a product is being repaired, it is quite common that new errors/defects are introduced. In this case, it is necessary to carry out *regression testing*, which consists of repeating the testing after the corrections have been done. Typically testing is responsible for about half of the costs in program development, and planning is necessary in order to optimise the use of time, labour, and resources. As a program has to be tested over and over, the program is usually divided, and a given subset is tested after *test planning*.

Test planning should start at the beginning of the programming process, i.e. during the establishment of the system requirements, and its details should be developed while the program is being designed. Planning the test in this way will reduce the expenditure of time and money.

During the planning process, there are various useful ideas which can be built into the testing. Such ideas include: a description of the testing process; a list of the requirements; a list of all the items to be tested; a schedule of the testing; a record of the procedures and results; an inventory of tools required for the testing and the programs' utilisation; and any constraints which may affect the testing.

Testing is usually carried out by a person who is not involved directly in the programming task, as human nature dictates that people should feel a certain attraction to their products and hence be reluctant, even unconsciously, to discard their work and repeat it. In this case the programmers are not those who run the testing, but are merely told about the errors found and then asked to make the changes.

### 2.4.2 Reviews

There are three main types of review: one is related to *management review*, and consists in keeping the manager informed about the software project. The questions about costs, plans and schedules are all discussed in this review, which also includes the conclusions about the project development and how important the product is. As this kind of review is not important for the purpose of this research, it will not be explained in detail.

The second kind of review is called the *quality review*, and consists of a review of the work of a team or individual by a jury of project members and technical management. Its objectives are to check whether or not the requirements have been met, and to detect errors and inconsistencies, which will then be pointed out to the author. It involves both analysis and feedback relating to the material, and although important for software development, is not relevant to this piece of research.

However it is essential to add that during the review process there should be an exchange of information, i.e. the designer explaining his work and the project

engineers giving the feedback, which transforms the review into a type of lecture inside the work environment. This activity is very useful for the “junior” employees, so that they can learn about system design.

The third type of review is called *software verification*, or *program inspection*, and its aim is to discover defects. Sommerville (1994, 459) listed some important requirements to be followed while an inspection is being conducted:

- precise specification of the program
- familiarity with organisational standards by all members of the inspection
- use of an up-to-date- version of the code
- provision of a check-list of the most common errors
- managers should be prepare to spend more money if necessary;
- inspections should be considered as part of the verification process, and not as evaluation of the personnel

The inspection is a formal process in which the code has to be carefully analysed and its defects pointed out. An inspection is performed by a small team, usually of four members: the author, i.e. the person responsible for the program; the reader, who presents the code; the tester, who reviews/tests the code; and the chairman, who controls the group and motivates participation (the latter should be an outside guest).

Apart from being familiar with the language in question, the team should not spend more than two hours testing the program, as after this period the defect detection

may drop and some mistakes may occur. The inspection should also take place not only in the final stage of the product, but be a constant aspect of the programming process.

### 2.5 Debugging

The act of debugging occurs when a program or piece of software presents some errors. Shneiderman (Shneiderman, 1980) says that these errors can be found in two different forms: the simple one is when error messages appear as a consequence of syntax errors ('syntactic bugs') in the program. This problem can be solved by simply putting the program in the compiler, which will find the bugs. These errors are considered as 'mildly annoying' bugs (Pressman, 1992).

The other type of error, namely when the program is presenting outputs different from those required, i.e. running incorrectly, requires more time. This is because it is caused by 'semantic bugs' which are more difficult to find, which may prevent the program from running as intended, and can cause 'serious economic or physical damage' to the company (Pressman, 1992).

Semantics bugs cannot be located by a compiler, but need to be debugged by a programmer. In this sense, it can be seen as a waste of time because while the program is being debugged, the production has to wait until it is finished.

Debugging is a difficult process and can take up to three times longer than coding (Gould, 1975). Sommerville (Sommerville, 1992) says that sometimes debugging

and testing activities are considered parts of the same process, when in fact they are relatively different.

Testing determines that the program contains some defects, while debugging is concerned with locating and correcting then. To locate and correct the bugs, it is necessary to follow a programme of steps (see Figure 2), and then a hypothesis of how and why the program is presenting such unexpected behaviour has to be formulated.

After an error is found, the method for repairing it has to be designed and then the correction has to be done. After the repair is completed, it is necessary to run the testing again to ensure that the changes have been carried out correctly.

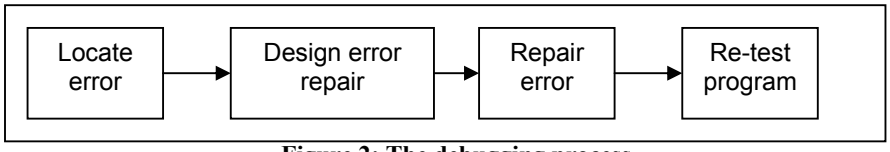


**Figure 2: The debugging process**

Gould (1975) describes some reasons why debugging is not an easy task. To begin with, programmers do not have knowledge about how many bugs are in the program. Also, they do not know the nature of the bugs and their location in the software. Then programmers have a restricted ability to simultaneously track several aspects of the program's detailed procedure specification. Finally, debugging requires a high degree of precision because computer programs need to be accurate. All of these reasons make it necessary that programmers debug the program over and over; this will inevitably retard the production of software.

In addition to being the longest part of programming, debugging is also the most expensive, according to Gould (Gould & Drongowski, 1974). It would cost from 25% to 50% of complex software.

Pressman (1992) argues that there are people who are good at debugging while others are not. He says that there are large variations in debugging when employees with the same instruction and experience perform the same task. For him this fact may be the consequence of some natural (innate) 'human trait'. This finding is closely linked to the main aim of this current research.

### 2.5.1 Importance of Debugging

Debugging is a vital part of programming as it is responsible for cleaning unwanted bugs from the software, making the program run as it was expected to. This procedure might provide reliable and cheap software. However if it still taking a long time and costing lots of money, the final product will have a higher price than ever.

Because of the high cost, and the time spent on debugging, exploring debugging and personality as Pressman (1992) suggested is very important in order to discover which personality factors may be influencing performance. When debugging is done by competent professionals, it is possible that it would take less time and consequently cost less than if it was done by inadequate (non skilled) professionals.

Companies have to hire programmers to work on the production of software, and if these programmers do not work properly, debugging will take a long time, then the companies will be spending more and more money on it and eventually the price of the software will be high. The program has to be almost completely clean of bugs to be on the market; on the contrary, if the program still presents many defects; it will not be consistent and will present poor quality.

When debugging is done by competent professionals, it will cost less money and take less time to be completed, and as a result the company will not spend that vast amount of money.

### 2.5.2 Debugging and Psychology – other studies

It is clear that there might be something altering the programmers' performance at debugging, as illustrated by Pressman (1992) earlier, in which professionals performed differently in the same task. And as soon the reason why the difference in the performance at debugging programs is established it would be easier to relocate programmers in companies for an optimization of the work which will 'improve the productivity of computing systems' (Gould & Drongowski, 1974).

As debugging needs people to work on it, and reliable software can only be offered after it is virtually free of bugs; then, if there were at least some clues to how to select the best professionals for each phase it would make the product not only more effective, but the time spent on it shorter, and consequently the result should be cheaper, more dependable software. Because skilled professionals will be producing reliable software and each programmer will be developing that specific

part of programming according to their capability. It is also likely that the people concerned will be happier if working of those phases where they are most productive.

In this sense, the idea that debugging may be influenced by personality has been chosen as a way of understanding why there is such huge variation between programmers as Pressman (1992) noticed. Is it because of some personality factor?

## 2.6 Bug Finders or Code-reviewers?

During this research, it was necessary to develop a terminology and a necessity to refer properly to the participants of this research emerged. At first, it was thought of calling them debugger but such term is a little bit complex as there may be some misunderstanding as this term is also used to refer to a tool on the computer to help in the debugging process; and after searching in dictionaries it was found that debugging is the act of *finding* and *removing* (fixing) the errors (bugs) in a program or piece of software making it run as expected. If the term debugger was used it could be referring to a person who besides finding the bugs in a program would also have to remove them.

Seeing that in this research the subjects will only be asked to find the bugs on a JAVA program and not to fix them it is crucial to adopt the correct terminology. Looking at the *Oxford English Dictionary*, it was found that the word “finder” can be understood as the one who finds or the one who comes upon a discovery by change or search which made this term an appropriate terminology for the

participants. However to be more precise, ‘code-reviewers’ will be the term used to refer to people working on the code review process, i.e. reviewing the code and then finding bugs in it.

It is interesting to add here that although it is said that the reviewer will become a specialist in this function (Naur, 1992), this current study aims to discover if there is a personality type influencing code-review performance.

### 2.7 Programming at Newcastle University – Course Structure

The participants chosen to take part in this current study were students, as they are less expensive than professionals; and of Newcastle University simply because it was an easier way to have access to them.

It was agreed that the students who were taking part in this research had to all be in the same level, so that there would be a uniformity of the acquired knowledge, rather than huge gaps, and that the language in which the program was written had to be common to all of them. As the students learn JAVA in their first year, it was thought that taking students from the second year would be a good idea, as the languages specifications should still be fresh in their minds.

JAVA is a relatively new computer programming language, and it is very popular and widely used nowadays. JAVA can be used in many kinds of projects but is a useful tool while Object-Oriented systems are being built.

At Newcastle University (NCL) there are three programmes of study which are CS – Computing Science, SE – Software Engineering, and IS – Information Systems. There are differences and similarities between them.

Broadly speaking, the students from IS are taught how to develop database systems, and also some other software issues (like administrating computer systems). The SE courses are focused on software tasks, while the CS students learn about hardware and software.

The three programmes of study have many modules in common. The Group Project (CSC226) [which was the bridge between the researcher and the students] for example is a module in which the students are grouped randomly and are asked to develop a project from its requirements until its final phase. Thus, the students can experience the team work which is a common practice in software companies. Figure 3 shows the aims of the group project at both sites (http://coursework.cs.ncl.ac.uk/modules/2002/csc226/.

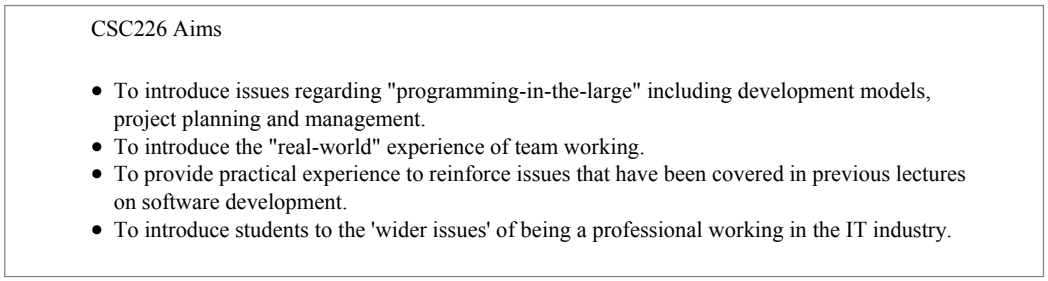


**Figure 3:** Aims of Group Project at Newcastle University.

Because of the similarities between programmes of study, and because the students are already able to develop a program as they are expected to in the group project, it was assumed that the students would be able to find the bugs easily, as JAVA language was a common language for all of them.

However, it is known that the students focus their attention on different interests during the course, and because of that it is likely that students from each course present some differences in the performance of 'code review exercise', i.e. students from Information Systems are expected to have lower scores in the code review exercise than students from Computing Science and Software Engineering, as the programme of their course does not go so deeply in the programming process.

A questionnaire was formulated to access the students' self-rated abilities and skills. Details about the questionnaire and about the students' profiles can be found later on, in Chapters 5 and 6.

# 3 PERSONALITY

Bernstein, Roy, Srull and Wickens (1991) state that there are four main approaches to describing and understanding personality: the psychodynamic, the dispositional, the behavioural and the phenomenological. Although it may not seem relevant for the purpose of this current research, each approach will be briefly described here as a way to introduce the bases of personality concepts. Special attention will be paid to Jung's personality development theory, the theory that people have a tendency to act according to specific psychological functions; as this is the basis of the personality assessment adopted in this present research (Bernstein, Roy, Srull, & Wickens, 1991).

The *psychodynamic approach* started with Freud, and states that personality is structured during conflicts between the basic needs, i.e. water, food, air, sex, and aggression; and the demands of the real world, i.e. how to satisfy such needs. Depending on how well a person succeeds in satisfying his/her needs, he or she will exhibit a healthy or unhealthy personality.

Freud had some innovative ideas which created considerable disagreement when first presented; this led to some variations in Freud's theory, and the work of the neo-Freudians. These neo-Freudians were Jung with *Analytic Psychology* (which will be described in detail later); Adler with *Individual Psychology*; and Erickson who used *Psychosocial Stages*, in places of psychosexual ones.

The next approach is the *Dispositional Approach* which describes a person as a collection of stable internal characteristics which lead to certain behaviour. This theory assumes that people are different from each other, having different wishes and feelings which are also expressed differently. Such differences are called personality differences, and a good definition is quoted by Brunas-Wagstaff, (Brunas-Wagstaff, 1998) and it was written by Allport:

> *Personality is a dynamic organization, inside the person, of psychophysical systems that create the person's characteristic patterns of behaviour, thoughts and feelings.*

Personality is then, the sum of a person's qualities, and his/her behaviours are mostly determined by his/her personality characteristics, in which case it is possible to say that a person is likely to behave in certain ways because in other situations he/she behaved in a similar way. However if the same person is facing an unexpected situation (illness, stress, lack of time, etc) it is likely that he/she will behave differently from the expected. Such a collection of qualities, which is established during a person's development, is acquired through a person's history, family background and experiences. Eysenck and Allport are some of the theorists involved in this approach (Bernstein *et al.*, 1991).

Skinner, Rotter, Bandura and Mischel are some theorists who are involved with the *Behavioural Approach*. They argue that the "personality is a label that summarizes a person's unique patterns of learned behaviours" (Bernstein *et al.*, 1991).

The *Phenomenological Approach* is also called the humanistic approach and it is assumed that personality is determined by the specific way in which a person perceives and interacts with the world. To understand a person, it is necessary to "wear the person's glasses and then perceive the world through that perspective" (Bernstein *et al.*, 1991). It is influenced by *Gestalt* psychology, and has been theorised by Carl Rogers and Maslow.

### 3.1 Jung's Theory of Personality

The personality assessment adopted in this research was developed from Jung's concepts of ***Extraversion and Introversion*** as well as ***Superior and Inferior Functions,*** which are explained in the *Principle of Entropy* and *Self-Actualisation.* However personality is described by Jung as being much more than this as it can be seen in Bischof (Bischof, 1970).

According to Bischof, Jung described personality using: *Principle of Entropy, Principle of Equivalence, Polarity, Self-Actualisation, Unconscious States,* and *Teleology*; such principles are strongly interlinked, thus all of them will be briefly described in this section, as well as the concepts of Extraversion-Introversion and Superior and Inferior Functions.

For Jung, *Polarity* is, in a way, central to the maintenance of life, as polarity suggests opposition and everything in the world is guided through opposites, which lead to conflicts, that, once solved, will bring progress to the person. There is also a period of equilibrium once the conflict is solved and the progress is achieved, but it is generally short, because there is always some conflict to deal with. This conflict

is usually resolved by one of these three actions: *compensation* (the initial goal is removed and the person redirects his/her desires to another goal as attractive as the initial one), *union* (opposite forces are united to find a satisfactory solution for both), or *opposition* (possible progress is achieved through the movement created by opposites).

Then there is the *Principle of Equivalence,* which is derived from the thermodynamic area of physics. This principle means that a person does not fail to fulfil his/her desires, as the desires will be always converted/diverted to another objective. The desire can be repressed, and the consequence for its repression is that the energy to realise the desire is transferred to a dream plan (diurnal or nocturnal dreams), thus allowing the person to resolve his/her conflicts.

The *Principle of Entropy* states that when two bodies of the same nature are put together, the more highly-charged body will lose energy and the two bodies will achieve a state of equilibrium; in Jung's theory this principle is adapted to explain that, although it is impossible to achieve a true state of equilibrium because a human being is not a closed system, a person can achieve moments of peace and tranquillity once he/she balances the energy spent in his/her conflicts. There are "mechanisms" that help in this balance and extroversion/introversion, and superior and inferior functions are some of them:

- ***Extraversion and introversion***: this is one of Jungian concepts to be adopted by the modern psychological world. Extraversion means that people move in the direction of other people. Extraverts centre their lives on

actions, while introversion means the opposite – people tend to move in the other direction, i.e. towards a quiet world, free from people, and are focused on subjective experiences. Jung suggested that people are in constant conflict with their "style", as in an external world they behave in a mood and in the internal (dream) world they act as if they were the other style.

- ***Superior functions and inferior functions***: these functions are better explained in the principle of self-actualisation. There are 4 functions: intuition, sensation, feeling and thinking. Generally a person uses more [and more] comfortably a determined function which is called superior function. Then a second function used is an auxiliary function and the remaining functions are called inferior functions.

The principle of *Self-Actualisation* addresses how a person evolves and achieves improvement. This principle uses existence and non-existence as a polarity, and progress is achieved once the existence is guaranteed. There are various components in this system which help in the maintenance of a person's existence:

- *Functions:* the objective of these functions is that a person develops all of them so that the psyche can achieve a balance. As absolute equilibrium is an impossible goal, there is a superior function (the one a person feels more comfortable with), and an inferior function (the one which is not very often used by the person); each of these two has an auxiliary function. For Jung there were only four functions: intuition, sensation, feeling, and thinking:

- *Intuition:* this is when a person makes a decision on the basis of the unconscious, as the person is unable to explain how he/she took the decision. The answers will be processed by an unknown method which is impossible to be discovered. This function can be considered as not being part of the mental functions as it cannot be sensed, felt, or analysed.

- *Sensation:* all decisions and thoughts can be taken by analysing the things as they are, i.e. by seeing, hearing, smelling, tasting and feeling. This function is non-rational as the person does not need to think achieve a conclusion, the answer is just in front of him/her.

- *Feeling:* although feeling has many meanings, in Jung, it means the values a person gives to other people, places or events. It is concerned with how a person feels about something, and then it can be concluded how important this something is to him/her. To Jung, feeling is a rational state, and to value things it is necessary that the mind makes judgements.

- *Thinking:* this is also a rational function, and is based on making logical analysis of the facts and then drawing a conclusion.

- *Extraversion-introversion:* are aspects of the self. The extravert shows his/her emotions and his/her actions are usually impulsive; the introvert hides his/her emotions and problems are solved somewhat passively. Self-actualisation is achieved when a person has a certain balance between these two characteristics.

Jung emphasised the *Unconscious States* more than the *Conscious State*. If a person ignores the unconscious stimulus sent, it is likely that the result will be disillusionment, compulsions or phobic states. The unconscious is divided into two parts which can work singly or in conjunction with one another. These are called the personal and collective unconscious.

Finally, there is the *Teleology Principle* which states basically that humans are getting better and better, and will eventually achieve the true state of self-actualisation. To Jung, humans are going in the right direction to their self-actualisation as they are supplied with a structured brain and polarities. Jung also considers that human began as individuals, and that the process of individuation starts from the self and goes in the direction of others.

### 3.2 Personality at work

Assuming that each person behaves according to his/her characteristics, it can be said that a person's personality is also reflected in the workplace and that such characteristics may influence people, whether directly or indirectly, when they are choosing a career.

If a person works in a job which matches his/her personality, then it can be said that a person is satisfied, and perhaps that the job will be well executed. This is because personality strongly influences people's careers.

It is also important to assess a person's personality, so that employees with a certain affinity can be put to work together while employees with conflicting types can be supervised and oriented in a way to avoid internal conflicts and consequently affect productivity. In this sense, it is important to understand how personality and productivity interact.

### 3.2.1 Personality and productivity

It is quite hard to establish whether or not a worker is being productive, but there are some types of measurement that can help in this establishment which are quantity, quality, and accidents/rejects (Furnham, 1994). Looking at quality and quantity is a positive way to measure the productivity, however looking at accidents and rejects is a negative way to measure the productivity.

Furnham, Jackson & Miller (1999) say that the measurement of personality in the work place is increasing nowadays, and this fact could be attributed to the necessity to have predictors which could help in selecting and retaining good workers.

In addition, Furnham et al. (1999), declare that there is some evidence to support the notion that personality measures can help in the prediction of work performance. The authors carried out a study to distinguish whether work performance was a result of either personality traits or *learning styles*[3]. They analysed the work performance of 203 tele-sales employees, and discovered that, although there were

[3] *Learning Styles* is a term that describes the manner in which a person acquires and processes information. Each person has a particular way of collecting and storing all the information he or she has received. The learning style affects how a person learns, acts, behaves, thinks, and works.

some personality traits linked to the learning styles (which would link personality trait indirectly to work performance), the highest predictor of work performance was learning style.

Although it is possible that the performance of the individuals analysed in this current research is to a certain extent determined by either learning styles or personality types, it was decided that the former factor (learning styles) was not going to be included in this present research because, as this is an MPhil project, the time was too short to extend the study.

When the job requirements match with the personality characteristics a person has, it may be possible that he/she will be more motivated to perform the job well. In this case, personal motivation may also influence job performance.

Ghezzi et al. (2003) stated that in the field of software engineering there are many intellectual activities, and in this sense people are the most important aspect in the production of high-quality products. Thus, personality characteristics of the employees must be considered as apart from the capability to perform well. There are good, average and inadequate professionals. It has to be also remembered that they are always interacting with each other. Such are the characteristics on which their performance depends.

Thus, it is very important that the employees' characteristics are known and that they are put to work in a team which matches with their necessities. This will make

the employee feel comfortable in the work place and his/her job will be performed without conflicts which may interfere in the production.

### *3.3 Introduction to recruitment and selection*

The process of selection for a job is done following steps which gradually eliminate candidates until there are few enough for the specific jobs available.

First of all, when an advertisement about a certain vacancy is prepared it contains the job specifications that the candidate must have knowledge of. This knowledge must be shown in the candidates' CV, and an analysis of it will eliminate those candidates who do not present this knowledge. Sometimes it is specified that the candidate must have studied a certain subject, and this has to be proved with a valid certificate.

This process has disadvantages as well as advantages. Those candidates who have a self-taught skill will not be able to prove it, and will consequently be excluded even if they have enough talent for the job, unless they have already had some previous working experience to prove it.

On the other hand however, the candidates with those specifications but who demonstrably lack competence to perform the job will be excluded too. The candidates who passed on the CV analysis will be then invited to an interview in which the interviewer must have in mind whether the applicant can and will do the work (Smither, 1998).

Usually after the interview, the candidates are selected. In some companies however, there are still some phases to complete. These other phases are related to candidates' abilities, personalities or even interests, and these will be assessed through psychological testing. This testing can be used to measure ability, aptitude and interests, or to assess personality or honesty (Smither, 1998).

In the specific case of programming, the better solution is the adoption of ability and aptitude tests to select people who know how to perform the job. Another solution is the use of psychological testing, in which personality factors can be assessed. The personality assessment is important not only to see how the job would be performed, but also how candidates would behave when facing difficult situations at work (or even problems at home) also it would give an idea of how candidates behave while working within groups or teams. Weinberg (Weinberg, 1971-1998) states that making mistakes when hiring people is easy when not considering personality factors, as it is difficult to predict how people will act during stressful situations.

However the facility of predicting how people would behave was not accessible as it is nowadays. Smither (1998) says that, during the 1960s and 1970s, assessing personality was not popular among employers. Either because the "validity [of tests] could diminish the chance of unfair discrimination", or because the "scores were not particularly useful in making predictions about performance", it was also said that it would be an "invasion of privacy", and that the result of some personality tests can be forged by smarter candidates. Nowadays, more personality measures

have been developed, making the assessment of personality easier and, to a certain extent, more reliable.

In a study undertaken to assess whether personality and cognitive ability are predictors of success in skill acquisition and job performance among air traffic controllers (Oakes, Ferris, Martocchio, Buckley, & Broach, 2001) it was found that there was a positive correlation between four (out of seven) personality traits and skill acquisition. In this sense, the use of personality testing alongside other methods of selection is an important predictor of success/failure on job performance.

Cusumano & Selby (1995) observed many aspects of a well-known software company, including the process of selecting and recruiting new employees. To begin with this company has a policy to hire young people, as "they can more easily socialise into the company way" (Cusumano & Selby, 1995).

New employees are selected according to various pre-determined characteristics provided by a team of functional experts. Recruiters call students in famous schools, colleges and universities and then manage the hiring process – the approval of the candidate is done by experienced employees, who also run the interviews. Good candidates come back for additional interviews. This process does not consider any personality type, but does consider candidate' skills and talents, and ability to work by themselves.

Only a small percentage of candidates are selected to work in this company and many candidates are rejected. It is believed that this company contracts only half of the employees needed to develop software. This policy results in overloading the staff and consequently there is a large loss of employees.

Selecting people in the way that this company does it is a bit of a risk. It is possible that during the selection some good professionals (who would not burnout as easily as the others) are excluded because they do not fit the company's policy.

### *3.4 The assessment of personality – MBTI*

According to Bernstein et al. (Bernstein *et al.*, 1991), there are three distinct ways to assess personality: observation, interview and personality tests. However as the current research allowed only a short time to collect data, it was necessary to adopt a method in which would be possible to have all participants doing the tasks at the same time. Interviewing lots of students together was thus impossible; in addition the method chosen would not require long time to be analysed. These two constraints made it impossible to adopt observation and interview. The next step was to choose the personality test which would help identify the personality types involved in the performance of code-reviewers.

There are two types of personality test, the objective and the subjective. An objective test is a paper and pencil test and consists of questions, statements or concepts (Bernstein, et al. 1991) in which the participants respond by selecting the answer which is closest to their feelings. These answers can be forced choice (yes-

no or true-false), multiple-response or even rating scales. Objective tests are easy to administer and are efficient.

Although there are many kinds of personality test, in this research one specific assessment will be used. One reason is that there are some tests that while assessing the personality also shows any mental disorders which participants may have, such as depression, schizophrenia etc, the MMPI is an example. Another reason is that the personality assessment chosen has already been used by some researchers in the field of computing science, which is described in this chapter.

In this study, the personality assessment chosen is the Myers Briggs Type Indicator (MBTI) which is one of the most widely used personality assessments (Bishop-Clark & Wheeler, 1994; Devito, 1985; Edwards, Lanning, & Hooker, 2002; Smither, 1998).

This method is based on Jung's theory of psychological types relating to bipolar factors or dimensions such as extroversion/introversion (EI), sensing/intuition (SN) and thinking/feeling (TF). When the MBTI was developed Myers and Briggs added another pair of characteristics which relates to judgment and perception (JP).

Relating to personality and career choice in general it can be said that the dimensions extraversion/introversion and judgement/perception determine people's attitude while sensing/intuition and thinking/feeling are responsible for the attraction to certain jobs.

Myers (Myers, 1990) explains the EI dimension as the manner in which people tend to "recharge their energy". Extroverts will focus their attention on other people through the external environment. On the other hand, introverts will be fully recharged after staying with close friends or family in an internal environment. With regard to career choice, this dimension influences people to choose the sort of occupation related to their style: extroverts need contact with people while introverts get stressed when they have too much contact with people, as they would prefer to work with impressions and ideas. However interest in jobs is mainly determined by the dimensions SN and TF (Myers, 1995).

The second dimension – SN is related to how people acquire information, i.e. if it occurs through the five senses in a concrete way, or through intuition using imagination and inspiration. The third dimension of inclination is TF and it is concerned with how people make decisions. The decisions can be made in two ways – one is done following some logical sequence of facts, and the other is when people decide based on some people-centred opinion (without being logical).

These two dimensions of preference are responsible for cognitive dimensions which determine how people feel attracted to and satisfied by their career choice. Both of these factors are also important tools in solving problems, as they help in the improvement of problem-solving skills. It is important to emphasise here that one type is not better than another; people just need to know how to take advantage of their type and then learn to recognize their preferences and abilities.

People's lifestyles are determined by the dimension JP (Myers, 1990). If people are controlled, orderly, and plan everything carefully they have a judgement orientation. On the other hand, if they are more flexible, spontaneous and adaptable to new situations they have a perception lifestyle. In a person's career, judgement and perception influence how the job is performed (Myers, 1990).

Because the MBTI is explained by a "format" of factors, as can be seen below (Figure 4), a combination of dimensions could be created giving origin to "sets". The sets are responsible for certain individual characteristics and there are two main sets which are established by rows and columns, and four other sets, which are considered as a combination of the main sets.

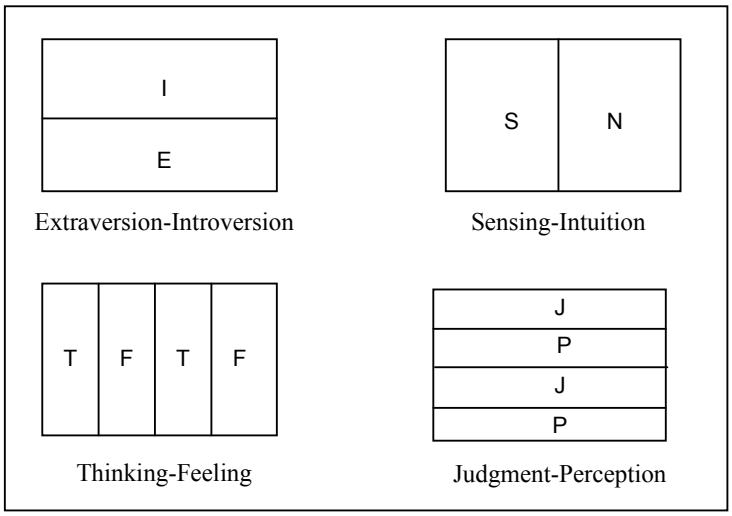


**Figure 4: Format of type tables**

The first main set is related to extraversion/introversion combined with the judging and perceiving (rows). In this sense, there are the introvert and extrovert irrational types (IJ's and EP's), and the introvert and extrovert rational types (IP's and EJ's). However the most important set of types is the combination of perception (S and N)

and judgment (T and F) (columns), as they are the responsible for the attraction and satisfaction people feel in their jobs and careers.

The MBTI is intended to be used with "normal individuals in counselling and within organizations" (Devito, 1985). Smither (1998) says that MBTI is not a useful tool in selecting people for new jobs. In fact, the MBTI is a useful tool in relocating people within organizations, i.e. relocating current employees for a special task and to improve work communication/interaction. The MBTI is also a popular personality measure between employers and employees.

Weinberg (1971/1998) states that after the first edition of his book, he would have written a completely different chapter about personality if he had known the advantages of the MBTI. He adds that this assessment is dealing "with normal personality differences" (Weinberg, 1998, p.8i) as some personality tests are related to mental disorders (this does not apply to the 16PF), i.e. they do not see the person as being normal.

Edwards et al. (2002) state that despite the popularity of MBTI it has been criticized on a number of grounds. One aspect is that some theorists consider typologies as an approach to label people. Also "there is evidence that the MBTI dimensions, rather than clustering into distinct categories or interacting with each other, are more akin to four additive main effects" (Edwards et al., 2002). In this sense, despite from labelling people, the MBTI tends to be a reductionism approach as the personality types are only considered according to the 4 bipolar aspects given

by the MBTI. But because this personality assessment is widely used even within the computing science field, it was decided that such criticise would not interfere in this current research.

### *3.5 Human Factors in Programming*

The process of programming, just like any other activity, may be influenced by human factors, a topic which has already been studied by researchers such as Weinberg (1971) and Shneiderman (1980) and others. It is therefore important to understand the importance of human factors in Software Engineering.

To begin with, members of staff in all sorts of employment need to be regarded as individuals, and the interaction between individuals within the organization must also be observed. Then, as computers are used by people, it is important to understand how this interface is built, as it will facilitate the development of a system which considers human limitations and abilities. Finally, productivity can be increased, which will help in the reduction of costs (Sommerville, 1992).

The same author states that the influence of personality factors on programming is already being considered in the selection of new programmers, although he concludes that it is quite unlikely that selecting people only on the basis of personality will succeeded. This statement is justified according to 3 reasons:

- It is possible that a programmer's personality may change over the course of his/her career,
- There is a risk of smart programmers cheating during the personality assessment

- Most important – there may be different personalities covering the different aspects of programming process (is it possible that some personalities will perform better than others?).

While it is true that a programmer has to have ability to deal with stressful situations and to adapt quickly to different situations, there is nevertheless no evidence that the performance of programmers can be characterised by any personality factor (Weinberg, 1971-1998). Naur (1992) stated that to work with problem-solving, success in the task can best be achieved when the programmer is open to unexpected possibilities, letting his/her intuition aid in solving the problems.

In addition, Naur (1992) argued that much of what the programmer does is a reflection of his/her "*personal style*", i.e. that it is not possible to determine a specific programming methodology because it depends on the variety of tools and techniques adopted. This is because each piece of software will require a particular tool, which will be used by each programmer according to his/her "*personal style*".

Most of the studies in human factors are interested in understanding the knowledge process and group working, which includes personality in groups, egoless programming and group interaction (Sommerville, 1992).

As there is little information about personality and individual performance in the programming process, an innovation in the field of human factors is proposed in this current dissertation which includes an investigation of the job performance and

a specific aspect of human factors – the personality type. By analysing this data it will be possible to confirm whether or not success or failure in the work field is influenced by personality.

### 3.5.1 Introduction to psychology and computing

This study has already claimed that programming may be influenced by personality or at least by some psychological aspects of how people behave or should behave. And that there may have some innate human trait influencing such performance (Pressman, 1992). Thus, although there are only a few efforts in this field, Weinberg (1971-1998) devotes a whole study to the importance of psychology in programming. An entire unit of his work is given over to discussing factors which might be influencing the individual programming performance, and includes a chapter on personality.

Discussing personality, Weinberg highlights certain crucial characteristics in a programmer. Traits such as suspicion, trustfulness, emotional stability, tolerance, tidiness, sense of humour, humility and assertiveness are most looked for in programmers (Weinberg 1971-1998). He adds that suspicion is very useful while debugging, as the programmer who has no doubts regarding the program will not succeed in her/his search. On the other hand, when working in groups/teams, Weinberg emphasises that suspicion is a fatal characteristic, as programmers should trust each other.

Weinberg also says that because of frequent changes in the software and the contradictory situations during its production, emotional stability is also very important, helping people to avoid getting stressed, and keeping the workplace stable. Together with this trait should be what Weinberg calls 'egoless' programming, i.e. a programmer should be tolerant enough to accept another's modifications' of her/his coding.

As well as suspicion, trust, stability emotional and tolerance, a certain element of humility is important too. Humility is needed to accept mistakes and criticism. On the other hand, being a bit assertive is essential. An ideal programmer must find equilibrium between humility and assertiveness. Last but not least, programmers must have a sense of humour to share laughs about the mistakes made in programming.

Analysing how programmers debug programs, Gould & Drongowski (1974), noticed that, during an experiment, programmers behaved as if they were "locked in an imaginary room" and they stayed 'there' until they finished debugging the program. In a further study, Gould (1975) also states that programmers prefer not to use computer assistance.

Bishop-Clark and Wheeler (1994) argue that there are certain individual variations in students' success in programming that could be explained as a consequence of personality traits. To illustrate this, Bishop-Clark and Wheeler did an experiment using the MBTI [within 114 college students] According to this study, there is

significant evidence that Sensing people are more successful in writing programs independent of the other three types.

The authors conclude that "sensors prefer to work with observable facts, tend to be focused toward practicality, have a memory for details, and prefer concrete, tangible experiences. Introductory computer programmers tend to be quite concrete and provide tangible feedback when the program runs. The entire process of writing, entering, and debugging programs is oriented toward a person focused on facts and experiences." (Bishop-Clark and Wheeler, 1994, p.336)

In another study of the relationship between cognitive style, personality and computer programming, Bishop-Clark (1995) found that although cognitive style and personality traits are correlated with achievement in computer programming, cognitive style failed to consistently explain why there are such individual differences in programming achievement.

A recent paper by Capretz (2003) analysed the personality types present in software engineering. He found that in a group of 100 software engineers, the most common personality type was ISTJ (24%), which are described by Myers (1990) as serious, quiet, concentrated, logical and realistic. They are also technically-oriented and prefer working with facts and reasons to dealing with people.

Capretz (2003) also lists other types that would make good systems analysts: ESTJ, ESTP, and ENTP – because they have an ability to solve organizational problem

and their communication skills are good also. The INTP are believed to be excellent programmers because they can notice the problem and present a practical solution to it. They should perform better in the earlier phases of programming, and are very good at problem solving, acting like academic researchers.

Although Capretz (2003) analysed the personality types present in the field of software engineering, he did not compare the types which would provide the best professionals, i.e. the professionals' skills and competencies were not analysed by the author, who was interested only in finding the common types in software engineering. He emphasises that although the types and interests do influence career choice, it is not possible to determine whether or not the type will predict performance in the job. However if a series of instruments are developed, it will help the programmer's type [plus in the evaluation of the programming performance] and it can lead to a correlational analysis which may present a type which is responsible for the success/failure in the field of programming. Such a proposal can be found later in this research.

# 4 HYPOTHESES

This project aimed to identify whether there are any personality factors that may influence performance in the code review exercise.

Reviewing code in programs is a specific process in debugging. While debugging means locating and removing existing bugs in computer programs, code review concerns only locating and pointing out the bugs. Sometimes reviewing code is considered even more difficult than debugging itself:

> *... In fact, localising the code where the error actually is, is usually the hardest part of debugging. Thus the real skill of debugging is locating the actual cause of the error given the symptoms" (Winder & Roberts, 1999).*

Adopting Winder and Roberts's argument that the *"real skill"* for bug-finders lies in finding the bugs in a program rather than in removing them, it could be assumed that if a programmer easily finds the bugs, correcting them is unlikely to present a problem.

In any organization however, there will always be people who are good at debugging, while others are bad. As regards the question of what made them good or bad, there may be validity in the suggestion made by Pressman (1992) that personality factors have an influence on job performance. He noticed huge

differences in performance between employees with the same kind of experience and background working for the same company, and these differences motivated this current research to find out what was really behind success or failure in code-review. Considering the above mentioned a number of hypotheses were formulated.

- *Hypothesis 1*: The knowledge acquired in their first year's modules related to JAVA language (students' marks on CSC131 – Introduction to Programming and Software Engineering, and on CSC132 – Algorithms and Data Structures in JAVA) is an important predictor of their performance in finding bugs in programs.
- *Hypothesis 2*: Sensor preference is a good predictor that a person may perform well at the task. As Bishop-Clark proposed: "(…) sensor people may be better at debugging than others" (Bishop-Clark & Wheeler, 1994).
- *Hypothesis 3*: There is a specific 4 letters MBTI type influencing on the task of code review. As Capretz (2003) found that ISTJ is a common type between Software Engineers, this type maybe a predictor of success in the task.

When correlating the results obtained in the bug-finding task with the personality assessment, it will be possible to identify whether there is any personality factor which is really influencing the performance on this task.

# 5 INSTRUMENTS and METHODS

To check if the hypotheses formulated in this research can be supported or otherwise, it was necessary to use standard instruments to analyse the actions related to the research. However before creating or selecting any instrument, it was necessary to list the objectives of the study as follows:

- To measure students' bug-finding ability
- To identify their personality type
- To identify whether the students' performance in finding bugs in computer programs is influenced by any personality factor
- To identify the students' programming abilities and skills

Considering such objectives, a set of three instruments was created and selected. These instruments were given to the participants following a sequence which was expected to have minimal impact on the results of each instrument, i.e. if the code review exercise was administered first, it could be possible that after performing badly on the task, the participant might mark his debugging abilities with low self-confidence when answering the questionnaire.

For this reason, the first measure administered was a demographic questionnaire (details can be found later on this chapter), which was followed by the *code-review task*; the personality assessment was then the last one to be administered. Instruments were therefore not counter-balanced for order and effects.

This research was carried at Newcastle University and the instruments were given on three separate days carefully chosen so that the students (second year – full time) would have more availability in completing the study. Pre-exam dates were avoided and the beginning of semester was chosen so that the students would not be so busy. There was an attempt to perform the same tasks in City University (London) but because fewer students turned up, it was decided to not use the data obtained.

The questionnaire was given in early October 2002, in their first lecture. The *code-review task,* for which students had an extra incentive in the form of a competition prize - was performed in late January, just after the beginning of the second semester. The personality assessment was done in mid March.

It was not compulsory that the students attend for the three tasks, but they received some extra marks for taking part in each of the tasks. The students could also withdraw at any point. After the three tasks were completed, the researcher gained access to some students' first years' marks for two modules related to the JAVA language (CSC131 – Introduction to Programming and Software Engineering, and CSC132 – Algorithms and Data Structures in JAVA). With these marks in hand, a comparison and analysis of the data obtained in the questionnaire and on the bug-finding task could be performed.

All data obtained in this research was analysed with SPSS software. The most-used techniques were Pearson's correlations (bivariate), regressions (linear), and T-tests.

The study was introduced to the students by their CSC226 – Group Project lecturer. In addition, the researcher sent e-mails to the CSC226 newsgroup inviting them to take part in each of the three studies. Each task was explained and the extra marks were always mentioned. As the experiment was performed during their normal lecture scheme, the number of missing students would not be significant, i.e. even if some students decided not to take part in the study, most would be there. As they had to sign the attendance list, they would prove they were doing the task and claim their extra marks. The researcher's e-mail address was given to the students in case they should require any additional information.

There are two outstanding points which are necessary to explain about this group. The first is that it was agreed between the researcher and the lecturers on the Group Project (who gave the researcher some of their slots to be used for the experiments) that the students would receive extra marks in their discipline (CSC226 – Group Project) if they took part in the research. Second, the students were not obliged to take part in the research, and were free to do any, all or none of the three parts of the study. The extra marks referred to the tasks performed only.

## 5.1 Questionnaire

### 5.1.1 Design

This questionnaire was designed with several distinct purposes in mind. The main purpose of this instrument was the understanding of students' skills; further questions focused on programming skill and on how the students self-assessed their ability in debugging programs.

Personal data about the students were obtained in the last part of this instrument. This part included questions such as age, programme of study (as there may be a slight difference between programme of study, PS, and their performance at programming), and some additional information about family background (do the students have a parent working in the area of Computing Science?) as well as if they had computer access at home.

### 5.1.2 Participants

The questionnaire was completed by 101 students; their ages ranged from 18-47, but the students were mostly distributed in the range 19years-old (39.6%), 20years-old (25.7%) and 21 years-old (11.9%). Over 80% of the students were male.

The students' nationalities was as follows 72.3% of them were British, 8.9% were European, 8.9% Asian and 7.9% were from other nationalities such American and African, and 2% of the data was missing.

Although there was a question about parents' occupations, it was decided not to use this part of the data as the percentage of students who had a father and/or mother who works or worked in computer related jobs was below 5% and did not present any significant result.

Referring to the students' programme of study, almost 49.5% of the students were from Computing Science (CS) and approximately 36.6% were from Information Systems (IS). There were only 13.9% of students from Software Engineering (SE). When asked about computer access at home, it was found that only 3% of the students do not have a computer at home, 27.7% have a computer but with no internet access, 32.7% are connected with 56K, and 35.6% with broadband. Only 1% of the data was missing.

### 5.1.3 Apparatus

The questionnaire was organised in a booklet format containing five pages (see Appendix 1). The first page contained the instructions for completing the questionnaire, the terms of privacy/confidentiality, and space for students' identification. This was compulsory, as the data obtained in this questionnaire was to be correlated with further data (the *code review exercise*, personality assessment, and module grades).

Within the questionnaire, there were two types of questions: open questions in which the participants could write their own point of view, and closed questions which could be of two types – multiple-choice or *visual analogue scale*, i.e. VAS.

The questionnaire was divided into four parts: *'programming'* with questions about students' self-assessment report in programming skills; *'working in a group'* and *'pastimes'*, which will not be discussed in this current study; and *'about yourself'*, in which participants were asked to give some additional information about themselves.

Concerning their programming skills, participants were asked about the time spent working on the computer; their preferred part and least preferred aspect of the overall programming task; previous experience in programming; their knowledge before and at the current year of university; and their knowledge of some languages: JAVA, C++ were specified, and two blank spaces where the participants would add other language knowledge were included as an additional tool to help in the understanding of their skills. It was expected that knowledge of other languages would increase their performance at the bug-finding task. Last but not least, students were asked to rate their skill at debugging their own and other people's programs.

The other section of the questionnaire was to get more information about the students; they were asked about age, nationality, siblings, their programme of study, and if they had access to a computer at home.

### 5.1.4 Procedure

The students were informed about the task by their newsgroup mailing list. They knew that the research was part of an activity from the Centre for Software

Reliability, and that if they took part in the research they would receive some extra marks by the end of the year. The questionnaire was set up for their first lecture slot and their lecturer was there to introduce the researcher to them.

After the introductions, the participants were asked to sign an attendance list to guarantee their extra marks. They were then told about the nature of the research, and received some information about how to fill out the questionnaire. This information was also present on the first page of the questionnaire.

The students then received a five page questionnaire. They were asked to read the instruction page, and if they had any questions they were able to ask the researcher. It was also explained that the questionnaire was an individual exercise, and that communication between them would disturb others. The maximum time slot to answer the questionnaire was one hour, but the students took no more than half an hour to finish it. The participants were dismissed when the researcher noticed that the majority of the participants had already finished with the questionnaire (so that they would not disturb the others).

## 5.2 Code-Review Exercise

### 5.2.1 Design

To distinguish if the success of finding bugs in computer programs was really influenced by certain personality factors or by certain pastimes it was necessary to develop a computer program with errors (bugs) added to it. This “bugged” program then had to be given to the students to find as many of the bugs as possible.

The program chosen was a pattern search in an ASCII data file. It was written by a skilled programmer of the DIRC project, Dr. L.B. Arief. The language adopted was JAVA as this is the common language for the students of the course – as this language is taught in the first year of the course, the principles and explanations of the language were still fresh in their minds.

The programmer then added sixteen bugs to the program; their degree of difficulty was varied, and was valued by him as follows: there were four bugs considered easy, five of them were valued as medium level of difficulty, four as hard, and three bugs were considered the hardest ones. One required some backtracking to modify a line in the previous page. It was necessary to insert bugs with different levels of difficulty so that only really good code-reviewers would spot the most difficult bugs.

The programs had to be piloted, and as it was necessary to maintain strict confidentiality, two PhD students (who had just graduated at Newcastle University) were chosen to pilot test the program. The time spent for the task was between an hour and an hour and half. They said that the program was satisfactory, but suggested some changes in the manual and in the Application Program Interface (API) of the program. The average of bugs they found was nine and the average of the total score, i.e. the sum of the value of bugs found was twenty-six. The program, its manual and the API can be found in Appendix 2

Apart from the extra marks reward, a competition prize was also included for this task as an additional incentive to avoid the students to cheat during the task and to perform the task with responsibility. This competition was set up between students of the same programme of study in which the three highest scores for each programme (CS, SE, and IS) received Amazon.co.uk vouchers (£30, £20, and £10, for first, second and third place respectively).

### 5.2.2 Participants

There were eighty-eight participants in this task (of whom fourteen did not complete the questionnaire). There were forty-three (three of them with no questionnaire) from CS, thirty-one (seven with no questionnaire) from SE and thirteen (four with no questionnaire) from IS.

With regard to valid data (with questionnaire and bug-finding task): 81.3% of the students were between nineteen and twenty-one years old. There were sixty male students and fourteen female. Relating to their nationality, fourteen out of eighty-eight participants did not answer this question and fifty-seven students were British.

### 5.2.3 Apparatus

A pack containing four single-sided pages of a program and two pages of its manual and API was given to the participants. There was a cover page with instructions for performing the task and details of the rules of the competition prize.

### 5.2.4 Procedure

The students were invited to take part in the second part of the experiment through their newsgroup. A reasonable amount of time was allocated to allow students to see the invitation, as it was sent three weeks before the date of the task.

As in the questionnaire, the participants were told that if they participated in this part of the study they would receive some extra marks in their final marks for the module. The competition prize was also mentioned.

The students had between one and a half and two hours to complete the task. A conference-size room was used, so that they could sit comfortably and keep some distance between one another.

As soon as the students arrived they were requested to sit, and when most were present, the researcher started to give the instructions about the task. They were informed that the objective of the task was to find as many as possible of the errors in the program (the number of errors was not specified), that all bugs were semantic ones, and that they only need to circle or to highlight them, i.e. that they were not required to correct the bugs found.

The researcher emphasised that the task was an individual one, and that the three highest scores of each program of study were going to be rewarded with gift vouchers. They were also informed that they were permitted to leave the class

between one and one and a half hours into the test. The experimenter then started distributing the packs to the participants.

## 5.3 MBTI

### 5.3.1 Participants

Seventy-seven students took part in this session, of whom thirteen had not done the bug-finding task and four had not taken part in the questionnaire. There were thirty-nine Computer Science students, twelve Software Engineering students, twenty-four Information Systems students and two other students (who were not present in any of the other studies) did not specify their programme of study.

Separating the group by gender, there were sixty-three male and fourteen female students. It was possible to divide the group according to their nationality as follows: British - fifty-five students, Asian – nine, European Continent – three, other – five, and five students who did not specify their nationality. It was noticed that during the completion of the MBTI some foreign students (most of them were from Asia) had difficulty in understanding certain words and sentences presented in the questions (these were carefully explained by the researcher).

### 5.3.2 Apparatus

The students received a package containing the MBTI booklet, the answer sheet, an explanatory leaflet about the MBTI, and a letter offering a feedback session with a qualified person.

The MBTI booklet used in this research was the European English Edition - Step 1 of which contains eighty-eight items divided into three parts. Parts I and III refer to how the person usually feels or acts, and Part II refers to the pair of words which most appeal to the person. The answer sheet was a self-scorable version. The cover letter can be found in the Appendix 3.

### 5.3.3 Procedure

While the students were arriving, the researcher asked them to sit down. When most of the students were there, they were told about the nature of the questionnaire, the possibility of having a feedback session, and also that they could keep the letter and leaflet for further information.

The test material was distributed, and the participants were asked to write their names on the answer sheet. They were also informed that although the publications of the researcher was going to keep their identity anonymous, putting their names on the answer sheet was necessary to enable the correlation of this data with the data of the previous tasks.

They were then asked to write the answers only on the answer sheet and to return both the question booklet and the answer sheet to the researcher. They were informed about the time of completion - which usually takes up to 25 minutes, but were told that they were free to use the whole session (one hour) if they wished. As soon as they finished it, they could leave the room.

# 6 RESULTS

The results will be presented following the order in which the tasks were performed, i.e. first the questionnaire, then the bug-finding task, and finally the personality assessment. At the end of this chapter, the hypotheses will be examined in light of the results.

Because the students were not obliged to take part in all 3 studies to guarantee their extra marks, there were some who did not complete all of the tasks. The total number of students who took part was 116 (101 did the first task, 88 the second and 77 the personality assessment) but considering the students who did all three tasks, there were 62 participants, plus two other students who completed the bug-finding task and the personality assessment. The valid number of participants in this group was then 64 students. The raw data of all tasks (including those participants who did not complete all of the tasks) are presented in a CD which can be found on the back-cover of this work.

## 6.1 Questionnaire

With regard to the frequency of studying programming, 64.5% of participants worked on programming “daily”, 32.2% said they did it “two to three times per week”; 1.6% selected “other frequency” and 1.6% of the data was missing. Regarding session times: 41.9% of the participants usually stay in front of the computer for “between 3 and 6 hours”, 45.2% “less than 3 hours”, 4.8% “between 6 and 9 hours”, with an additional 4.8% selecting “more than 9 hours”; 3.2% of the data was missing.

When questioned about their preferred step in programming, 43.5% said they preferred "Coding", 14.5% "Design", 19.4% "Documentation", 11.3% "Debugging", 8.1% "Specification" and 3.2% of the data was missing. The figures relating to the step in which they would prefer *not* to work are as follows: 40.3% , "documentation"; 30.6%, "coding"; 12.9%, "specification"; 12.9% said "debugging" and 1.6% "design"; 1.6% of the data was missing (Figure 5).

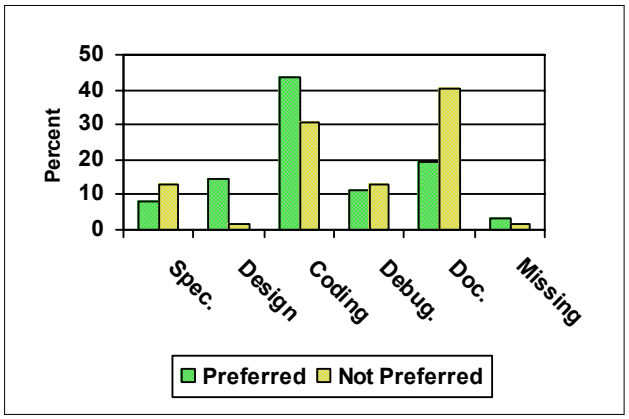


**Figure 5: Percentages of Preferred and Not Preferred Steps in Programming**

From these 62 students only a small number (12.9%) had had some kind of work experience related to computer programming, and they can be grouped according to the duration of this experience as follows: 37.5% had worked less than a year, 37.5% for one year, and 25% for 2 years.

Concerning their programming knowledge, students were asked to mark on an VSA [where the extreme left (0.0) meant "no knowledge" and the extreme right (10.0) meant "expert", see Appendix 1] their degree of knowledge in both situations: before entering the university and at their current period of study (2nd year).

To better visualise the results, the data were truncated into 10 blocks, in which the value 1 denotes the marks between 0.0 and 0.9; 2 corresponds to the marks between 1.0 and 1.9 and so on until block 10.0 which has the marks between 9.0 and 10.0. This grouping system is used for all rating scale data obtained in this questionnaire and is only used to describe the results – for a precise correlation or any other statistical analysis, the decimal values were adopted.

Relating to the students knowledge the data ranged from 0.0 to 7.5 when the students were referring to their knowledge of computer programming before starting the university and from 2.2 to 8.1 when referring to their knowledge at the second year of programming (see figure 6). The raw data for students' knowledge before and at the second year of programming can be found in Appendix 4 and Appendix 5.

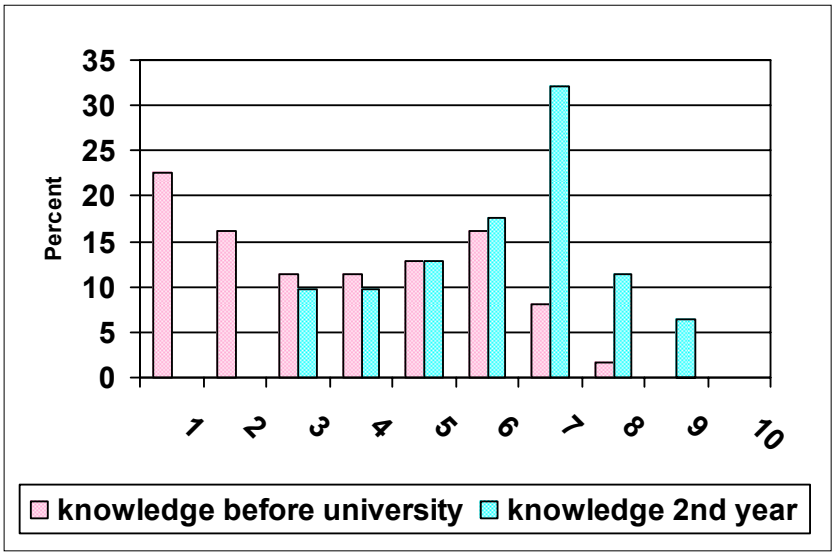


**Figure 6: Student's self rated Knowledge before University and at the 2nd year.**

Regarding their experience with JAVA, which was the common language for all the students who took part in this research, 59.7% of them said they had less than 1 year of experience, 37.1% between 1 and 3 years and only 3.2% more than 3 years of experience in this language.

Their skill in JAVA as well as how much they like/dislike working in JAVA was obtained from VAS. The data was also grouped in such a way as to facilitate the visualisation.

In general, the students' JAVA skill ranged from 1.7 to 8.4. Only 1.6% of the participants said that their JAVA skill was 2; 9.7% scored their skill in JAVA as 3; 25.8% marked 4; 17.7% - 5, 11.3% - 6; 21% scored 7; 11.3% marked 8 and only 1.6% marked their knowledge as 9.

When the students were questioned about whether they like working with JAVA, some students (14.5%) said they did not like working with JAVA and they rated value 2 or 1; some other students, 12.9%,achieved valuation 3 or 4; 11.3% of the students appeared to be indifferent to this language as they evaluated their liking as 5; then it was possible to notice an increased number of students evaluating as 6, 7 and 8, with scores of 14.5%, 21.0% and 9.7% respectively. Some students, 16.2%, had evaluated their skill to this language as 9 and 10. The raw data for students' skill in JAVA and how much they like working on JAVA can be found in Appendix 6 and Appendix 7. The students' knowledge of C++ language is

presented in Appendix 8 and Appendix 9. Student's knowledge in other languages can be found in the CD attached.

The students' debugging skill was also measured on a rating scale, and marked according to the students' self assessment of how easy/difficult it was for them to debug their own program, as well as debugging other people's (the extreme left meant very difficult and extreme right – very easy). Again to better visualise the marking the results were truncated into 10 blocks. The raw data is in Appendix 10 and Appendix 11.

When asked about their ability to debug their own program, 55.7% of the participants marked between 5 and 7 (see Table 1 for more results). With regard to debugging other people's programs, 17.7% selected value 3, 12.9% rated their ability as 4, 14.5% choose 6 and a further 19.4% selected 7. Table 2 presents more results on the participants' abilities at debugging other people's programs.

**Table 1: Grouped Data – Debugging own program**

| | | Frequency | Percent | Valid Percent | Cumulative Percent |
|---|---|---|---|---|---|
| Valid | 2 | 3 | 4.8 | 4.9 | 4.9 |
| | 3 | 5 | 8.1 | 8.2 | 13.1 |
| | 4 | 8 | 12.9 | 13.1 | 26.2 |
| | 5 | 13 | 21.0 | 21.3 | 47.5 |
| | 6 | 13 | 21.0 | 21.3 | 68.9 |
| | 7 | 8 | 12.9 | 13.1 | 82.0 |
| | 8 | 11 | 17.7 | 18.0 | 100.0 |
| | Total | 61 | 98.4 | 100.0 | |
| Missing | System | 1 | 1.6 | | |
| Total | | 62 | 100.0 | | |

**Table 2: Grouped Data - Debugging Other's People Program**

| | | Frequency | Percent | Valid Percent | Cumulative Percent |
|---|---|---|---|---|---|
| Valid | 2 | 6 | 9.7 | 9.8 | 9.8 |
| | 3 | 11 | 17.7 | 18.0 | 27.9 |
| | 4 | 8 | 12.9 | 13.1 | 41.0 |
| | 5 | 6 | 9.7 | 9.8 | 50.8 |
| | 6 | 9 | 14.5 | 14.8 | 65.6 |
| | 7 | 12 | 19.4 | 19.7 | 85.2 |
| | 8 | 5 | 8.1 | 8.2 | 93.4 |
| | 9 | 1 | 1.6 | 1.6 | 95.1 |
| | 10 | 3 | 4.8 | 4.9 | 100.0 |
| | Total | 61 | 98.4 | 100.0 | |
| Missing | System | 1 | 1.6 | | |
| Total | | 62 | 100.0 | | |

With the data obtained from the questionnaire it was possible to draw some conclusions with regard to the characteristics of the participants. When cross-tabulating data, for example, it was possible to find additional information about how long the students practice on computers. It can be seen in the graph below (Figure 7), that most of them study daily, and usually spend no more than 6 hours per session of study.

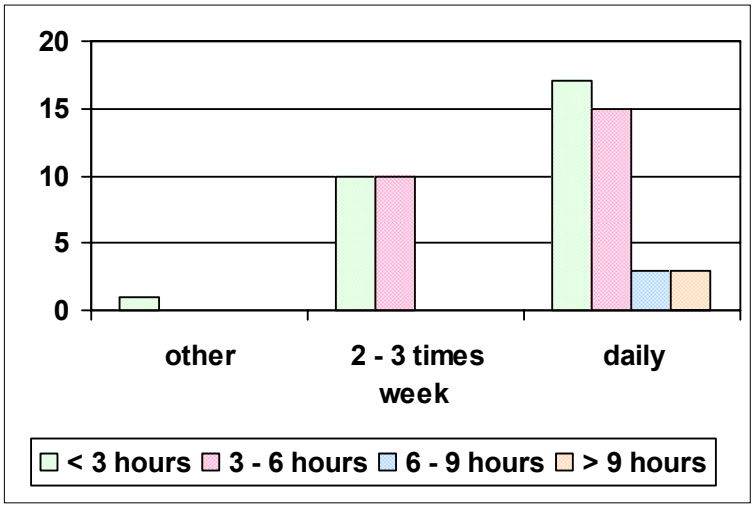


**Figure 7: Frequencies of Studying Programming and Time Spent per Session.**

Another finding was related to their preferred (axis x) and not-least (bars) step in the programming process. In Figure 8, it is possible to see that those who like doing

design do not like debugging. On the other hand, most people who said that coding was their preferred phase of programming do not seem to like documentation. Those who enjoy working on documentation generally do not like doing coding.

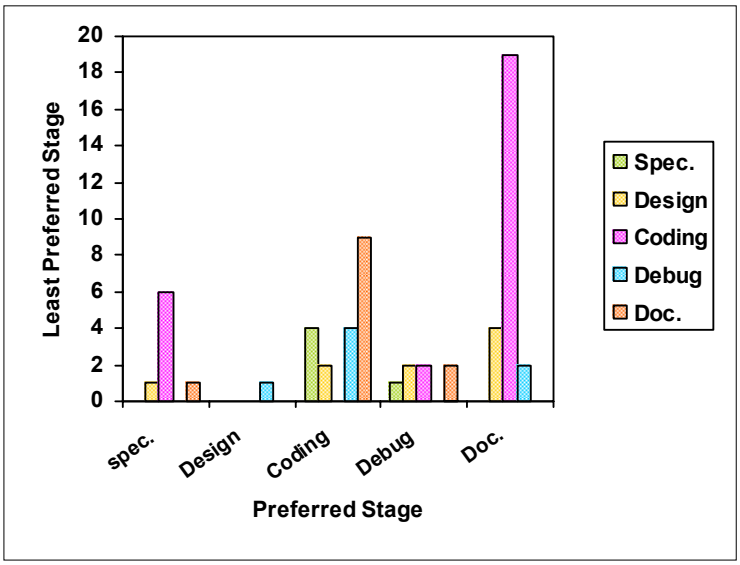


**Figure 8: Preferred and Not-Preferred Step in Programming Process**

By cross-tabulating data from the preferred step in programming and the students' programme of study, the following table describing the students' preferred step according to their programme of study could be drawn (Table 3).

**Table 3: Preferred Step in Programming and Students' Programme of Study**

| Preferred Step | Programme of Study | | | Total |
|---|---|---|---|---|
| | CS | SE | IS | |
| Spec | 1 | 0 | 4 | 5 |
| Design | 6 | 2 | 1 | 9 |
| Coding | 20 | 7 | 0 | 27 |
| Debugging | 3 | 0 | 4 | 7 |
| Doc | 2 | 0 | 10 | 12 |
| Total | 32 | 9 | 19 | 60 |

By correlating the data relating to students knowledge and their ability to debug programs, it was found that when the students rated their ability in debugging other people's programs, there were no significant correlations with their computing

knowledge at all. However there were significant correlations between the students' self rating of ability in debugging their program and their self-rating of knowledge on programming. See Table 4 for Pearson's correlations. There were no significant results when correlating the ability on debugging other people's program and the students' programming knowledge.

**Table 4: Students' Knowledge and Ability to Debug own program**

| | | Debug Own |
|---|---|---|
| Knowledge Before | Pearson Correlation | .516** |
| | Sig. (2-tailed) | .000 |
| | N | 61 |
| Knowledge 2nd year | Pearson correlation | .570** |
| | sig. (2-tailed) | .000 |
| | n | 61 |

** Correlation is significant at the 0.01 level (2-tailed).

### 6.2 Code-Review Task

From the 64 participants who completed this task, the 2 extra students were from CS and IS. It was important to know each student's programme of study, as the competition was set up according to their programme of study.

The total number of bugs present in the program was 16. The maximum number of bugs found in the task was 10 – only by one student; this was followed by 8 bugs, found by 4 students. Most of the participants – 42 students – found between 1 and 4 bugs, and 9 participants did not find any bugs at all. See Table 5 for more details about the bugs found in this task.

The bugs were weighted according to the proportion in which they were found (in the whole sample as there was the prize competition). The ones which were found by many participants were considered easy bugs, and received a value of 2; the

others which were found by some but not many participants were considered of medium difficulty, and were valued as 3; and the hardest ones, which were found by only a few participants received a value of 4. Figure 9 shows the distribution of bugs found according to their difficulty level – exceptionally, the sample in this table is of 88 participants (see dark bars in the chart) but an overlap with the 64 participants was included (see pale bars in the chart). See the attached CD for individuals' performances in this task.

**Table 5: Percentage of bug founds (bf) during the Code review exercise**

| Bugs found | Frequency | Percent | Cumulative Percent |
|---|---|---|---|
| 0 | 9 | 14.1 | 14.1 |
| 1 | 8 | 12.5 | 26.6 |
| 2 | 17 | 26.6 | 53.1 |
| 3 | 7 | 10.9 | 64.1 |
| 4 | 10 | 15.6 | 79.7 |
| 5 | 5 | 7.8 | 87.5 |
| 6 | 2 | 3.1 | 90.6 |
| 7 | 1 | 1.6 | 92.2 |
| 8 | 4 | 6.3 | 98.4 |
| 10 | 1 | 1.6 | 100.0 |
| Total | 64 | 100.0 | |

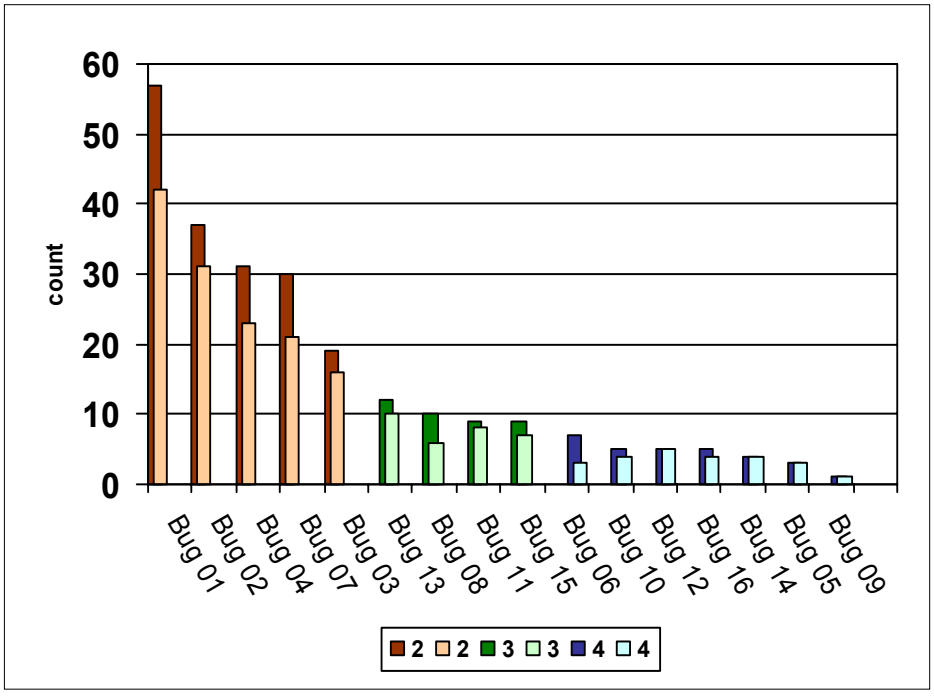


**Figure 9: Proportion of bugs found and their values according to difficulty level.**

Apart from finding bugs in the program, some participants suggested changes where there was no error: their corrections would have actually inserted bugs! These are referred to as "false bugs" (fb). The maximum number of fb found was a total of 13 false bugs found by 1 participant. Many of the participants – 31 – found between 2 and 4 false bugs, and a few students (N= 11) did not mark any false bugs at all.

The cross-tabulation of bugs and false bugs found in the task is shown in Table 6. The participant who found 10 bugs also found 5 false bugs.

**Table 6: Distribution Bugs Found and False Positives - Code review exercise**

| | NUMBER OF FALSE BUGS | | | | | | | | | | | Participants |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BUG FOUND | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 10 | 13 | |
| 0 | | 1 | 3 | 1 | | | 2 | 2 | | | | 9 |
| 1 | 1 | | 1 | | 5 | | | | 1 | | | 8 |
| 2 | 5 | 3 | 1 | 4 | 1 | | 1 | | 1 | | 1 | 17 |
| 3 | 1 | | | 2 | 1 | 2 | | | | 1 | | 7 |
| 4 | 3 | | 2 | | 2 | | 3 | | | | | 10 |
| 5 | | | 1 | 1 | 2 | 1 | | | | | | 5 |
| 6 | | 1 | | 1 | | | | | | | | 2 |
| 7 | | | | | 1 | | | | | | | 1 |
| 8 | 1 | 1 | 2 | | | | | | | | | 4 |
| 10 | | | | | | 1 | | | | | | 1 |
| Total | 11 | 6 | 10 | 9 | 12 | 4 | 6 | 2 | 2 | 1 | 1 | 64 |

As mentioned before, a competition was set up for this task. The 3 highest scorers in each programme of study received gift vouchers of £30, £20 and £10, according to their classification. The students were classified according to their total score, i.e. the sum of the weighted bugs found in the task.

In Table 7, the final results of the code-review task (only for the 3 highest scores of each programme of study) are shown. As there were two winners who did not complete the 3 tasks or at least the MBTI, they are italicised and shown in a

different font for emphasis in the Table that follows. A complete table can be found in Appendix 12.

**Table 7: Students / Marking score prize competition.**

| Part. # | Programme Study | Bug Found | **Total Score** | False Bugs | Prize |
|---|---|---|---|---|---|
| 9 | SE | 10 | **27** | 5 | £30 |
| 52 | CS | 8 | **25** | 2 | £30 |
| 5 | CS | 8 | **22** | 0 | £20 |
| 82 | CS | 8 | **20** | 2 | £10 |
| 48 | SE | 6 | **15** | 3 | £20 |
| *110* | *SE* | *5* | *13* | *5* | *£10* |
| *91* | *IS* | *4* | *11* | *13* | *£30* |
| 21 | IS | 4 | **8** | 0 | £20 |
| 67 | IS | 3 | **8** | 10 | £10 |

Although the false bugs found in the task are shown in the table, they were not counted in the evaluation of working, as in code review the bugs are not repaired, but merely located. A false response will not alter the program, as the programmer and code reviewer can discuss "false bugs" before making any changes (correcting or removing the bug) to the program (see Appendix 13 for the percentage of false bugs). Besides, considering the presence of false bugs and doing analysis in this matter would require much more work and time for an MPhil.

An analysis of variance was carried out with the objective of comparing the performance of the students in this task with that in their first year modules. There were significant variations while comparing their performance on their first year modules and in the code-review task according to their programme of study (Table 8).

**Table 8: ANOVA – Students' Performance and their Programme of Study.**

| | | Sum of Squares | df | Mean Square | F | Sig. |
|---|---|---|---|---|---|---|
| TOSCORE | Between Groups | 601.611 | 2 | 300.805 | 10.318 | .000 |
| | Within Groups | 1778.327 | 61 | 29.153 | | |
| | Total | 2379.938 | 63 | | | |
| CSC131 | Between Groups | 4434.190 | 2 | 2217.095 | 13.021 | .000 |
| | Within Groups | 7321.919 | 43 | 170.277 | | |
| | Total | 11756.109 | 45 | | | |
| CSC132 | Between Groups | 2889.109 | 2 | 1444.555 | 7.249 | .002 |
| | Within Groups | 8369.469 | 42 | 199.273 | | |
| | Total | 11258.578 | 44 | | | |

In order to work out whether the differences in the students' performances have a significant relationship with their background, a Scheffe multiple comparison was done according to their programme of study. Table 9 shows that IS students' performance in the code review task and in the CSC modules (131 and 132) was significantly inferior to CS and SE students.

**Table 9: ANOVA - multiple comparisons – PS and students' performance**

**Scheffe**

| Dependent Variable | (I) PROG. STUDY | (J) PROG. STUDY | Mean Difference (I-J) | Std. Error | Sig. | 95% Confidence Interval | |
|---|---|---|---|---|---|---|---|
| | | | | | | Lower Bound | Upper Bound |
| TOTAL SCORE | CS | SE | -2.79 | 2.018 | .389 | -7.86 | 2.27 |
| | | IS | 5.73(*) | 1.513 | .002 | 1.93 | 9.53 |
| | SE | CS | 2.79 | 2.018 | .389 | -2.27 | 7.86 |
| | | IS | 8.52(*) | 2.167 | .001 | 3.08 | 13.96 |
| | IS | CS | -5.73(*) | 1.513 | .002 | -9.53 | -1.93 |
| | | SE | -8.52(*) | 2.167 | .001 | -13.96 | -3.08 |
| CSC 131 | CS | SE | -2.91 | 4.923 | .841 | -15.39 | 9.58 |
| | | IS | 30.69(*) | 6.275 | .000 | 14.78 | 46.60 |
| | SE | CS | 2.91 | 4.923 | .841 | -9.58 | 15.39 |
| | | IS | 33.60(*) | 7.278 | .000 | 15.15 | 52.05 |
| | IS | CS | -30.69(*) | 6.275 | .000 | -46.60 | -14.78 |
| | | SE | -33.60(*) | 7.278 | .000 | -52.05 | -15.15 |
| CSC 132 | CS | SE | -3.89 | 5.326 | .768 | -17.40 | 9.63 |
| | | IS | 26.78(*) | 7.486 | .004 | 7.78 | 45.78 |
| | SE | CS | 3.89 | 5.326 | .768 | -9.63 | 17.40 |
| | | IS | 30.67(*) | 8.483 | .003 | 9.14 | 52.19 |
| | IS | CS | -26.78(*) | 7.486 | .004 | -45.78 | -7.78 |
| | | SE | -30.67(*) | 8.483 | .003 | -52.19 | -9.14 |

* The mean difference is significant at the .05 level.

## 6.3 MBTI

Some interesting findings emerged when the students' responses to the MBTI were marked. All types were represented in this sample, perhaps because of the

number of participants, but some of the types were represented by fewer students than others (see Appendix 14 and 15).

As the combination of factors SN and TF are responsible for the people's choice of work, a Chi-Square analysis was performed to verify the proportion of each of the 4 possible combinations, i.e. **SF, ST, NF** and **NT** ($X^2$= 6.625, df= 3 and Sig= 085). In Table 10 can be noted that whilst the number of **ST** and **NT** are higher than expected this is not statistically significant.

**Table 10: Distribution of the SN and TF types combinations.**

| | Observed N | Expected N | Residual |
|---|---|---|---|
| ST | 21 | 16.0 | 5.0 |
| SF | 15 | 16.0 | -1.0 |
| NT | 20 | 16.0 | 4.0 |
| NF | 8 | 16.0 | -8.0 |
| Total | 64 | | |

Apart from analysing the 4 letter types (combinations of the 4 bipolar factors – EI, SN, TF, and JP), the data obtained in this task was also analysed according to the continuous scale of the bipolar factors, or set of preferences, which was analysed and scored as a percentage, with 0 indicating an extreme I, and 100 an extreme E in the case of an EI scale. This means that a person will experience more difficulty in dealing with situations which require the opposite behaviour. A score around 50 (from 45 to 55) would indicate a balanced preference, i.e. more flexibility in dealing with both sides. A complete table with each participant's type in the scales can be found in the Appendix 16.

## 6.4 Testing the Hypotheses

### 6.3.1 Hypothesis One

This hypothesis stated that the students' performance in their first year module related to JAVA language (CSC131 – Introduction to Programming and Software Engineering, and CSC132 – Algorithms and Data Structures in JAVA; for the marks see Appendix 15) was a good predictor of their performance in the bug finding task. It was carried out a Pearson's Correlations and it was found that the students' performance in their first year modules had a significant correlation with their performance in the bug finding task, as can be seen in Table 11:

**Table 11: First Year Marks and Performance at Code-Review task**

| | | CSC131 | CSC132 |
|---|---|---|---|
| BUGFOUND | Pearson Correlation | .674** | .619** |
| | Sig. (2-tailed) | .000 | .000 |
| | N | 46 | 45 |
| TOTAL SCORE | Pearson Correlation | .622** | .569** |
| | Sig. (2-tailed) | .000 | .000 |
| | N | 46 | 45 |

** Correlation is significant at the 0.01 level (2-tailed)

To confirm it, a Partial Correlation was also carried out and as can be seen in Table 12 there was a strong correlation too which could support the hypothesis.

**Table 12: Partial Correlations - Task Performance and First Year Marks**

| | | CSC131 | CSC132 |
|---|---|---|---|
| BUGFOUND | Coefficient | .6561 | .6186 |
| | D/F | 43 | 43 |
| | 2-tailed Significance | P= .000 | P= .000 |
| TOTAL SCORE | Coefficient | .6048 | .5693 |
| | D/F | 43 | 43 |
| | 2-tailed Significance | P= .000 | P= .000 |

### 6.3.2 Hypothesis Two

This second hypothesis stated that the students' personality preference for perceiving the world using the five common senses would result in more bugs

being found than was the case with those students who used intuition alone. To test this hypothesis, a one-tailed correlation was performed.

**Table 13: Code-Review Performance and "Sense and Intuition".**

| | | Scale Sensor/Intuition |
|---|---|---|
| TOTAL SCORE | Pearson Correlation | -.246* |
| | Sig. (1-tailed) | .025 |
| | N | 64 |

* Correlation is significant at the 0.05 level (1-tailed).

As can be seen in the table above, it can be observed that there was a negative significance which means that Intuitive students had a better performance than the Sensor students. The hypothesis then could not be confirmed.

### 6.3.3 Hypothesis Three

This last hypothesis stated that there would be a specific personality type which would perform significantly better than others at the code-review task. However, because the sample was small (64 participants) and the personality given by the MBTI were 16 distinct types, it was impossible to determine which one was directly influencing the performance in the task of code review.

It was possible, however, to carry out a cross-tabulation when combining Perception – SN, and Judgment – TF factors, (see Chapter 3, 3.4 – 'The Assessment of Personality – MBTI), students who have a NT preference performed better than those with other preferences (Table 14). In order to test the significance, a one way ANOVA was carried out. This yielded a non-significant result (df= 63, F=1.970 and Sig 2tailed=.128). However to compare NT's versus non-NT's a T-Test was carried out. As hypothesised, NT's performed better than

non-NT's. The results show that this was a significant difference (t= 1.850, df= 62 and Sig. 1tailed= .035)

**Table 14: SN and TF Cross Tabulation – Mean Score (MS) at Code-Review**

| | *T* | *F* | M.S |
|---|---|---|---|
| *S* | 6.62 | 4.27 | *10.89* |
| *N* | **9.10** | 8.13 | *17.23* |
| M.S | *15.72* | *12.40* | |

# 7 DISCUSSION

Before discussing the hypotheses, it is necessary to stress that the sample analysed in this study refers only to a small part of the computer science population. The sample analysed was formed by students from the second year of computing science at Newcastle University and does not represent the professional field meaning that the results presented here cannot be taken as wholly applicable to the selection of new employees.

Before discussing the three hypotheses tested, there will be a brief discussion of the results obtained in the code-review exercise. The students' performance in the task required special attention as the code-review was the major instrument for this study.

## 7.1 Code-Review Task

There are various additional considerations relevant to the program developed for analysis, and consequently to the students' performance in this task. To begin with, it is important to stress that when the program was being developed, its difficulty level was carefully studied, as it was important to introduce bugs of different levels of difficulty.

The intention was that only the hardest bugs would be spotted by those who were really good at code-reviewing. In this sense, there would be a variance between the good, the average and the weak bug-finders. When the task was performed by

the students, there were large variations between the scores. The objective of spotting the good code-reviewers was then achieved.

The variance in performance in the task was a consequence of the different difficulty levels of the bugs. Only the good code-reviewers found the hard bugs, and the easy bugs in the program was found by most of the students. Such variation could have been a consequence of the fact that the chosen program was difficult for the students' level of knowledge.

On the other hand, if the program had been easier, more students would have found the harder bugs, and the results achieved in this research could have been very different. It could be that this procedure would have indicated no relationship at all with the personality types. In this regard, it can be assumed that the level of difficulty of the program was just right, as it made it possible to analyse the personality profiles that are related to good performance in the code-review.

Another point to explain here is related to the considerable quantity of false bugs marked in the task. There were lots of students who marked at least one false bug; only 12 students (from a total of 78 participants) marked none. Unfortunately there was no much time to analyse all the data obtained in the task, in this sense the false bugs were not deeply analysed.

Although there were students who returned the program with few false bugs, others returned the task containing lots of them. The reason for so many false

bugs in a single program could be that the students, although self-rated their skills and abilities in programming 'with high self-confidence', were not so keen when they faced the program. When the number of false bugs marked was not so high, it could be considered that:

1. The program was too difficult for the students' level of knowledge, and that the marking of false bugs was a result of the students' lack of ability to find the correct bugs.
2. The students were only interested in the reward (extra marks for group A, or cash for groups B and C); in this case they did not pay attention to the program, and marked some bits of the code only to avoid returning an unmarked program.

In a real code-review process, the bugs are only identified by the reviewer, i.e. he/she does not make any change in the code. In this current study, it was therefore decided that the presence of false bugs in the code-review exercise was not going to be analysed, as the false bugs would not influence the output of the program. In this case, the presence of false bugs could only suggest that the task was neither an easy task nor that the participants were coping as was expected.

## 7.2 Hypotheses

### 7.2.1 Hypothesis One

This hypothesis stated that the students' performances in their first year module related to JAVA language would be a good predictor of their performance in the bug-finding task. To test this hypothesis Pearson's correlations were carried out to

test the relationship between the students' performance in the code review exercise (number of bugs found and the sum of bugs' weight) and their grades in the first year modules (CSC131 and CSC132). The correlation was significant and high, strong evidence that the students' performance in the task was in fact a reflection of how good they were at the modules in which Java was taught.

In this case, the best students as measured by their stage one exams were better at the code review task, meaning that their performance could have been predicted according to their marks. It would be interesting to check the students' performance in other parts of the programming process as a way to distinguish their abilities and skills.

### 7.2.2 Hypothesis Two

This hypothesis stated that the students' personality preferences for perceiving the world using the five common senses rather than intuition would find more bugs than those who use the intuition rather than the senses. This hypothesis was based on Bishop-Clark and Wheeler's (1994) suggestion that "sensors" would be better at debugging than the "intuitive". It was not possible to support this hypothesis.

Although debugging programs differs from code-review, these tasks (debugging and code-review) present a common characteristic: each aims to locate the bugs in the program. According to Winder and Roberts (1999), the real skill for the debuggers is to find the bugs in the programs. In this sense, it would be possible for debugging and code-review to share the same personality factor, i.e. "sensors".

Bearing in mind that “sensors” would have the highest scores in the task, a 1-tailed Pearson’s correlation with the students’ total score was carried out. It revealed a significant negative correlation which rejected the hypothesis: the better code-reviewers were “intuitive” rather than “sensors”.

“Intuition refers to perception of possibilities, meanings and relationships by way of insight … the intuitive may develop characteristics that can follow from emphasis on intuition and become imaginative, theoretical, abstract, future oriented, or creative.” (Myers & McCaulley, 1985). As ‘intuitives’ are described as those who look through possibilities, the intuitive students were seen to be those looked through many possibilities to find the bugs on the program. These had a better performance than those who just relied on their observation (Sensor’s characteristics).

### 7.2.3 Hypothesis Three

This hypothesis stated that there would be a specific personality type influencing the performance of code-reviewing. This hypothesis could not be deeply analysed as the size of the sample was too small compared with the possible personality types (16).

However, as it was found that the “intuitive” students had a better performance than the “sensor” students, it was suggested that there would be at least one other factor helping the “intuitive” factor to achieve the success in the task. To identify which other factor/preference would be linked to the “intuitive” preference in the

code-review performance, a series of analyses (cross-tabulations, T-Test and ANOVA) was carried out.

There was a significant difference between “intuitive” and “thinking” preferences (NT); in contrast to non-NT’s students in that NT’s had a significantly better performance. The combination of the factors SN (the manner of which people perceive the world) and TF (the way in which they make judgments) is, according to Myers Briggs, the one that determines the job satisfaction and career choice.

The characteristic of NTs (Myers & McCaulley, 1985, page35) is that they perceive the world through their intuition, i.e. they gain their insights from meanings rather than from observation of the facts. They also make judgements by thinking, i.e. by forming logical connections between facts, rather than by weighing the decisions. They are always “looking at the possibilities, theoretical relationships, and abstract patterns” – the way they will make judgements is impersonal. The NTs are known as “logical and ingenious”, and they are best at solving problems (the task of finding bugs in programs could also be considered a problem-solving task) within their field of special interest (in this case, programming).

Although there is no evidence about how the job is performed (well or badly), there is a list of the “occupational choices for NT” (Appendix D, Myers & McCaulley, 1985, p259). Some of them were computer-related jobs, but the programming phases were not specified there. In a sample of 86 computer

systems analysts, there were 41.86% NTs; in another sample of 57 computer specialists, there were 36.98% NTs; and the NTs represented 36.50% of a 200 sample of computer programmers.

The statistics above might suggest that NTs do feel a certain attraction to computer related careers (unfortunately the programming phases and performance are not specified). It may also suggest that there are NTs who are unaware of their potential as good code-reviewers. If a company organises its employees according to their personality types and their potential abilities, productivity could increase and quality improve.

In addition, it can be assumed that as NTs presented particular qualities which contributed to the success in the code review task, there are other types which are influencing the success at the other steps of the programming process. As Weinberg suggests, each part of the programming process requires a specific task with particular "combinations of skills and personality traits" (Weinberg, 1971-1998). Thus it could be useful to look at the other parts of the programming process and the personality types involved.

# 8 CONCLUSION

By the end of this research, it was possible to raise some interesting points and indicate directions for future studies. It was found that the participants' personality types, and their previous performance at the first year's modules, had some influence on their performance at code-review.

Regarding the role of personality in the performance of the code-review task it was possible to conclude that the "Intuitive" (N) people seemed to be better than the "Sensors" (S), the opposite finding predicted by Bishop-Clark and Wheeler (1994).

The "N" preference was not the only one related to success in the task, as the thinking preference "T" when combined with "Intuition" had an important function when the task was performed. The "NTs" were "labelled" as a logical and ingenious type because although they do act according to their logical thinking, their attention is focused on possibilities. They are better at dealing with theoretical and technical developments, and they have good potential for solving problems. In code review, the theoretical portion can be seen as the JAVA concepts learned during lectures, and the technical portion of the code-review as putting all the knowledge acquired into practice.

In addition to the NTs' good performance, their performance could equally be compared with their marks in the first year's modules. This could be another

factor which would help in the prediction of the reviewers' abiliies in carrying out the task. Such findings would not be a useful predictor of their performance if used in isolation, but it could help in the selection of qualified and competent professionals when used in combination.

It would be necessary however to carry out further research, this time with mature professionals rather than students. The results obtained in such research could then be compared with the results of this current research. If it is found that "intuitive" factor is also present in the good code-reviews (in the professional field), it could be concluded that this factor, i.e. the intuitive preference, has an important role in the success of finding bugs in programs.

If it is found that each part of the programming process is different, and involves characteristics unique to itself, it can be concluded that each part of this same process requires specific actions. This suggests a series of experiments to be performed, however, unfortunately, the evaluation of the other programming phases will be much more different than in the code-review task. Given that such tasks are best allocated on the basis of a programmer's personality type, [and then] it could be concluded that each stage will require a particular personality type, as once suggested by Weinberg (1971).

Using the appropriate types in each part of the programming process would increase productivity and consequently reduce the costs and time spent in each stage of the software development process, thus reducing the time and costs, more

money and time could be devoted to dependability and reliability research, so that reliable and dependable software and systems could be developed in shorter time and with lower costs.

It is important to emphasise that other preferences, i.e. ST, SF and NF, can improve employees' ability and potential at code-review, simply by understanding their preferences, i.e. learning their weakness and working on the improvement of it.

Considering all of these findings and thoughts about this research, some directions for future research can be proposed:

- A study using professionals. These could be both from the code-reviewer team, and from the program development team. This would support or otherwise the idea that personality type is relevant to the performance of code-review. This study could include a selection of professionals, i.e. from juniors to seniors; so that all levels of experience plus the personality types could be analysed and compared.

- A study checking the personality types involved in other areas of programming. This research has to be carried out within other areas of computer programming as a way to discover the personality types related to success in each part of the development process. Again it would be interesting that apart from analysing data from students, junior and senior professionals could also be included. Then the professionals could be

reorganised, i.e. relocated according to their potentials skills. However, in order to do this it would be necessary to certify that the whole sample's performance are analysed in all the stages of programming, which could generate more precise findings, i.e. it can be possible that each stage has more than one personality type influencing the task. It may also be possible that a person performs particularly well in more than one task, which would imply that a certain type can be good at more than one specific task.

# 9 APPENDIX

**Appendix 1: Questionnaire Designed for the Second year Students**

DIRC

**UNIVERSITY OF NEWCASTLE**
**School of Computing Science**
**Centre of Software Reliability**
**DIRC Project**

Questionnaire Number

Name: _____

First of all we would like to thank you for participating in this research and if you would like to receive any further information about this study please give us your e-mail address:
E-mail: _____

Please note, that the information you supply in this and future questionnaires, is *confidential* and your name will only be used to correlate the information you provide in this questionnaire with other data you may provide later. The information you supply is for research purposes only and will not detrimentally affect your grades, nor will it be published in any way which can be associated with you.

In this questionnaire you will find both: closed and open questions. In the closed questions you will be asked to tick (when multiple choice) or to mark (rating scale) the answer that best indicates how you feel in relation to that specific question.
In some specific closed questions you will be asked to detail your answer in a few words too.
For example:

a) Do you have any previous working experience before started the university?
☐ No
☐ Yes – job title: *clothes shop assistant*

b) How did you feel leaving your job and starting the university?
Please note that the extreme left hand side indicates that you didn't feel satisfied and the extreme right hand side indicates that you felt strongly satisfied.


In the open questions you will be asked to specify your answers – but please note that the clearer you are, the more reliable the research will be.
For example:

c) Please specify how long you worked before starting the university: *3 years and 7 months.*

**Programming**

When you are creating programs how often do you work in front of the computer?

| About the frequency: | About the time spent per session: |
|---|---|
| ☐ Daily | ☐ Less than 3 hours |
| ☐ 2-3 times a week | ☐ 3 to 6 hours |
| ☐ Once per week | ☐ 6 to 9 hours |
| ☐ Other: _____ | ☐ More than 9 hours |

Suppose that you and a group of friends are developing a program (any kind of a program) and that you are allowed to choose the step of the programming process in which you would prefer to work. (Please tick one from each column)

| Which one it would be? | In which one would you not prefer to work? |
|---|---|
| ☐ Specification | ☐ Specification |
| ☐ Design | ☐ Design |
| ☐ Coding | ☐ Coding |
| ☐ Testing/Debugging | ☐ Testing/Debugging |
| ☐ Documentation | ☐ Documentation |

Do you have any previous working experience in programming before starting the university?

☐ No

☐ Yes – job title: _____ duration of the job: _____

---

In the next questions you will be asked to answer according to the following scale in which the extreme left hand side indicates *very poor* and the extreme right hand side indicates *expert.*

**1** – Please indicate the level that best describes your knowledge in Computer Programming:

Before you started the university?


Now that you are at attending the second stage of Computing Science?

**2** – How would you rate your knowledge in the **following programming languages**?

***Java:***
What is your experience of Java? (If no experience then proceed to C++)

Please tick:

| ☐ | ☐ | ☐ | ☐ |
|---|---|---|---|
| Less than or up to 1 year | More than 1 but less than 3 years | Between 3 and 5 years | More than 5 years |

How would you rate your skill with regard to programming in Java? (Please mark the line where you think appropriate)

Very poor ——— Expert

How much do you like using this language? (Please mark the line where you think appropriate)

Not at all ——— Very much

---

***C++:***
What is your experience of C++? (If no experience then proceed to *other languages*, below)

Please tick:

| ☐ | ☐ | ☐ | ☐ |
|---|---|---|---|
| Less than or up to 1 year | More than 1 but less than 3 years | Between 3 and 5 years | More than 5 years |

How would you rate your skill with regard to programming in C++? (Please mark the line where you think appropriate)

Very poor ——— Expert

How much do you like using this language? (Please mark the line where you think appropriate)

Not at all ——— Very much

***Other languages:*** *if you have experience in any other languages, please use this space:*

What is your experience of _____?

Please tick:

| ☐ | ☐ | ☐ | ☐ |
|---|---|---|---|
| Less than or up to 1 year | More than 1 but less than 3 years | Between 3 and 5 years | More than 5 years |

How would you rate your skill with regard to programming in this language? (Please mark the line where you think appropriate)

Very poor ——————————— Expert

How much do you like using this language? (Please mark the line where you think appropriate)

Not at all ——————————— Very much

---

What is your experience of _____?

Please tick:

| ☐ | ☐ | ☐ | ☐ |
|---|---|---|---|
| Less than or up to 1 year | More than 1 but less than 3 years | Between 3 and 5 years | More than 5 years |

How would you rate your skill with regard to programming in this language? (Please mark the line where you think appropriate)

Very poor ——————————— Expert

How much do you like using this language? (Please mark the line where you think appropriate)

Not at all ——————————— Very much

---

- In the next 2 questions have to be answered according to the following scale in which the extreme left hand side indicates *very hard* and the extreme right hand side indicates *very easy*:

How hard is it for you to find errors in your own programs?

Very hard ——————————— Very easy

How hard is it for you to find errors in other people's programs?

Very hard ——————————— Very easy

**A little about yourself**

Please take a few moments to fill in some personal details, which will help us put your data in a broader context

Are you the only child in the family?
☐ Yes
☐ No, I am the: ☐ oldest ☐ middle ☐ youngest

Age: _____

Gender: ☐ male ☐ female

Nationality: _____

Please specify the actual (and previous - if necessary) occupation of:
**Your father:**
Current occupation - _____
Previous occupation - _____
**Your mother:**
Current occupation - _____
Previous occupation - _____

Your programme of study (i.e. Computing Science, Information Systems, G402, etc.):_____, in the following stage: _____.

***Computer access at home:***

Please indicate which of the following applies to you:

| | |
|---|---|
| No access to a computer at home | ☐ |
| Access to a computer without internet access | ☐ |
| Access to a computer with 56K dialup | ☐ |
| Access to a computer with broadband | ☐ |

Other (please specify): _____

---

If you have any feedback which you think would improve this questionnaire, please give it here:

---

David.Greathead@ncl.ac.uk – A.Devito-Cunha@ncl.ac.uk

## Appendix 2: Code-Review: instructions, manual, API and JAVA code


**UNIVERSITY OF NEWCASTLE**

**School of Computing Science**

**Centre of Software Reliability**

**DIRC Project**

Name: _____

Programme of Study: _____

Task Number

Thanks for taking part in this research, please read this page carefully and if you have any questions, please contact one of the researchers.

---

Please note, that the results of this task and any other information you provide in the future are *confidential* and your name will only be used to correlate the information you provide in this task with other data. The information you supply is for research purposes only and will not detrimentally affect your grades, nor will it be published in any way which can be associated with you.

---

Remember that there will be a total of 9 Amazon.co.uk vouchers of £30, £20 and £10 to be distributed as follow:

IS - £30, £20 and £10 for the first, second and third respectively

CS - £30, £20 and £10 for the first, second and third respectively

SE - £30, £20 and £10 for the first, second and third respectively

---

INSTRUCTIONS OF THE TASK

This task is a debugging study and all you need to do is find the bugs present in it. All bugs are semantic (**there is no syntax errors**) and you do not need to correct the errors. You have to highlight or circle the bugs you find.

You can use the blank pages of this paper to work on it.

Information about the program can be found in the next 2 pages and no additional information will be given to you.

The time to complete this task is **1h30min**. Please do not leave the class before this time is up.

This task is an individual task so please do not communicate with others for the duration.

**File:** WordSearch.java

**Purpose:**
This program searches for occurrences of a string (which can either be a word or part of a word) from an ASCII file (the data file), and based on the search criteria, prints out a report on the search.

**Inputs:**
- A string to search for
- A choice of whether the search should return only exact word matches or part word matches (i.e. substring matches) as well
- A choice whether the search is case sensitive or not
- The name of the data file on which the search should be performed

**Outputs:**
- An error message if:
  - the data file does not exist
  - there is no occurrence of the searched-for string in the file
- Otherwise, a report is produced, containing:
  - a summary of the search options
  - the line numbers where occurrences of the string are found and how many are found on that line. If part (i.e. not exact) string search was conducted then for each occurrence, the full word should also be printed
  - the total number of occurrences of the searched-for string in the file

**Examples:**
If the data file (called "data.txt") contains:

*Searching, shopping and swapping sites are firm favourites with UK surfers, shows a list of the top websites of 2002 drawn up by net ratings firm Nielsen. There are few surprises in the list and almost all of them will be well known to anyone who spends time online. Search engine Google takes the spot as the top site of the year.*

the following results are obtained:

```
$ java WordSearch
Enter (part of) the word to search: in
Exact Word Match? [Y/N]: n
Case Sensitive? [Y/N]: n
From which file?: data.txt

Searching for "in" in file "data.txt"...

        Found 3 occurence(s) on line 1: Searching shopping
swapping
        Found 1 occurence(s) on line 2: ratings
        Found 1 occurence(s) on line 3: in
        Found 2 occurence(s) on line 4: online engine

The search is case in-sensitive.
The pattern "in" is found 7 times in file "data.txt".

$ java WordSearch
Enter (part of) the word to search: in
Exact Word Match? [Y/N]: y
```

```
Case Sensitive? [Y/N]: n
From which file?: data.txt

Searching for "in" in file "data.txt"...

        Found 1 occurence(s) on line 3

The search is case in-sensitive.
The word "in" is found 1 times in file "data.txt".

$ java WordSearch
Enter (part of) the word to search: In
Exact Word Match? [Y/N]: y
Case Sensitive? [Y/N]: y
From which file?: data.txt

Searching for "In" in file "data.txt"...


The search is case sensitive.
Sorry, there is no "In" in file "data.txt".
```

JAVA substring API

public String **substring**(int beginIndex, int endIndex)

Returns a new string that is a substring of this string. The substring begins at the specified beginIndex and extends to the character at index endIndex - 1. Thus the length of the substring is endIndex-beginIndex.

Examples:

"hamburger".substring(4, 8) returns "urge"

"smiles".substring(1, 5) returns "mile"

**Parameters:**

beginIndex - the beginning index, inclusive.

endIndex - the ending index, exclusive.

**Returns:**

the specified substring.

**Throws:**

IndexOutOfBoundsException - if the beginIndex is negative, or endIndex is larger than the length of this String object, or beginIndex is larger than endIndex.

**Note:**

For the purpose of this test, the program is un-commented and the API is incomplete. This is intentional and constitutes part of the test.

```
import java.io.*;
import java.lang.*;
import java.util.*;

public class WordSearch
{
  final static String[] preDelimList = {" ",
                                        "(",
                                        "{",
                                        "[",
                                        "<",
                                        "-",
                                        "/",
                                        "#",
                                        "'",
                                        "\""};

  final static String[] postDelimList = {" ",
                                         ")",
                                         "}",
                                         "]",
                                         ">",
                                         "-",
                                         "/",
                                         ".",
                                         ",",
                                         ":",
                                         ";",
                                         "?",
                                         "!",
                                         "'",
                                         "\""};

  public static void main (String argv[])
  {
    Vector lines = new Vector();
    String tempString = "";
    String pattern = "";
    String filename = "";
    boolean wordMatch = false;
    boolean caseSensitive = false;
    int counter = 0;

    try
    {
      BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
      System.out.print("Enter (part of) the word to search: ");
      pattern = in.readLine();

      String choice = "";
      do
      {
        System.out.print("Exact Word Match? [Y/N]: ");
        choice = in.readLine().toLowerCase();
      }
      while (! (choice.equals("y") || choice.equals("n")) );
      wordMatch = choice.equals("y");

      do
      {
        System.out.print("Case Sensitive? [Y/N]: ");
        choice = in.readLine().toLowerCase();
      }
      while (! (choice.equals("y") || choice.equals("n")) );
      caseSensitive = choice.equals("y");

      System.out.print("From which file?: ");
      filename = in.readLine();

      File ff = new File(filename);
      FileReader fr = new FileReader(ff);
      BufferedReader fileInput = new BufferedReader(fr);

      while((tempString = fileInput.readLine()) == null)
```

```
    {
      lines.add(tempString);
    }

    in.close();
    fileInput.close();
  }
  catch (IOException e)
  {
    System.out.println("Can't read from file \"" + filename + "\"!\n");
    System.exit(1);
  }

  System.out.println();
  System.out.println("Searching for \"" + pattern + "\" in file \"" +
    filename + "\"...\n");

  for (int i=0; i<=lines.size(); i++)
  {
    String aLine = (String)lines.elementAt(i);

    Vector resultsPerLine = doSearch(aLine, pattern, caseSensitive,
                                     wordMatch);
    int foundPerLine = Integer.parseInt((String)resultsPerLine.elementAt(0));

    if (foundPerLine != 0);
    {
      counter += foundPerLine;
      System.out.print("\tFound " + foundPerLine + " occurence(s) on line "
        + i+1 );

      if (!wordMatch)
      {
        System.out.print(": ");
        for (int j=1; j<resultsPerLine.size(); j++)
        {
          String aWord = (String)resultsPerLine.elementAt(j);
          System.out.print(aWord + " ");
        }
      }
      System.out.println();
    }
  }

  System.out.print("\nThe search is case ");
  if (caseSensitive)
  {
    System.out.println("sensitive.");
  }
  else
  {
    System.out.println("in-sensitive.");
  }

  if (counter > 0)
  {
    if (wordMatch)
    {
      System.out.println("The word \"" + pattern + "\" is found " +
        counter + " times in file \"" + filename + "\".\n");
    }
    else
    {
      System.out.println("The pattern \"" + pattern + "\" is found " +
        counter + " times in file \"" + filename + "\".\n");
    }
  }
  else
  {
    System.out.println("Sorry, there is no \"" + pattern + "\" in file \""
      + filename + "\".\n");
  }
}
```

```
public static boolean isPreDelim(String ch)
{
  for (int i=1; i<preDelimList.length; i++)
  {
    if (ch.equals(preDelimList[i])) return true;
  }
  return false;
}

public static boolean isPostDelim(String ch)
{
  for (int i=0; i<postDelimList.length; i++)
  {
    if (ch.equals(postDelimList[i])) return true;
  }

  return false;
}

public static String getCompletedWord(String src, int index)
{
  int begin = 0;
  int end = src.length();

  int i = index;
  while (i > 0)
  {
    i--;
    String ch = src.substring(i-1, i);
    if (isPreDelim(ch))
    {
      begin = i;
      break;
    }
  }

  while (i<src.length())
  {
    i++;
    String ch = src.substring(i, i+1);
    if (isPostDelim(ch))
    {
      end = i;
      break;
    }
  }

  return (src.substring(begin, end));
}

public static Vector doSearch(String txt, String pat, boolean wordMatch,
                              boolean caseSensitive)
{
  int txtLen = txt.length();
  int patLen = pat.length();
  Vector results = new Vector();
  int totalFound = 0;


  for (int i=0; i<txtLen-patLen; i+=patLen)
  {
    String toCompare = txt.substring(i, patLen);

    if (!caseSensitive)
    {
      toCompare = toCompare.toLowerCase();
      pat = pat.toLowerCase();
    }

    if (toCompare == pat)
    {
      if (wordMatch)
      {
        boolean exactMatch = false;
```

```
          if (patLen == txtLen)
            exactMatch = true;
          else
          {
            String pre = null;
            String post = null;

            if (i > 0)
            {
              pre = txt.substring(i-1, i);
            }
            if (i+patLen <= txtLen)
            {
              post = txt.substring(i+patLen, i+patLen+1);
            }

            if (i == 0)
            {
              if (post != null)
              {
                exactMatch = isPostDelim(post);
              }
              else exactMatch = true;
            }
            else if (i+patLen == txtLen)
            {
              if (pre != null)
              {
                exactMatch = isPreDelim(post);
              }
              else exactMatch = true;
            }
            else
            {
              if (pre != null && post != null)
              {
                if (isPreDelim(pre) && isPostDelim(post))
                  exactMatch = true;
              }
              else exactMatch = true;
            }
          }

          if (exactMatch) totalFound++;
        }
        else
        {
          String completedWord = getCompletedWord(pat, i+1);
          results.add(completedWord);
          totalFound++;
        }
      }
    }

    Integer tempInt = new Integer(totalFound);
    results.insertElementAt(tempInt.toString(), 0);
    return results;
  }
}
```

## Appendix 3: MBTI - Cover Letter



Dear student,

Attached is a Myers-Briggs Type Indicator (MBTI) question booklet and answer sheet for your completion. The MBTI has been subjected to rigorous research tests demonstrating its reliability and validity and it is widely used in management and personal development as well as team building in both private and public sectors. It is a very constructive and positive instrument and usually leaves people feeling good about themselves.

In order to get most benefit from the MBTI please take note of the following:

- The MBTI is **not** in any way a **test**. It is not measuring anything, nor making any value judgements of you as a person, nor in any way saying how 'well' or 'ill' you are psychologically. It seeks to indicate how you like to live your life, and work. You can therefore be totally relaxed as you make your choices.

- The MBTI is a dual choice questionnaire. As you make your choice seek to answer, not necessarily according to a 'work self' or any other role you have, but primarily according to what is most natural for you.

- Answer the questions at one sitting. There is no time limit, but please don't discuss your answers with any other person. If you are unclear of the meaning of a question, or if you cannot decide on an answer, omit it. Try not to have too many omissions however!

- Record your responses on the answer sheet. Please make sure you use a ball point pen and having filled in your name, complete the answer sheet by making an 'x' in the appropriate box. **Please don't make any marks on the question booklet or open the perforations on the answer sheet.**

Please read carefully the instructions on the first page of the booklet before you begin.

If you would like to have individual feedback regarding your MBTI results please contact me via David Greathead e-mail address David.Greathead@ncl.ac.uk or by contacting me direct e-mail address Helen.Doyle@ncl.ac.uk

Best wishes

Helen Doyle
Training Officer
Staff Development Unit

E:\tmp\ForAlelcover_letter_ncl.doc

# QUESTIONNAIRE

## Students Knowledge in Computing Programming before University and at the Current Year.

**Appendix 4: Students' self-rated Programming Knowledge – Before University**

| | | Frequency | Percent | Valid Percent | Cumulative Percent |
|---|---|---|---|---|---|
| Valid | .0 | 2 | 3.1 | 3.2 | 3.2 |
| | .2 | 4 | 6.3 | 6.5 | 9.7 |
| | .3 | 2 | 3.1 | 3.2 | 12.9 |
| | .4 | 1 | 1.6 | 1.6 | 14.5 |
| | .5 | 1 | 1.6 | 1.6 | 16.1 |
| | .6 | 1 | 1.6 | 1.6 | 17.7 |
| | .7 | 1 | 1.6 | 1.6 | 19.4 |
| | .8 | 1 | 1.6 | 1.6 | 21.0 |
| | .9 | 1 | 1.6 | 1.6 | 22.6 |
| | 1.0 | 1 | 1.6 | 1.6 | 24.2 |
| | 1.2 | 3 | 4.7 | 4.8 | 29.0 |
| | 1.3 | 1 | 1.6 | 1.6 | 30.6 |
| | 1.4 | 1 | 1.6 | 1.6 | 32.3 |
| | 1.7 | 1 | 1.6 | 1.6 | 33.9 |
| | 1.8 | 2 | 3.1 | 3.2 | 37.1 |
| | 1.9 | 1 | 1.6 | 1.6 | 38.7 |
| | 2.1 | 1 | 1.6 | 1.6 | 40.3 |
| | 2.2 | 1 | 1.6 | 1.6 | 41.9 |
| | 2.4 | 2 | 3.1 | 3.2 | 45.2 |
| | 2.5 | 2 | 3.1 | 3.2 | 48.4 |
| | 2.9 | 1 | 1.6 | 1.6 | 50.0 |
| | 3.0 | 1 | 1.6 | 1.6 | 51.6 |
| | 3.1 | 1 | 1.6 | 1.6 | 53.2 |
| | 3.4 | 1 | 1.6 | 1.6 | 54.8 |
| | 3.5 | 1 | 1.6 | 1.6 | 56.5 |
| | 3.7 | 3 | 4.7 | 4.8 | 61.3 |
| | 4.1 | 3 | 4.7 | 4.8 | 66.1 |
| | 4.4 | 3 | 4.7 | 4.8 | 71.0 |
| | 4.6 | 1 | 1.6 | 1.6 | 72.6 |
| | 4.9 | 1 | 1.6 | 1.6 | 74.2 |
| | 5.3 | 1 | 1.6 | 1.6 | 75.8 |
| | 5.4 | 1 | 1.6 | 1.6 | 77.4 |
| | 5.6 | 1 | 1.6 | 1.6 | 79.0 |
| | 5.7 | 1 | 1.6 | 1.6 | 80.6 |
| | 5.8 | 3 | 4.7 | 4.8 | 85.5 |
| | 5.9 | 3 | 4.7 | 4.8 | 90.3 |
| | 6.0 | 1 | 1.6 | 1.6 | 91.9 |
| | 6.3 | 1 | 1.6 | 1.6 | 93.5 |
| | 6.8 | 1 | 1.6 | 1.6 | 95.2 |
| | 6.9 | 2 | 3.1 | 3.2 | 98.4 |
| | 7.5 | 1 | 1.6 | 1.6 | 100.0 |
| | Total | 62 | 96.9 | 100.0 | |
| Missing | System | 2 | 3.1 | | |
| Total | | 64 | 100.0 | | |

**Appendix 5: Students' self rated Programming Knowledge – 2nd Year**

| | | Frequency | Percent | Valid Percent | Cumulative Percent |
|---|---|---|---|---|---|
| Valid | 2.2 | 1 | 1.6 | 1.6 | 1.6 |
| | 2.6 | 1 | 1.6 | 1.6 | 3.2 |
| | 2.7 | 2 | 3.1 | 3.2 | 6.5 |
| | 2.8 | 1 | 1.6 | 1.6 | 8.1 |
| | 2.9 | 1 | 1.6 | 1.6 | 9.7 |
| | 3.2 | 1 | 1.6 | 1.6 | 11.3 |
| | 3.3 | 1 | 1.6 | 1.6 | 12.9 |
| | 3.5 | 2 | 3.1 | 3.2 | 16.1 |
| | 3.8 | 1 | 1.6 | 1.6 | 17.7 |
| | 3.9 | 1 | 1.6 | 1.6 | 19.4 |
| | 4.0 | 1 | 1.6 | 1.6 | 21.0 |
| | 4.5 | 3 | 4.7 | 4.8 | 25.8 |
| | 4.8 | 1 | 1.6 | 1.6 | 27.4 |
| | 4.9 | 3 | 4.7 | 4.8 | 32.3 |
| | 5.0 | 1 | 1.6 | 1.6 | 33.9 |
| | 5.1 | 2 | 3.1 | 3.2 | 37.1 |
| | 5.2 | 4 | 6.3 | 6.5 | 43.5 |
| | 5.4 | 1 | 1.6 | 1.6 | 45.2 |
| | 5.6 | 1 | 1.6 | 1.6 | 46.8 |
| | 5.7 | 1 | 1.6 | 1.6 | 48.4 |
| | 5.9 | 1 | 1.6 | 1.6 | 50.0 |
| | 6.0 | 2 | 3.1 | 3.2 | 53.2 |
| | 6.1 | 1 | 1.6 | 1.6 | 54.8 |
| | 6.2 | 4 | 6.3 | 6.5 | 61.3 |
| | 6.3 | 1 | 1.6 | 1.6 | 62.9 |
| | 6.4 | 2 | 3.1 | 3.2 | 66.1 |
| | 6.5 | 1 | 1.6 | 1.6 | 67.7 |
| | 6.6 | 1 | 1.6 | 1.6 | 69.4 |
| | 6.7 | 3 | 4.7 | 4.8 | 74.2 |
| | 6.8 | 3 | 4.7 | 4.8 | 79.0 |
| | 6.9 | 2 | 3.1 | 3.2 | 82.3 |
| | 7.0 | 2 | 3.1 | 3.2 | 85.5 |
| | 7.1 | 2 | 3.1 | 3.2 | 88.7 |
| | 7.2 | 1 | 1.6 | 1.6 | 90.3 |
| | 7.3 | 1 | 1.6 | 1.6 | 91.9 |
| | 7.8 | 1 | 1.6 | 1.6 | 93.5 |
| | 8.0 | 3 | 4.7 | 4.8 | 98.4 |
| | 8.1 | 1 | 1.6 | 1.6 | 100.0 |
| | Total | 62 | 96.9 | 100.0 | |
| Missing | System | 2 | 3.1 | | |
| Total | | 64 | 100.0 | | |

## Students and the JAVA Language

**Appendix 6: Participants self-rated JAVA Skill**

| | | Frequency | Percent | Valid Percent | Cumulative Percent |
|---|---|---|---|---|---|
| Valid | 1.7 | 1 | 1.6 | 1.6 | 1.6 |
| | 2.0 | 1 | 1.6 | 1.6 | 3.2 |
| | 2.2 | 1 | 1.6 | 1.6 | 4.8 |
| | 2.4 | 1 | 1.6 | 1.6 | 6.5 |
| | 2.5 | 1 | 1.6 | 1.6 | 8.1 |
| | 2.7 | 1 | 1.6 | 1.6 | 9.7 |
| | 2.9 | 1 | 1.6 | 1.6 | 11.3 |
| | 3.0 | 2 | 3.1 | 3.2 | 14.5 |
| | 3.1 | 3 | 4.7 | 4.8 | 19.4 |
| | 3.2 | 1 | 1.6 | 1.6 | 21.0 |
| | 3.3 | 1 | 1.6 | 1.6 | 22.6 |
| | 3.6 | 3 | 4.7 | 4.8 | 27.4 |
| | 3.7 | 1 | 1.6 | 1.6 | 29.0 |
| | 3.8 | 2 | 3.1 | 3.2 | 32.3 |
| | 3.9 | 3 | 4.7 | 4.8 | 37.1 |
| | 4.2 | 1 | 1.6 | 1.6 | 38.7 |
| | 4.3 | 2 | 3.1 | 3.2 | 41.9 |
| | 4.5 | 1 | 1.6 | 1.6 | 43.5 |
| | 4.6 | 3 | 4.7 | 4.8 | 48.4 |
| | 4.7 | 2 | 3.1 | 3.2 | 51.6 |
| | 4.8 | 1 | 1.6 | 1.6 | 53.2 |
| | 4.9 | 1 | 1.6 | 1.6 | 54.8 |
| | 5.0 | 1 | 1.6 | 1.6 | 56.5 |
| | 5.1 | 1 | 1.6 | 1.6 | 58.1 |
| | 5.2 | 2 | 3.1 | 3.2 | 61.3 |
| | 5.3 | 2 | 3.1 | 3.2 | 64.5 |
| | 5.7 | 1 | 1.6 | 1.6 | 66.1 |
| | 6.0 | 1 | 1.6 | 1.6 | 67.7 |
| | 6.1 | 2 | 3.1 | 3.2 | 71.0 |
| | 6.2 | 1 | 1.6 | 1.6 | 72.6 |
| | 6.3 | 2 | 3.1 | 3.2 | 75.8 |
| | 6.4 | 1 | 1.6 | 1.6 | 77.4 |
| | 6.6 | 1 | 1.6 | 1.6 | 79.0 |
| | 6.7 | 4 | 6.3 | 6.5 | 85.5 |
| | 6.9 | 1 | 1.6 | 1.6 | 87.1 |
| | 7.2 | 2 | 3.1 | 3.2 | 90.3 |
| | 7.3 | 1 | 1.6 | 1.6 | 91.9 |
| | 7.4 | 2 | 3.1 | 3.2 | 95.2 |
| | 7.8 | 1 | 1.6 | 1.6 | 96.8 |
| | 7.9 | 1 | 1.6 | 1.6 | 98.4 |
| | 8.4 | 1 | 1.6 | 1.6 | 100.0 |
| | Total | 62 | 96.9 | 100.0 | |
| Missing | System | 2 | 3.1 | | |
| Total | | 64 | 100.0 | | |

**Appendix 7: How much students like working with JAVA Language**

| | | Frequency | Percent | Valid Percent | Cumulative Percent |
|---|---|---|---|---|---|
| Valid | .0 | 2 | 3.1 | 3.2 | 3.2 |
| | .3 | 1 | 1.6 | 1.6 | 4.8 |
| | .6 | 1 | 1.6 | 1.6 | 6.5 |
| | 1.0 | 1 | 1.6 | 1.6 | 8.1 |
| | 1.2 | 1 | 1.6 | 1.6 | 9.7 |
| | 1.4 | 1 | 1.6 | 1.6 | 11.3 |
| | 1.7 | 1 | 1.6 | 1.6 | 12.9 |
| | 1.9 | 1 | 1.6 | 1.6 | 14.5 |
| | 2.3 | 1 | 1.6 | 1.6 | 16.1 |
| | 2.4 | 1 | 1.6 | 1.6 | 17.7 |
| | 2.7 | 1 | 1.6 | 1.6 | 19.4 |
| | 2.8 | 2 | 3.1 | 3.2 | 22.6 |
| | 2.9 | 1 | 1.6 | 1.6 | 24.2 |
| | 3.1 | 1 | 1.6 | 1.6 | 25.8 |
| | 3.3 | 1 | 1.6 | 1.6 | 27.4 |
| | 4.2 | 1 | 1.6 | 1.6 | 29.0 |
| | 4.3 | 1 | 1.6 | 1.6 | 30.6 |
| | 4.5 | 1 | 1.6 | 1.6 | 32.3 |
| | 4.6 | 1 | 1.6 | 1.6 | 33.9 |
| | 4.7 | 1 | 1.6 | 1.6 | 35.5 |
| | 4.8 | 2 | 3.1 | 3.2 | 38.7 |
| | 5.0 | 1 | 1.6 | 1.6 | 40.3 |
| | 5.1 | 1 | 1.6 | 1.6 | 41.9 |
| | 5.2 | 2 | 3.1 | 3.2 | 45.2 |
| | 5.3 | 1 | 1.6 | 1.6 | 46.8 |
| | 5.4 | 2 | 3.1 | 3.2 | 50.0 |
| | 5.5 | 1 | 1.6 | 1.6 | 51.6 |
| | 5.7 | 1 | 1.6 | 1.6 | 53.2 |
| | 6.2 | 1 | 1.6 | 1.6 | 54.8 |
| | 6.4 | 1 | 1.6 | 1.6 | 56.5 |
| | 6.6 | 4 | 6.3 | 6.5 | 62.9 |
| | 6.7 | 1 | 1.6 | 1.6 | 64.5 |
| | 6.8 | 2 | 3.1 | 3.2 | 67.7 |
| | 6.9 | 4 | 6.3 | 6.5 | 74.2 |
| | 7.0 | 1 | 1.6 | 1.6 | 75.8 |
| | 7.2 | 2 | 3.1 | 3.2 | 79.0 |
| | 7.5 | 2 | 3.1 | 3.2 | 82.3 |
| | 7.8 | 1 | 1.6 | 1.6 | 83.9 |
| | 8.0 | 1 | 1.6 | 1.6 | 85.5 |
| | 8.2 | 2 | 3.1 | 3.2 | 88.7 |
| | 8.4 | 1 | 1.6 | 1.6 | 90.3 |
| | 8.6 | 1 | 1.6 | 1.6 | 91.9 |
| | 8.8 | 1 | 1.6 | 1.6 | 93.5 |
| | 9.0 | 1 | 1.6 | 1.6 | 95.2 |
| | 9.1 | 2 | 3.1 | 3.2 | 98.4 |
| | 9.6 | 1 | 1.6 | 1.6 | 100.0 |
| | Total | 62 | 96.9 | 100.0 | |
| Missing | System | 2 | 3.1 | | |
| Total | | 64 | 100.0 | | |

## Students' Knowledge in C++ Language

**Appendix 8: Students' self-rated skill in Programming in C++ Language**

| | | Frequency | Percent | Valid Percent | Cumulative Percent |
|---|---|---|---|---|---|
| Valid | .1 | 1 | 1.6 | 4.0 | 4.0 |
| | .2 | 2 | 3.1 | 8.0 | 12.0 |
| | .5 | 1 | 1.6 | 4.0 | 16.0 |
| | .6 | 1 | 1.6 | 4.0 | 20.0 |
| | .7 | 1 | 1.6 | 4.0 | 24.0 |
| | .9 | 2 | 3.1 | 8.0 | 32.0 |
| | 1.5 | 1 | 1.6 | 4.0 | 36.0 |
| | 1.6 | 1 | 1.6 | 4.0 | 40.0 |
| | 1.8 | 1 | 1.6 | 4.0 | 44.0 |
| | 2.0 | 1 | 1.6 | 4.0 | 48.0 |
| | 2.3 | 1 | 1.6 | 4.0 | 52.0 |
| | 2.5 | 1 | 1.6 | 4.0 | 56.0 |
| | 3.3 | 1 | 1.6 | 4.0 | 60.0 |
| | 3.4 | 1 | 1.6 | 4.0 | 64.0 |
| | 4.1 | 1 | 1.6 | 4.0 | 68.0 |
| | 4.6 | 1 | 1.6 | 4.0 | 72.0 |
| | 4.9 | 1 | 1.6 | 4.0 | 76.0 |
| | 5.0 | 1 | 1.6 | 4.0 | 80.0 |
| | 6.9 | 1 | 1.6 | 4.0 | 84.0 |
| | 7.4 | 1 | 1.6 | 4.0 | 88.0 |
| | 7.9 | 1 | 1.6 | 4.0 | 92.0 |
| | 8.0 | 1 | 1.6 | 4.0 | 96.0 |
| | 8.3 | 1 | 1.6 | 4.0 | 100.0 |
| | Total | 25 | 39.1 | 100.0 | |
| Missing | System | 39 | 60.9 | | |
| Total | | 64 | 100.0 | | |

**Appendix 9: How much students like programming using C++ language**

| | | Frequency | Percent | Valid Percent | Cumulative Percent |
|---|---|---|---|---|---|
| Valid | .3 | 1 | 1.6 | 4.0 | 4.0 |
| | .5 | 2 | 3.1 | 8.0 | 12.0 |
| | .6 | 1 | 1.6 | 4.0 | 16.0 |
| | 1.5 | 1 | 1.6 | 4.0 | 20.0 |
| | 1.8 | 1 | 1.6 | 4.0 | 24.0 |
| | 1.9 | 1 | 1.6 | 4.0 | 28.0 |
| | 2.8 | 1 | 1.6 | 4.0 | 32.0 |
| | 3.4 | 1 | 1.6 | 4.0 | 36.0 |
| | 3.6 | 1 | 1.6 | 4.0 | 40.0 |
| | 4.0 | 1 | 1.6 | 4.0 | 44.0 |
| | 4.8 | 1 | 1.6 | 4.0 | 48.0 |
| | 4.9 | 1 | 1.6 | 4.0 | 52.0 |
| | 5.1 | 1 | 1.6 | 4.0 | 56.0 |
| | 5.2 | 2 | 3.1 | 8.0 | 64.0 |
| | 5.4 | 1 | 1.6 | 4.0 | 68.0 |
| | 6.2 | 1 | 1.6 | 4.0 | 72.0 |
| | 6.6 | 1 | 1.6 | 4.0 | 76.0 |
| | 7.2 | 1 | 1.6 | 4.0 | 80.0 |
| | 7.3 | 1 | 1.6 | 4.0 | 84.0 |
| | 8.1 | 1 | 1.6 | 4.0 | 88.0 |
| | 8.2 | 1 | 1.6 | 4.0 | 92.0 |
| | 8.3 | 1 | 1.6 | 4.0 | 96.0 |
| | 8.6 | 1 | 1.6 | 4.0 | 100.0 |
| | Total | 25 | 39.1 | 100.0 | |
| Missing | System | 39 | 60.9 | | |
| Total | | 64 | 100.0 | | |

## Students self-rated skill in debugging programs

**Appendix 10: Students and their skill debugging their own program**

| | | Frequency | Percent | Valid Percent | Cumulative Percent |
|---|---|---|---|---|---|
| Valid | 1.0 | 1 | 1.6 | 1.6 | 1.6 |
| | 2 | 1 | 1.6 | 1.6 | 3.3 |
| | 2 | 1 | 1.6 | 1.6 | 4.9 |
| | 2.2 | 1 | 1.6 | 1.6 | 6.6 |
| | 2.3 | 1 | 1.6 | 1.6 | 8.2 |
| | 2.5 | 1 | 1.6 | 1.6 | 9.8 |
| | 2.6 | 1 | 1.6 | 1.6 | 11.5 |
| | 2.8 | 1 | 1.6 | 1.6 | 13.1 |
| | 3.2 | 2 | 3.1 | 3.3 | 16.4 |
| | 3.4 | 2 | 3.1 | 3.3 | 19.7 |
| | 3.5 | 2 | 3.1 | 3.3 | 23.0 |
| | 3.6 | 1 | 1.6 | 1.6 | 24.6 |
| | 3.8 | 1 | 1.6 | 1.6 | 26.2 |
| | 4.1 | 1 | 1.6 | 1.6 | 27.9 |
| | 4.2 | 1 | 1.6 | 1.6 | 29.5 |
| | 4.3 | 1 | 1.6 | 1.6 | 31.1 |
| | 4.4 | 2 | 3.1 | 3.3 | 34.4 |
| | 4.5 | 1 | 1.6 | 1.6 | 36.1 |
| | 4.6 | 3 | 4.7 | 4.9 | 41.0 |
| | 4.8 | 1 | 1.6 | 1.6 | 42.6 |
| | 4.9 | 3 | 4.7 | 4.9 | 47.5 |
| | 5.0 | 2 | 3.1 | 3.3 | 50.8 |
| | 5.1 | 2 | 3.1 | 3.3 | 54.1 |
| | 5.2 | 1 | 1.6 | 1.6 | 55.7 |
| | 5.3 | 1 | 1.6 | 1.6 | 57.4 |
| | 5.5 | 3 | 4.7 | 4.9 | 62.3 |
| | 5.6 | 1 | 1.6 | 1.6 | 63.9 |
| | 5.7 | 2 | 3.1 | 3.3 | 67.2 |
| | 5.9 | 1 | 1.6 | 1.6 | 68.9 |
| | 6.0 | 1 | 1.6 | 1.6 | 70.5 |
| | 6.2 | 1 | 1.6 | 1.6 | 72.1 |
| | 6.3 | 1 | 1.6 | 1.6 | 73.8 |
| | 6.4 | 2 | 3.1 | 3.3 | 77.0 |
| | 6.5 | 1 | 1.6 | 1.6 | 78.7 |
| | 6.6 | 1 | 1.6 | 1.6 | 80.3 |
| | 6.9 | 1 | 1.6 | 1.6 | 82.0 |
| | 7.0 | 1 | 1.6 | 1.6 | 83.6 |
| | 7.2 | 2 | 3.1 | 3.3 | 86.9 |
| | 7.3 | 1 | 1.6 | 1.6 | 88.5 |
| | 7.4 | 2 | 3.1 | 3.3 | 91.8 |
| | 7.5 | 2 | 3.1 | 3.3 | 95.1 |
| | 7.6 | 1 | 1.6 | 1.6 | 96.7 |
| | 7.8 | 1 | 1.6 | 1.6 | 98.4 |
| | 7.9 | 1 | 1.6 | 1.6 | 100.0 |
| | Total | 61 | 95.3 | 100.0 | |
| Missing | System | 3 | 4.7 | | |
| Total | | 64 | 100.0 | | |

**Appendix 11: Students and their skill debugging other's people program**

| | | Frequency | Percent | Valid Percent | Cumulative Percent |
|---|---|---|---|---|---|
| Valid | 1.2 | 1 | 1.6 | 1.6 | 1.6 |
| | 1.3 | 1 | 1.6 | 1.6 | 3.3 |
| | 1.4 | 1 | 1.6 | 1.6 | 4.9 |
| | 1.5 | 1 | 1.6 | 1.6 | 6.6 |
| | 1.6 | 1 | 1.6 | 1.6 | 8.2 |
| | 1.7 | 1 | 1.6 | 1.6 | 9.8 |
| | 2.0 | 2 | 3.1 | 3.3 | 13.1 |
| | 2.1 | 2 | 3.1 | 3.3 | 16.4 |
| | 2.2 | 2 | 3.1 | 3.3 | 19.7 |
| | 2.3 | 1 | 1.6 | 1.6 | 21.3 |
| | 2.6 | 1 | 1.6 | 1.6 | 23.0 |
| | 2.7 | 2 | 3.1 | 3.3 | 26.2 |
| | 2.8 | 1 | 1.6 | 1.6 | 27.9 |
| | 3.0 | 1 | 1.6 | 1.6 | 29.5 |
| | 3.1 | 1 | 1.6 | 1.6 | 31.1 |
| | 3.2 | 1 | 1.6 | 1.6 | 32.8 |
| | 3.3 | 1 | 1.6 | 1.6 | 34.4 |
| | 3.4 | 1 | 1.6 | 1.6 | 36.1 |
| | 3.6 | 1 | 1.6 | 1.6 | 37.7 |
| | 3.8 | 2 | 3.1 | 3.3 | 41.0 |
| | 4.2 | 1 | 1.6 | 1.6 | 42.6 |
| | 4.5 | 1 | 1.6 | 1.6 | 44.3 |
| | 4.8 | 2 | 3.1 | 3.3 | 47.5 |
| | 4.9 | 2 | 3.1 | 3.3 | 50.8 |
| | 5.0 | 2 | 3.1 | 3.3 | 54.1 |
| | 5.1 | 2 | 3.1 | 3.3 | 57.4 |
| | 5.3 | 1 | 1.6 | 1.6 | 59.0 |
| | 5.6 | 1 | 1.6 | 1.6 | 60.7 |
| | 5.7 | 1 | 1.6 | 1.6 | 62.3 |
| | 5.8 | 1 | 1.6 | 1.6 | 63.9 |
| | 5.9 | 1 | 1.6 | 1.6 | 65.6 |
| | 6.0 | 2 | 3.1 | 3.3 | 68.9 |
| | 6.1 | 3 | 4.7 | 4.9 | 73.8 |
| | 6.2 | 1 | 1.6 | 1.6 | 75.4 |
| | 6.4 | 2 | 3.1 | 3.3 | 78.7 |
| | 6.7 | 2 | 3.1 | 3.3 | 82.0 |
| | 6.8 | 1 | 1.6 | 1.6 | 83.6 |
| | 6.9 | 1 | 1.6 | 1.6 | 85.2 |
| | 7.2 | 2 | 3.1 | 3.3 | 88.5 |
| | 7.4 | 1 | 1.6 | 1.6 | 90.2 |
| | 7.5 | 1 | 1.6 | 1.6 | 91.8 |
| | 7.8 | 1 | 1.6 | 1.6 | 93.4 |
| | 8.0 | 1 | 1.6 | 1.6 | 95.1 |
| | 9.1 | 1 | 1.6 | 1.6 | 96.7 |
| | 9.7 | 1 | 1.6 | 1.6 | 98.4 |
| | 9.9 | 1 | 1.6 | 1.6 | 100.0 |
| | Total | 61 | 95.3 | 100.0 | |
| Missing | System | 3 | 4.7 | | |
| Total | | 64 | 100.0 | | |

## CODE-REVIEW EXERCISE

**Appendix 12: Performance in the Bug-finding task – prize distribution**[4]

| Bug Found | False Bugs | Part. # | Total Score | Differs | Prog. Study | Prize |
|---|---|---|---|---|---|---|
| 10 | 5 | 9 | 27 | 22 | SE | £30 |
| 8 | 2 | 52 | 25 | 23 | CS | £30 |
| 8 | 0 | 5 | 22 | 22 | CS | £20 |
| 8 | 2 | 82 | 20 | 18 | CS | £10 |
| 7 | 4 | 31 | 17 | 13 | CS | X |
| 8 | 1 | 84 | 17 | 16 | CS | X |
| 6 | 3 | 48 | 15 | 12 | SE | £20 |
| 6 | 1 | 98 | 15 | 14 | CS | X |
| 5 | 4 | 75 | 14 | 10 | CS | X |
| *6* | *5* | *93* | *14* | *9* | *CS* | *X* |
| *5* | *5* | *110* | *13* | *8* | *SE* | *£10* |
| *5* | *1* | *79* | *12* | *11* | *CS* | *X* |
| 5 | 3 | 104 | 12 | 9 | CS | X |
| 5 | 4 | 1 | 11 | 7 | SE | X |
| 5 | 5 | 12 | 11 | 6 | SE | X |
| 5 | 2 | 45 | 11 | 9 | CS | X |
| 4 | 6 | 53 | 11 | 5 | CS | X |
| 4 | 0 | 69 | 11 | 11 | SE | X |
| *4* | *13* | *91* | *11* | *-2* | *IS* | *£30* |
| 4 | 4 | 27 | 10 | 6 | SE | X |
| 4 | 6 | 60 | 10 | 4 | CS | X |
| 4 | 2 | 13 | 9 | 7 | CS | X |
| 4 | 6 | 41 | 9 | 3 | CS | X |
| 4 | 0 | 56 | 9 | 9 | CS | X |
| *3* | *3* | *112* | *9* | *6* | *CS* | *X* |
| 4 | 4 | 14 | 8 | 4 | SE | X |
| 4 | 0 | 21 | 8 | 8 | IS | £20 |
| 4 | 2 | 29 | 8 | 6 | CS | X |
| 3 | 4 | 32 | 8 | 4 | CS | X |
| 3 | 10 | 67 | 8 | -2 | IS | £10 |
| *4* | *3* | *100* | *8* | *-5* | *CS* | *X* |
| *4* | *3* | *101* | *8* | *-5* | *CS* | *X* |
| *3* | *2* | *107* | *8* | *6* | *SE* | *X* |
| *3* | *0* | *108* | *8* | *8* | *SE* | *X* |
| *4* | *2* | *111* | *8* | *6* | *SE* | *X* |
| 3 | 3 | 16 | 7 | 4 | CS | X |
| 3 | 0 | 37 | 7 | 7 | CS | X |
| *3* | *2* | *102* | *7* | *5* | *SE* | *X* |
| 3 | 5 | 55 | 6 | 1 | CS | X |
| 3 | 5 | 72 | 6 | 1 | CS | X |
| 3 | 3 | 81 | 6 | 3 | CS | X |
| 2 | 13 | 15 | 5 | -8 | IS | X |
| 2 | 1 | 17 | 5 | 4 | CS | X |
| 2 | 3 | 36 | 5 | 2 | CS | X |
| 2 | 6 | 68 | 5 | -1 | IS | X |
| 2 | 0 | 2 | 4 | 4 | SE | X |
| *2* | *3* | *4* | *4* | *1* | *CS* | *X* |
| 2 | 3 | 8 | 4 | 1 | CS | X |
| 2 | 0 | 22 | 4 | 4 | IS | X |
| 2 | 2 | 24 | 4 | 2 | IS | X |

[4] Although in the Chapter 6 the results are described only for those who took part either in the 3 tasks, or in the bug-finding and MBTI, in this table are presented the data from all 88 participants of this task as there were a winner from the competition prize who did not took part in the others task. However to facilitate the visualisation of those who have their results described in the chapter 6, those who did not realised the three or two tasks have their data typed in *italic.*

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 8 | 28 | 4 | -4 | CS | X |
| 2 | 3 | 30 | 4 | 1 | SE | X |
| 2 | 2 | 54 | 4 | 2 | CS | X |
| 2 | 0 | 57 | 4 | 4 | CS | X |
| 2 | 1 | 65 | 4 | 3 | IS | X |
| 2 | 0 | 66 | 4 | 4 | CS | X |
| 2 | 3 | 70 | 4 | 1 | CS | X |
| 1 | 4 | 77 | 4 | 0 | IS | X |
| 2 | 1 | 78 | 4 | 3 | IS | X |
| 2 | 8 | 88 | 4 | -4 | IS | X |
| 2 | 4 | 95 | 4 | 0 | CS | X |
| *2* | *0* | *105* | *4* | *4* | *CS* | *X* |
| *1* | *4* | *109* | *4* | *0* | *SE* | *X* |
| 2 | 0 | 113 | 4 | 4 | IS | X |
| *1* | *3* | *39* | *3* | *0* | *IS* | *X* |
| *1* | *6* | *7* | *2* | *-4* | *IS* | *X* |
| 1 | 4 | 42 | 2 | -2 | CS | X |
| *1* | *5* | *51* | *2* | *-3* | *CS* | *X* |
| 1 | 4 | 59 | 2 | -2 | IS | X |
| 1 | 4 | 64 | 2 | -2 | IS | X |
| 1 | 4 | 74 | 2 | -2 | IS | X |
| 1 | 2 | 80 | 2 | 0 | CS | X |
| 1 | 0 | 97 | 2 | 2 | IS | X |
| *1* | *0* | *103* | *2* | *2* | *IS* | *X* |
| *1* | *8* | *115* | *2* | *-6* | *IS* | *X* |
| 0 | 3 | 6 | 0 | -3 | IS | X |
| 0 | 6 | 11 | 0 | -6 | IS | X |
| 0 | 2 | 33 | 0 | -2 | CS | X |
| 0 | 2 | 44 | 0 | -2 | CS | X |
| 0 | 1 | 50 | 0 | -1 | IS | X |
| 0 | 7 | 73 | 0 | -7 | IS | X |
| 0 | 6 | 76 | 0 | -6 | IS | X |
| *0* | *3* | *86* | *0* | *-3* | *IS* | *X* |
| 0 | 2 | 92 | 0 | -2 | IS | X |
| *0* | *2* | *106* | *0* | *-2* | *SE* | *X* |
| *0* | *7* | *114* | *0* | *-7* | *IS* | *X* |
| *0* | *0* | *18* | . | . | *IS* | *X* |
| *0* | *0* | *19* | . | . | *CS* | *X* |

**Appendix 13: Percentage of False bugs found in the task**

| False bugs | Frequency | Percent | Valid Percent | Cumulative Percent |
|---|---|---|---|---|
| 0 | 11 | 17.2 | 17.2 | 17.2 |
| 1 | 6 | 9.4 | 9.4 | 26.6 |
| 2 | 10 | 15.6 | 15.6 | 42.2 |
| 3 | 9 | 14.1 | 14.1 | 56.3 |
| 4 | 12 | 18.8 | 18.8 | 75.0 |
| 5 | 4 | 6.3 | 6.3 | 81.3 |
| 6 | 6 | 9.4 | 9.4 | 90.6 |
| 7 | 2 | 3.1 | 3.1 | 93.8 |
| 8 | 2 | 3.1 | 3.1 | 96.9 |
| 10 | 1 | 1.6 | 1.6 | 98.4 |
| 13 | 1 | 1.6 | 1.6 | 100.0 |
| Total | 64 | 100.0 | 100.0 | |

# MBTI

**Appendix 14: MBTI types within 2nd year students of Computing Science**

| MBTI Type | Frequency whole sample | Percent whole sample |
|---|---|---|
| ENFJ | 2 | 2.6 |
| ENFP | 3 | 3.9 |
| ENTJ | 3 | 3.9 |
| ENTP | 10 | 13.0 |
| ESFJ | 5 | 6.5 |
| ESFP | 3 | 3.9 |
| ESTJ | 5 | 6.5 |
| ESTP | 7 | 9.1 |
| INFJ | 3 | 3.9 |
| INFP | 3 | 3.9 |
| INTJ | 5 | 6.5 |
| INTP | 5 | 6.5 |
| ISFJ | 7 | 9.1 |
| ISFP | 3 | 3.9 |
| ISTJ | 5 | 6.5 |
| ISTP | 8 | 10.4 |
| Total | 77 | 100.0 |

**Appendix 15: MBTI Types Valid Data***

| MBTI Type | Frequency of valid data | Percent of valid data |
|---|---|---|
| ENFJ | 2 | 3.1 |
| ENFP | 2 | 3.1 |
| ENTJ | 3 | 4.7 |
| ENTP | 10 | 15.6 |
| ESFJ | 5 | 7.8 |
| ESFP | 3 | 4.7 |
| ESTJ | 5 | 7.8 |
| ESTP | 6 | 9.4 |
| INFJ | 3 | 4.7 |
| INFP | 1 | 1.6 |
| INTJ | 4 | 6.3 |
| INTP | 3 | 4.7 |
| ISFJ | 5 | 7.8 |
| ISFP | 2 | 3.1 |
| ISTJ | 4 | 6.3 |
| ISTP | 6 | 9.4 |
| Total | 64 | 100.0 |

*sample used in the Statistical Analysis within Code Review Task and Questionnaire

**Appendix 16: MBTI Types and the types' scales[5]**

| ID | Type | EI | SN | TF | JP |
|---|---|---|---|---|---|
| 1 | ISTJ | 33 | 70 | 85 | 95 |
| 2 | ISTJ | 42 | 53 | 86 | 56 |
| 5 | ENTP | 58 | 16 | 74 | 33 |
| 6 | ESTJ | 57 | 72 | 71 | 69 |
| 8 | ESFP | 63 | 51 | 49 | 35 |
| 9 | INTJ | 9 | 44 | 66 | 78 |
| 11 | ESFJ | 69 | 60 | 49 | 82 |
| 12 | ENTP | 58 | 47 | 57 | 11 |
| 13 | ISFJ | 34 | 64 | 41 | 69 |
| 14 | ENTP | 84 | 35 | 52 | 48 |
| 15 | INTJ | 10 | 47 | 68 | 60 |
| 16 | ISTP | 18 | 55 | 78 | 43 |
| 17 | INFP | 34 | 9 | 44 | 35 |
| 21 | ESTP | 52 | 53 | 66 | 38 |
| 22 | ESTJ | 87 | 87 | 52 | 65 |
| 27 | ENTP | 58 | 24 | 71 | 30 |
| 28 | ESFP | 87 | 51 | 21 | 44 |
| 29 | ENTP | 94 | 25 | 63 | 37 |
| 30 | INTP | 28 | 27 | 77 | 17 |
| 31 | ENTJ | 58 | 27 | 88 | 71 |
| 32 | ENFP | 82 | 27 | 44 | 22 |
| 33 | ESTJ | 60 | 55 | 62 | 64 |
| 36 | ISFJ | 18 | 96 | 41 | 71 |
| 37 | ISTP | 40 | 66 | 74 | 13 |
| 41 | ESFJ | 52 | 58 | 38 | 71 |
| 42 | ISTJ | 36 | 64 | 69 | 60 |
| 44 | ENTP | 99 | 18 | 89 | 0 |
| 45 | ESFJ | 61 | 75 | 28 | 56 |
| 48 | ISTP | 1 | 58 | 58 | 49 |
| 50 | INTJ | 13 | 45 | 72 | 71 |
| 52 | INFJ | 28 | 13 | 49 | 67 |
| 53 | INFJ | 46 | 40 | 44 | 80 |
| 54 | INTJ | 40 | 29 | 66 | 75 |
| 55 | ESTJ | 57 | 91 | 66 | 56 |
| 56 | INTP | 49 | 49 | 77 | 17 |
| 57 | INFJ | 37 | 44 | 23 | 60 |
| 59 | ENTP | 76 | 29 | 62 | 46 |
| 60 | ENTJ | 79 | 44 | 78 | 71 |
| 64 | ESTP | 66 | 87 | 78 | 44 |
| 65 | ESTP | 78 | 68 | 78 | 29 |
| 66 | ISFP | 42 | 53 | 13 | 21 |
| 67 | ENTP | 97 | 16 | 54 | 10 |
| 68 | ISTP | 9 | 79 | 54 | 40 |
| 69 | INTP | 22 | 42 | 69 | 32 |
| 70 | ENTJ | 99 | 22 | 86 | 65 |
| 72 | ISFJ | 45 | 79 | 46 | 53 |
| 73 | ISFP | 18 | 75 | 44 | 33 |
| 74 | ENTP | 78 | 47 | 69 | 38 |

[5] The scale is formatted as follows: from 0 to 49 means that a person has one of this types as his/her preferences (depending on which column the number is located) – I, N, F or P. From 51 – 100, means that a person has the preferences for one (or more) of this types: E, S, T, J.

| | | | | | |
|---|---|---|---|---|---|
| 75 | ESTP | 51 | 53 | 72 | 46 |
| 76 | ESFJ | 78 | 89 | 10 | 62 |
| 77 | ESFP | 57 | 62 | 28 | 19 |
| 78 | ENFP | 79 | 25 | 28 | 10 |
| 80 | ESTJ | 60 | 62 | 89 | 60 |
| 81 | ISFJ | 40 | 91 | 41 | 89 |
| 82 | ENTP | 78 | 42 | 85 | 5 |
| 84 | ISTJ | 28 | 91 | 54 | 67 |
| 88 | ENFJ | 57 | 31 | 18 | 87 |
| 92 | ESFJ | 58 | 74 | 41 | 65 |
| 94 | ISTP | 27 | 57 | 55 | 43 |
| 95 | ENFJ | 57 | 42 | 31 | 53 |
| 97 | ISFJ | 49 | 64 | 41 | 80 |
| 98 | ESTP | 76 | 74 | 95 | 21 |
| 105 | ISTP | 24 | 74 | 66 | 0 |
| 114 | ESTP | 60 | 75 | 60 | 44 |

# ADDITIONAL DATA – STUDENTS' FIRST YEAR MARKINGS

**Appendix 17: Students CSC131 and CSC132 markings**

| ID | CSC131 | CSC132 |
|---|---|---|
| 1 | 85 | 85 |
| 2 | 53 | 45 |
| 5 | 92 | 91 |
| 6 | 36 | 40 |
| 8 | 53 | 51 |
| 9 | 89 | 77 |
| 12 | 71 | 66 |
| 13 | 75 | 79 |
| 14 | 71 | 69 |
| 16 | 85 | 74 |
| 17 | 63 | 75 |
| 27 | 62 | 50 |
| 28 | 42 | 27 |
| 29 | 64 | 55 |
| 30 | 51 | 54 |
| 31 | 75 | 86 |
| 32 | 55 | 51 |
| 33 | 50 | 40 |
| 42 | 53 | 56 |
| 44 | 64 | 57 |
| 45 | 68 | 58 |
| 48 | 87 | 88 |
| 50 | 27 | . |
| 52 | 75 | 67 |
| 53 | 47 | 40 |
| 54 | 74 | 65 |
| 55 | 83 | 72 |
| 56 | 76 | 65 |
| 57 | 72 | 81 |
| 60 | 79 | 76 |
| 64 | 24 | 34 |
| 65 | 55 | 25 |
| 66 | 56 | 55 |
| 69 | 70 | 75 |
| 70 | 68 | 62 |
| 72 | 77 | 59 |
| 75 | 63 | 56 |
| 80 | 72 | 58 |
| 81 | 59 | 55 |
| 82 | 82 | 84 |
| 84 | 84 | 76 |
| 94 | 48 | 68 |
| 95 | 74 | 73 |
| 98 | 84 | 66 |
| 105 | 67 | 63 |
| 114 | 45 | 49 |

## 10 REFERENCES

Bernstein, D. A., Roy, E. J., Srull, T. K., & Wickens, C. D. (1991). Personality. In Psychology (2nd ed., pp. xxiv, 740, 129). Boston: Houghton Mifflin.

Bischof, L. J. (1970). Interpreting personality theories (2d ed.): Harper & Row.

Bishop-Clark, C. (1995). Cognitive style, personality, and computer programming. computers in human behavior, 11, 241-260.

Bishop-Clark, C., & Wheeler, D. (1994). The Myers-Briggs Personality Type and its Relationship to Computer Programming. Journal of Research on Computing Education, 26, 358-370.

Boehm, B. W. (1981). Software engineering economics. Englewood Cliffs, N.J.: Prentice-Hall.

Bray, I. (2002). An introduction to requirements engineering. Harlow: Addison-Wesley.

Brunas-Wagstaff, J. (1998). Personality: a cognitive approach. London ; New York: Routledge.

Capretz, L. F. (2003). Personality types in software engineering. International Journal Human-Computer Studies, 58, 207-214.

Cusumano, M. A., & Selby, R. W. (1995). Microsoft secrets: how the world's most powerful software company creates technology, shapes markets, and manages people. New York: Free Press.

Devito, A. J. (1985). Review of Myers-Briggs Type Indicator. In J. Mitchell (Ed.), The Ninth Mental measurements yearbook (pp. 10301032): Lincoln, Neb.

Edwards, J. A., Lanning, K., & Hooker, K. (2002). The MBTI and Social Information Processing: An Incremental Validity Study. Journal of Personality Assessment, 78, 432-450.

Furnham, A. (1994). Personality at work : the role of individual differences in the workplace. New York: Routledge.

Furnham, A., Jackson, C. J., & Miller, T. (1999). Personality, learning style and work performance. Personality & Individual Differences, 27, 1113-1122.

Ghezzi, C., Jazayeri, M., & Mandrioli, D. (2003). Fundamentals of software engineering (2nd ed.). Upper Saddle River, N.J.: Prentice Hall.

Gould, J. D. (1975). Some psychological evidence on how people debug computer program. International Journal of Man-Machine Studies, 7, 151-182.

Gould, J. D., & Drongowski, P. (1974). An Explanatory Study of Computer Program Debugging. Human Factors, 16, 258-277.

Myers, I. B. (1990). Introduction to type: a description of the theory and applications of the Myers-Briggs Type Indicator (12th ed.). Palo Alto, CA: Consulting Psychologists Press.

Myers, I. B., & McCaulley, M. H. (1985). Manual: a guide to the development and use of the Myers-Briggs Type Indicator (5th ed.). Palo Alto, Calif.: Consulting Psychologists Press.

Myers, I.B. and P.B. Myers (1995). Gifts differing: understanding personality type. Palo Alto, Calif.: Davies-Black Pub.

Naur, P. (1992). Computing, a human activity. New York Reading, Mass.: ACM Press; Addison-Wesley Pub. Co

Oakes, D. W., Ferris, G. R., Martocchio, J. J., Buckley, M. R., & Broach, D. (2001). Cognitive ability and personality of training program skill acquisition and job performance. Journal of Business and psychology, 15, 523-548.

Pressman, R. S. (1992). Chapter 19 - Software testing strategies, pp.654-658. In Software engineering : a practitioner's approach (3rd ed., pp. xxi, 793). New York: McGraw-Hill.

Shneiderman, B. (1980). Software psychology: human factors in computer and information systems: Cambridge, Mass. : Winthrop Publishers.

Smither, R. D. (1998). The psychology of work and human performance (3rd ed.). New York: Longman.

Sommerville, I. (1992). Software engineering (4th ed.). Reading, Mass.: Addison-Wesley Pub. Co.

Weinberg, G. M. (1971-1998). The psychology of computer programming. New York,: Van Nostrand Reinhold.

Winder, R., & Roberts, G. (1999). The Programming Process (chapter 9). In Developing Java software (2nd ed., pp. 246-268). New York: Wiley.